CA-MetaCOBOL™ +

Program Development Guide
CA-DATACOM / DB

Release 1.1

R003M+11DGP

Release 1.1, January, 1992

Contents

TM

1. About This Manual

1.1 Purpose

This manual describes how to use the CA-MetaCOBOL+ CA-DATACOM/DB Facility to develop programs that make full use of the functions and features of the CA-DATACOM/DB environment.

This manual assumes familiarity with the COBOL language. It provides descriptions, examples, and instructions on how to use the CA-DATACOM/DB Facility Data Manipulation Language (DML) to store and retrieve information from the CA-DATACOM/DB database management system.

1.2 Organization

The CA-MetaCOBOL+ *Program Development Guide - CA-DATACOM/DB* is the companion to the CA-MetaCOBOL+ *Program Development Reference - CA-DATACOM/DB*.

Chapter	**Description**
1	Discusses the purpose of the manual, gives a list of CA-MetaCOBOL+ documentation, and explains notation conventions for CA-MetaCOBOL+.
2	Describes the basic components needed to understand this manual, including the CA-DATACOM/DB Facility and the CA-MetaCOBOL+ Translator.

Chapter	Description
3	Explains the basic functions of CA-DATACOM/DB and techniques for using it.
4	Explains the concepts and organization of the CA-DATADICTIONARY.
5	Describes how to begin using CA-DATACOM/DB and CA-DATADICTIONARY.
6	Discusses how to use the CA-DATACOM/DB Data Manipulation Language.
7	Provides a sample program for using the CA-DATACOM/DB Facility.
8	Gives statement formats for CA-DATACOM/DB.
Index	Gives page references for important topics covered in this manual.

1.3 Publications

In addition to this manual, the following publications are supplied with CA-MetaCOBOL+.

Title	Contents
Introduction to CA-MetaCOBOL+	Introduces the CA-MetaCOBOL+ Work Bench, Structured Programming Facility, Quality Assurance Facility, CA-DATACOM/DB Facility, Macro Facility, Panel Definition Facility, the Online Programming Language and the Chart Writer Language.
Installation Guide - MVS	Explains how to install CA-MetaCOBOL+ in the MVS environment.
CA-ACTIVATOR Installation Supplement - MVS	Explains how to install CA-MetaCOBOL+ in the MVS environment using CA-ACTIVATOR.
Installation Guide - VSE	Explains how to install CA-MetaCOBOL+ in the VSE environment.

Title	Contents
Supplement - MVS	MVS environment using CA-ACTIVATOR.
Installation Guide - VSE	Explains how to install CA-MetaCOBOL+ in the VSE environment.
Installation Guide - CMS	Explains how to install CA-MetaCOBOL+ in the CMS environment.
User Guide	Explains how to customize, get started, and use CA-MetaCOBOL+. Includes information on keyword expansion, the CA-MetaCOBOL+ translator, and CA macro sets and programs.
Structured Programming Guide	Introduces the Structured Programming Facility. Includes information on creating, testing, and maintaining structured programs.
Macro Facility Tutorial	Introduces the Macro Facility. Includes information on writing basic macros, model programming, macro writing techniques, and debugging.
Macro Facility Reference	Includes detailed information on the program flow of the CA-MetaCOBOL+ macro translator, macro format, definition of comments, macro nesting, macro prototypes, symbolic words, and model programming.
Quality Assurance Guide	Introduces the Quality Assurance Facility. Includes all the necessary information to perform quality assurance testing on COBOL source programs.
Program Development Reference CA-DATACOM/DB	Contains all CA-DATACOM/DB Facility constructs and statements.
Panel Definition Facility Command Reference	Contains all Panel Defintion Facility commands.
Panel Definition Facility User Guide	Includes all the information necessary to create, edit, duplicate, rename, delete, index, and print panel definitions and members. Also describes how to generate BMS source.
Online Programming Language Reference	Contains all Online Programming Language statements.
Online Programming Language Guide	Provides further instruction for using Online Programming Language statements.

All manuals are updated as required. Instructions accompany each update package.

1.4 Notation Conventions

The following conventions are used in the command formats throughout this manual:

UPPERCASE	is used to display commands or keywords you must code exactly as shown.
lowercase italic	is used to display information you must supply. For example, DASD space parameters may appear as *xxxxxxx xxxxxxx xxxxxxx*.
<u>Underscores</u>	either show a default value in a screen image or represent the highlighting of a word in a screen image.
Brackets [ ]	mean that you can select one of the items enclosed by the brackets; none of the enclosed items is required.
Braces { }	mean that you must select one of the items enclosed by the braces.
Vertical Bar |	separates options. One vertical bar separates two options, two vertical bars separate three options, and so on. You must select one of the options.
Ellipsis . . .	means that you can repeat the word or clause that immediately precedes the ellipsis.

1.5 Summary of Revisions

The following modifications and enhancements have been made to this manual for Version 1.1 of MetaCOBOL+.

Minor technical and editorial changes have been made throughout the manual.

2. Introduction

This manual describes how to code CA-DATACOM/DB Facility statements in application programs to access CA-DATACOM/DB databases. The application tasks included in this manual illustrate common usage of frequently used CA-DATACOM/DB Facility statements. For a complete description of all CA-DATACOM/DB Facility statements, refer to the CA-MetaCOBOL+ *Program Development Reference - CA-DATACOM/DB* manual.

This chapter presents an overview of the following components of CA-MetaCOBOL+:

- CA-DATACOM/DB Facility (Section 2.1), herein referred to as the DATACOM/DB Facility
- CA-MetaCOBOL+ Translator (Section 2.2)
- CA-MetaCOBOL+ Dialogs (Section 2.3)
- Structured Programming (SP) Facility (Section 2.4)
- CA-MetaCOBOL+ software requirements (Section 2.5)

2.1 The CA-DATACOM/DB Facility

Data stored by CA-DATACOM/DB can be accessed with standard **CALL** statements from COBOL application programs. However, these "native" calls are typically complex and thus difficult to code and debug.

On the other hand, the CA-DATACOM/DB Facility's high-level, COBOL-like statements are much easier to code. With the CA-DATACOM/DB Facility, application programmers do not need to be concerned with coding call-level accesses to CA-DATACOM/DB. This is because COBOL programs can be input to the CA-MetaCOBOL+ Translator, which generates correctly formatted calls from embedded CA-DATACOM/DB Facility statements.

The CA-DATACOM/DB Facility Data Manipulation Language (DML) provides COBOL-like statements for index searching, record retrieval, and updating. The CA-DATACOM/DB Facility also provides statements for extracting dataviews from CA-DATADICTIONARY.

2.2 CA-MetaCOBOL+ Translator

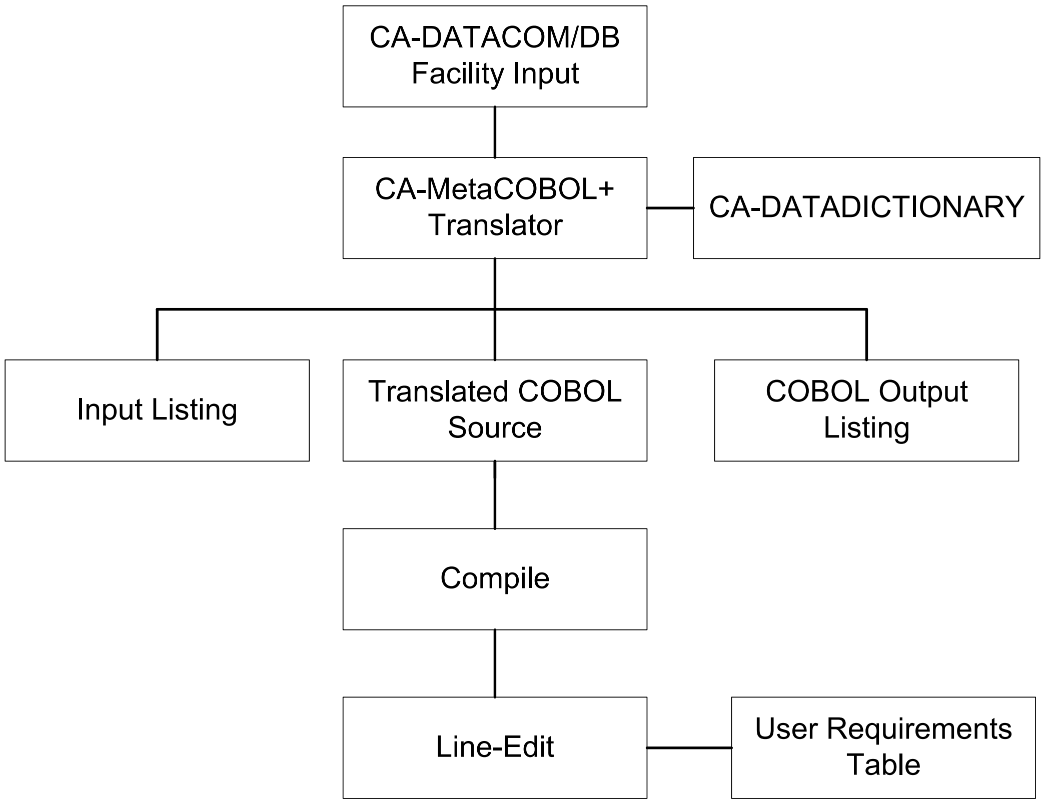
The CA-MetaCOBOL+ Translator translates CA-DATACOM/DB Facility statements into compiler-ready COBOL. It also verifies that the CA-DATADICTIONARY dataviews specified in the CA-DATACOM/DB Facility program exist in CA-DATADICTIONARY and that the program is authorized to use each dataview.

A high-level diagram of the translation process is shown in the following table. As shown, a COBOL program with embedded CA-DATACOM/DB Facility statements is input to the Translator and translated. The CA-DATADICTIONARY interface determines whether a CA-DATACOM/DB Facility program is authorized to use a dataview and whether the dataview specified in the program is valid.

The CA-MetaCOBOL+:Translator output consists of the following:

- a COBOL source file, which contains the compiler-ready code generated from CA-DATACOM/DB Facility statements.
- various output listings, including the Input listing and a listing of the generated COBOL.

The following illustration displays the CA-DATACOM/DB Facility translation process:



2.3 CA-MetaCOBOL+ Dialogs

CA-MetaCOBOL+ Dialogs are a package of interactive software specifically designed for the development and maintenance of COBOL application programs. They run under CA-ROSCOE, CA-VOLLIE, or TSO ISPF/PDF.

The dialogs allow you to converse with CA-MetaCOBOL+ through menus. In many cases, you only need to confirm default panel entries by pressing Enter. Your responses are edited and verified directly, thus minimizing errors that could otherwise occur during translation. Missing or invalid entries are flagged with an error message, and diagnostic assistance for error messages is available via PF keys. Testing and debugging are performed faster since you can review listings, error messages, and test results online.

The Dialog Facility provides services that present you with fill-in-the-blank and selection panels and with prompts for generating CA-DATACOM/DB Facility code. These services are briefly described below. For more information concerning these and other services provided by CA-MetaCOBOL+, refer to the CA-MetaCOBOL+ *User Guide*.

The CA-MetaCOBOL+ Dialog Facility provides the following services:

Program Shell Generation
The Program Shell Generation service, based on the type of program requested, generates a program shell containing COBOL statements such as COBOL division headers, PROGRAM-ID, and CONFIGURATION SECTION.

File Definition
The File Definition service provides prompts for defining the external (DD) and COBOL file names, and generating record names for your program.

Data Definition and Procedure Code Generation
The Data Definition and Procedure Code Generation services enhance your programming editor with keyword expansion. Keyword expansion provides you with the ability to generate entire CA-DATACOM/DB Facility, SP Facility, conventional COBOL, and command-level CICS syntax from a short form. For example, during a CA-MetaCOBOL+ session, you can expand the keyword .FOR into the following "model" of the FOR statement:

```
FOR EACH/FIRST/ANY <identifier-1/literal>
      dataview-name RECORDS
   <WHERE (condition)>
   <HOLD RECORDS>
   <COUNT IN identifier-2>
   <ORDER <UNIQUE> RECORDS ON
         <ASCENDING/DESCENDING> dataview-field-1
        <<ASCENDING/DESCENDING> dataview-field-2...>
       <process>
   <WHEN NONE
        process>
   <WHEN END
        process>
   <WHEN ERROR
        process>
ENDFOR
```

You can then delete clauses and add data names as appropriate for the application.

Program Formatting
The Program Formatting service provides you with the ability to format the Procedure Division of your program. Standard indentation and improved readability are among the benefits of this service.

Job Submission
The Job Submission service generates the JCL for translating, compiling, and link-editing a program. The generated jobstream can include CA-MetaCOBOL+ translation, command-level CICS precompile, COBOL compile, and link-edit jobsteps, depending on program attributes. The Job Submission service also allows you to view and edit the generated jobstream or to submit it directly.

2.4 Structured Programming with the SP Facility

A widely accepted tenet of data processing is that structured programming techniques should be used at all times. A properly structured program is a single-entrance, single-exit module consisting of statements taken from three categories: sequence, selection, and repetition. The Structured Programming (SP) Facility supports structured programming by requiring “structured” statements in COBOL programs.

SP Facility statements conform to structured programming techniques. The selection and repetition constructs are single-entrance, single-exit control structures. (The SP Facility places all other COBOL statements in the “sequence” category.) SP Facility statements prohibit the GO TO statement and prohibit the use of the PERFORM…THRU statement. Int his way the SP Facility encourages top=down design and development, and provides for Procedure Division code with easily nested syntax.

SP Facility examples of selection and repetition constructs follow. These statements do not represent complete syntax:

SP Facility Selection Constructs:

```
IF condition                SELECT [ALL]
    process                 WHEN condition
ELSE                             process
    process                 WHEN condition
ENDIF                            process
                            ENDSELECT
```

SP Facility Repetition Constructs:

```
LOOP                        SEARCH
    process                 WHEN END
UNTIL condition                 process
    process                 WHEN condition
ENDLOOP                         process
                            ENDSEARCH
```

The terminators ENDIF, ENDSELECT, ENDLOOP, and ENDSEARCH provide an exit for the construct, thus permitting unambiguous nesting.

SP Facility statements are translated into compiler-ready COBOL by the CA-MetaCOBOL+ Translator. CA-DATACOM/DB Facility and SP Facility statements may both be coded in the same program and translated at the same time. For more
Structured Programming Guide
information on the SP Facility, refer to the CA-MetaCOBOL+ Structured Programming Guide. .

2.5 Software Environment

The following products are required for using the CA-DATACOM/DB Facility as documented in this manual:

- CA-MetaCOBOL+ Release 1.0 or later
- CA-DATACOM/DB Release 7.4 or later
- CA-DATADICTIONARY Release 2.3H or later

3. Understanding CA-DATACOM/DB

CA-DATACOM/DB is Computer Associates relational database management system. CA-DATACOM/DB enables vast amounts of data to be processed both randomly and sequentially without compromising system performance. Full recovery/restart and transaction backout provide control and security of data. This section is provided for application developers who are unfamiliar with CA-DATACOM/DB.

3.1 System Overview

Data stored by CA-DATACOM/DB is accessed with standard CALL statements from an application program. Coding "native" calls in COBOL programs to access CA-DATACOM/DB is typically complex. Use of the CA-DATACOM/DB Facility's high-level COBOL-like statements simplifies coding requests for CA-DATACOM/DB services.

All data definitions for the information contained in CA-DATACOM/DB databases, including dataviews used by CA-DATACOM/DB Facility programs, are contained in CA-DATADICTIONARY. CA-DATADICTIONARY is maintained by the Database/CA-DATADICTIONARY Administrator at your site and is described in Chapter 4.

3.2 Structure of CA-DATACOM/DB

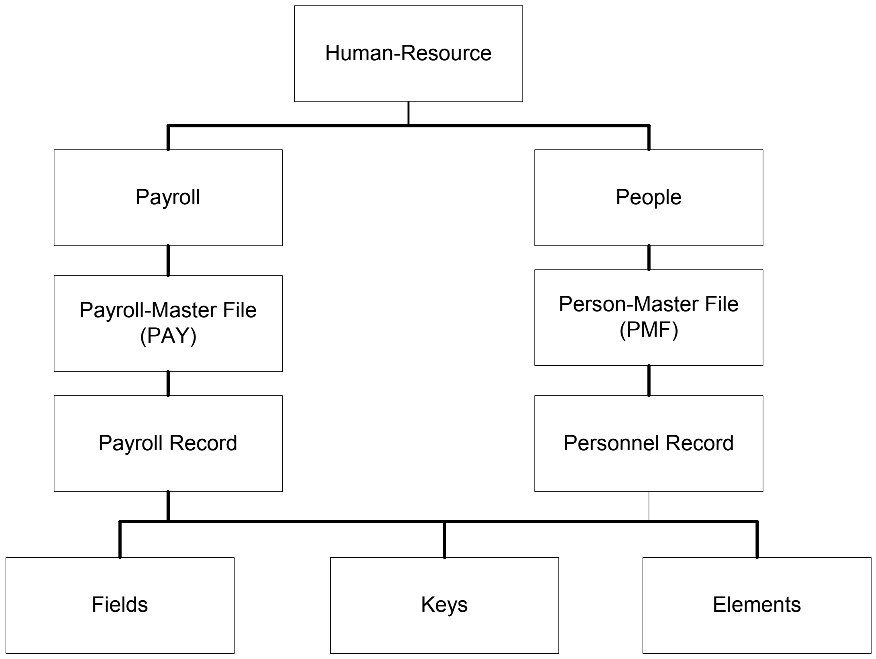
The structure of a CA-DATACOM/DB system can be summarized as follows:

CA-DATACOM/DB is an information base containing one or more *databases*, each of which may contain multiple physical *areas* and multiple logical files called *tables*. Physically, tables are composed of related *records*, the smallest physical unit of which is the *field*. Logically, tables are composed of *elements* of one or more fields, and *keys* of one or more fields.

The italicized keywords in this definition comprise the basic CA-DATACOM/DB :entities, which are defined as follows:

database	A CA-DATACOM/DB database is a collection of related data items with an index to those items. A CA-DATACOM/DB system can have multiple databases.
area	An area is the physical grouping of one or more logical tables. Up to 240 areas per database may exist.
table	A table is the logical grouping of records. Each record within a table has the same format.
record	A record is a collection of logically related fields. In CA-DATACOM/DB these fields are grouped into elements that are accessible in any combination. One or more keys can be assigned to each record.
element	An element is a named grouping of contiguous fields within a record. An element can span any part of a record, and a record can be described by several elements.
keys	A key is a field or multiple fields within a record and forms an index entry by which a record can be identified. A key can be up to 180 bytes long. A record can contain one master key, which uniquely identifies the record, and up to 98 secondary keys. A key is used to qualify or disqualify databases upon a key value.
field	The smallest unit of data that can be addressed by CA-DATACOM/DB Facility programs.

The following illustration shows the organization of a sample database called HUMAN-RESOURCE. This sample database is distributed with the CA-DATACOM/DB system and is referred to throughout this manual.



3.2.1 CA-DATACOM/DB Data Storage and Retrieval

Application programs request and send data to CA-DATACOM/DB at a logical rather than a physical level. That is, information on the physical location of data is not passed to CA-DATACOM/DB. Based on the physical device and record information provided by the Database Administrator (DBA) in CA-DATADICTIONARY, CA-DATACOM/DB determines the "path" it must take to access a database and retrieve the data requested by an application program.

CA-DATACOM/DB stores data in two-dimensional files called *tables*. A table is made up of rows of related fixed-format records. Shown below are some of the records contained in the Person-Master File (PMF) of the HUMAN-RESOURCE database.

According to relational database terminology, the columns in the following table are *domains* and each row, which represents an occurrence of data, is a *tuple*. The data contained in the table is *normalized*. That is, precisely one value per *domain* exists for each *tuple*; never a set of values.

```
ID         NAME            STREET          CITY     STATE/ZIP
00001      ALLEN ANTHONY   1322 TEMPAST DR BOSTON MA 02115
00002      LENZEN RUTH     642 RAY RD      SAN FRANCISCO CA
94104
00003      MEAD HENRY      902 PATRIA      BOSTON MA 02115
                                   .
                                   .
```

Keys and elements consist of fields within a record. An element is the basic unit of addressable data, while a key is the basic unit of access. Thus, CA-DATACOM/DB does not recognize the individual fields NAME, STREET, and CITY, etc., as shown in the table above; it associates the data with elements (EMDTA, ADEMP, and IDEMP) and keys, as shown in the table below.

For example, element EMDTA in the table below consists of the entire data record. Element ADEMP consists of the address information only, and element IDEMP consists of the ID and NAME only. The key STZIP is a concatenation of the STATE and ZIP fields.

Element	Key	Field
EMDTA	EMPNO	ID
	STZIP	NAME
		STREET
		CITY
		STATE
		ZIP
ADEMP	EMPNO	NAME
	STZIP	STREET
		CITY
		STATE
		ZIP
IDEMP	EMPNO	ID
		NAME

Note that the element and key names shown in the table above are their five-character CA-DATACOM/DB names. They have corresponding CA-DATADICTIONARY names, which may be up to 32 characters long for CA-DATACOM/DB 7.5 (15 characters for CA-DATACOM/DB 7.4), and may also have COBOL compiler names, which may be up to 30 characters long. For example, in the HUMAN-RESOURCE database, the key EMPNO has the CA-DATADICTIONARY name NUMBER and the compiler name EM-IDENTIFICATION-NUMBER.

It is the CA-DATADICTIONARY and compiler names for elements, keys, and fields that are referenced by CA-DATACOM/DB Facility programs. Consult with your DBA for the CA-DATADICTIONARY and compiler names of the necessary CA-DATACOM/DB database elements, keys, and fields to be accessed by your CA-DATACOM/DB Facility application.

3.2.2 Components of CA-DATACOM/DB

The following CA-DATACOM/DB:components are CA-DATACOM/DB designations for areas that contain CA-DATACOM/DB environmental and control information. They are maintained by the DBA.

Master List	The Master List defines the index and table buffers, operating information, and control information.
CXX	CXX is the or for an entire CA-DATACOM/DB information base. It contains a of all databases and the definitions of each database, including its areas, s, , and elements.
IXX	IXX is the and contains the key values and pointers to data. The IXX enables CA-DATACOM/DB to search for records quickly.
LXX	LXX is the and contains log records for transaction backup and program-generated records.
PXX	PXX is the and contains various CA-DATACOM/DB operational statistics for monitoring and tuning CA-DATACOM/DB. Statistics, dumps, and other information are stored in the statistics and diagnostics area in a unique format. A PXX report is generated with the CA-DATACOM/DB utility called DBUTLTY. Refer to the CA-DATACOM/DB *Database Administration Guide* or your DBA for a description on how to obtain a PXX report with DBUTLTY.
RXX	RXX is the and contains log records for long-term backup.
URT	The (URT) specifies the CA-DATACOM/DB resources that are necessary for a program to execute successfully. It defines which tables within a database may be accessed and updated. The URT also specifies various additional parameters, such as extra buffer allocations for sequential processing. You must consult with your DBA to determine the status of the URT before link-editing a CA-DATACOM/DB Facility program.

3.3 CA-DATACOM/DB Call-Level Programming

programming involves formatting database accesses and then issuing them with COBOL CALL statements. This means that you must determine which CA-DATACOM/DB command to invoke with the call and then initialize the parameters required by the command. You must also provide a workarea to receive the data requested from CA-DATACOM/DB. In CA-DATACOM/DB Facility application programs, however, you do not need to specify this information because it is generated automatically.

A is a five-byte instruction that specifies a request for CA-DATACOM/DB services. In a native call from a COBOL program, the CA-DATACOM/DB command is specified in the request area. An example of a command is GETIT, which retrieves records sequentially from any table in a database.

At run time, CA-DATACOM/DB interprets and verifies the call, locates the data, and returns the data record requested. A status code indicating the status of the request is returned to the application program.

The format of a native COBOL call to CA-DATACOM/DB and the definition of each parameter are shown below.

```
          CALL 'DBNTRY' USING user-information-block,
                              request-area,
                              element-list,
                              workarea,
                             [request-qual-area]
```

DBNTRY
: The CA-DATACOM/DB called by the program.

user information block
: The user information block identifies the requesting program to CA-DATACOM/DB. This area consists of the program name and is used to provide a unique identifier of the program for analyzing the PXX and LXX areas.

request area
: The request area specifies the CA-DATACOM/DB command and the name of the table to be searched. The and key value, which respectively represent the search argument and data value to be matched with the argument values in the table, are also located in the request area. The request area contains fields for the return code, error code, database identification, and record identification.

element list
: The element list specifies which data elements are to be retrieved, updated, or added.

workarea
: The call-level access:call parameters:workarea is where data requested by the application program is returned.

request qualification area
: The call-level access:call parameters:request qualification area is used to format requests for CA-DATACOM/DB set-at-a-time processing. This area contains the selection criteria and ordered-by parameters that determine which records are included in the set and the order in which they are returned to the program.

At a minimum, a request for CA-DATACOM/DB services must include a user information block and a request area. The presence of the remaining call parameters depends on the CA-DATACOM/DB command that is invoked. For example, the request qualification area is included only when CA-DATACOM/DB set-at-a-time processing is requested by the application program.

3.4 CA-DATACOM/DB Access Techniques

CA-DATACOM/DB provides two basic record-processing techniques: and . Both are supported by the CA-DATACOM/DB Facility.

3.4.1 Record-at-a-time Processing

Record-at-a-time processing is *key* processing. In key processing, a single record is selected per CA-DATACOM/DB request using previously defined key names and values. A record with the exact less-than-or-equal-to or greater-than-or-equal-to value specified may be selected. Access is available both to index entries and data records. Data records cannot be accessed directly. Instead, index entries must be accessed first. Then, optionally, data records are retrieved via index entries. The records always retrieved in the order of the selecting key.

Position is established in a database when a record is selected. Once position is established, an application program can "jump around" the database. That is, a program can use the position of the most recently selected record to retrieve a previously selected record, a record with duplicate key values, the next record regardless of key value, or the record with the next highest or lowest key value.

A record-at-a-time processing:index searching:specific search or generic search can be used to determine in which tables CA-DATACOM/DB will search for the specified key value. In a , the table name is specified in the request area, thus restricting the search to a single table. In a generic search, the table name is not included in the request area. Thus, the area of the search is larger in a generic search than in a specific search.

In the search strategies described above, records are selected randomly. Records also may be selected sequentially. Sequential record-at-a-time processing is a fast batch processing method that selects successive records in the sequence in which they are stored. While selecting records sequentially is inflexible in that it does not allow a database to be navigated arbitrarily, this strategy may be useful when circumstances demand maximum efficiency. (Note that the ACCESS= and SEQBUFS= parameters in the User Requirements Table may require modification for CA-DATACOM/DB Facility programs that use sequential processing. Consult your DBA.)

3.4.2 Set-at-a-time Processing

In , a *set* of records is defined based on the data values contained in a table. A set can be any size, from one record to an entire table. Once a set has been defined, individual records in the set can be accessed.

A set is built by selecting records that meet search conditions. A search condition is a compound Boolean expression comprised of one or more predicates and joined by the logical operators AND and OR. A predicate is composed of a subject, , and object. For example, the search condition

```
   EM-IDENTIFICATION-NUMBER > 100 AND EM-CITY-ADDRESS = 'HOUSTON'
```

provides the application program with the set of records for employees with ID numbers greater than 100 who live in Houston.

Set-at-a-time processing is useful when selection criteria specify a field that is not defined as a key. For example, in the selection criteria above, you could specify

```
     EM-CITY-ADDRESS = 'HOUSTON'
```

as the selection criteria, even though EM-CITY-ADDRESS is not a key. Another advantage of set-at-a-time processing is that records may be returned in either ascending or descending order.

CA-DATACOM/DB Facility programs request set-at-a-time processing with the FOR statement. The FOR statement is described in Section 6.6.2, "FOR Statement."

3.4.3 Exclusive Control

In record-at-a-time or set-at-a-time processing, when a program retrieves a record from CA-DATACOM/DB for updating, the program must retrieve the record with to prevent concurrent updates by more than one program. CA-DATACOM/DB allows other programs to read a record held with exclusive control, but not update it. Exclusive control is released by CA-DATACOM/DB when the update is complete or by using a FREE statement.

4. Understanding CA-DATADICTIONARY

This chapter is an overview of basic CA-DATADICTIONARY concepts. Its purpose is to assist the CA-DATACOM/DB Facility programmer in defining requests for CA-DATADICTIONARY information and modifications to the Database Administrator (DBA). The DBA is responsible for maintaining CA-DATADICTIONARY.

4.1 System Overview

CA-DATADICTIONARY is a repository of the descriptions of all components in a CA-DATACOM/DB information base. CA-DATADICTIONARY also describes the relationships between the components.

CA-DATADICTIONARY is a useful data processing tool that is able to:

- manage definitions and syntax
- enforce naming standards
- support data protection
- serve as a centralized security control mechanism
- create data relationships

CA-DATADICTIONARY plays an integral role in CA-DATACOM/DB Facility programming. The CA-MetaCOBOL+ Translator accesses CA-DATADICTIONARY to perform the following tasks:

- retrieve the COBOL definition for the field and keys required by the program.

- extract database ID, table name, element names, element security codes, and key names.

- ensure that programs use only those dataviews to which a relationship exists.

CA-DATACOM/DB Facility programs retrieve dataviews from CA-DATADICTIONARY. A dataview is a collection of CA-DATACOM/DB elements that can be requested in a single access. The DBA must define the dataviews required by an application to CA-DATADICTIONARY before you attempt to code a CA-DATACOM/DB Facility program. The DBA must also list which programs may use a dataview.

During the CA-MetaCOBOL+ translation, the CA-DATADICTIONARY interface verifies that the dataview specified in a CA-DATACOM/DB Facility DATAVIEW statement exists and that the CA-DATACOM/DB Facility program is authorized to use it. If a CA-DATACOM/DB Facility DATAVIEW statement specifies keys, the CA-DATADICTIONARY interface also verifies that the specified keys are valid for the dataview. (CA-DATACOM/DB Facility statements are described in Chapter 6.)

4.2 CA-DATADICTIONARY Concepts

Defining databases, tables, elements, dataviews, and programs is the responsibility of the DBA. Although you are not responsible for maintaining CA-DATADICTIONARY data definitions, as a CA-DATACOM/DB Facility programmer, you should understand how CA-DATADICTIONARY defines and categorizes data contained in CA-DATACOM/DB, and how CA-DATACOM/DB Facility programs make use of dataviews defined in CA-DATADICTIONARY.

CA-DATADICTIONARY describes the components, or *entities*, of an information base and the relationships between them. The entities are subdivided into s and s.

An entity-type is a general category of items in the database that share common characteristics and are grouped together for ease of reference. Any data entered into CA-DATADICTIONARY is associated with an entity-type. CA-DATADICTIONARY entity-types relevant to CA-DATACOM/DB Facility programs are:

```
AREA
DATABASE
DATAVIEW
ELEMENT
FIELD
FILE (TABLE)
KEY
PROGRAM
RECORD
```

An entity-occurrence is a specific instance within an entity-type. For example, TABLE is an entity-type. This means that the PMF table, which was described in Chapter 3, is an entity-occurrence of the entity-type TABLE.

The DBA records certain attributes for each entity-occurrence in CA-DATADICTIONARY. For example, for each entity-occurrence of the FILE entity-type, there is data on the particular file type, device type, and record size. Data pertaining to the status (TEST or PROD) and the version of the entity-occurrence also exists.

4.3 Version and Status

CA-DATADICTIONARY allows multiple occurrences of such entities as programs or dataviews, thus enabling entity-occurrences with the same name to coexist in CA-DATADICTIONARY. It accomplishes this by assigning a status and a version number to a single occurrence of a dataview. Each status has at least one version number.

The following are the status attributes for CA-DATADICTIONARY Version 2.3.

INCO	The INCO entity-occurrence is incomplete (i.e., it has not met all of the rules established by CA-DATADICTIONARY and/or the user for its existence). More than one version may exist for entity-occurrences with this status.
TEST	The TEST entity-occurrence represents an item that is in development and, as such, is subject to change. CA-DATADICTIONARY allows up to 999 versions of dataviews in TEST status.
QUAL	The QUAL entity-occurrence is in the process of being placed in production.
PROD	The PROD entity-occurrence represents an item as it is used in day-to-day production. Entity-occurrences with this status may not be modified. Only one version may exist for entity-occurrences with PROD status.
HIST	The HIST entity-occurrence represents an item that either has been replaced by a new version or removed from use; it cannot be changed. More than one version may exist for dataviews with this status. CA-DATADICTIONARY allows 998 versions of dataviews with HIST status.

4.4 Relationships

CA-DATADICTIONARY not only describes the characteristics of the entity-occurrences, but it also depicts the structure between entity-occurrences and entity-types (which entity-occurrences and entity-types are used with one another). This is accomplished using relationship-types. In this way the DBA can establish a relationship between an entity-occurrence and an entity-type or among entity-occurrences and entity-types.

The DBA specifies a relationship to CA-DATADICTIONARY with a relationship-type. A relationship-type has the following format:

```
subject.object,relationship-name
```

The subject and object can be any combination of entity-occurrences and entity-types. The relationship-name is a combination of a three-character designation for the subject and object, and one of the following relationship designators: ACCESS, AUTH, CALL, CONTAIN, COPY, INCLUDE, PRODUCE, RESIDE, and USE. The DBA may also create a site-specific relationship-type.

Note: The $INTERNAL relationship is an exception to this standard. The $INTERNAL relationship is a system relationship-type and exists at installation time.

The following relationships must be defined to CA-DATADICTIONARY for CA-DATACOM/DB Facility programs because they relate elements with a dataview and a program with the dataview. These relationships are specified to CA-DATADICTIONARY as follows:

```
DATAVIEW.ELEMENT,$INTERNAL
PROGRAM.DATAVIEW,PGM-DVW-USE
```

In other words, these relationships state that a dataview must be able to *access* elements and a program must be able to *use* a dataview.

In addition, the relationship DATAVIEW.KEY,$INTERNAL must exist between the dataview and key for a CA-DATACOM/DB Facility record-at-a-time dataview.

4.5 CA-DATADICTIONARY COBOL Compiler Names

Dataviews specified by CA-DATACOM/DB Facility programs and their associated COBOL data descriptions are retrieved from CA-DATADICTIONARY during translation. The COBOL data names are used by CA-DATACOM/DB Facility programs to reference individual fields in a dataview and are defined by the DBA. Therefore you must ask the DBA for the COBOL data descriptions associated with each dataview your program will access.

CA-DATADICTIONARY does not require COBOL compiler names for each dataview. If a compiler reference name is not specified for each field, the CA-DATADICTIONARY entity occurrence name may be used to reference the field in a dataview.

Warning:
Make sure that the DBA does not have COBOL or CA-DATACOM/DB Facility reserved words as names for compiler names, dataviews, elements, or keys in CA-DATADICTIONARY. These will cause translation and compiler errors. A list of the CA-DATACOM/DB Facility reserved words is provided in the CA-MetaCOBOL+ *Program Development Reference - CA-DATACOM/DB* manual.

5. Getting Started

In this chapter, the following preliminary considerations for developing CA-DATACOM/DB Facility programs are presented. For complete information, refer to the CA-MetaCOBOL+ *User Guide*.

- A list of the CA-DATACOM/DB and CA-DATADICTIONARY specifications required from the DBA is provided.
- The advantages of using record-at-a-time and set-at-a-time processing are described.
- An overview of the JCL to translate, compile, link-edit, and execute a CA-DATACOM/DB Facility program is described.

5.1 CA-DATACOM/DB and CA-DATADICTIONARY Specifications

Before you begin developing a program, ask the DBA for the status of the CA-DATACOM/DB and CA-DATADICTIONARY specifications outlined in this section.

5.1.1 CA-DATACOM/DB Requirements

CA-DATACOM/DB Version

CA-DATACOM/DB Release 7.4 and later support set-at-time processing. Prior releases do not support this facility.

User Requirements Table Name

The CA-DATACOM/DB User Requirements Table (URT) (File Table prior to Release 7.4) name must be specified in the INCLUDE link-edit statement for batch applications.

You should also determine the status of the following URT parameters: ACCESS=, DBID=, OPEN=, SEQBUFS=, SYNONYM=, TBLNAM=, and UPDATE=.

5.1.2 CA-DATADICTIONARY Requirements

For the following CA-DATADICTIONARY requirements that specify translate-time options, refer to Section 5.3.1, "Translate-time Options."

CA-DATADICTIONARY Version

The CA-DATADICTIONARY interface requires CA-DATADICTIONARY Release 2.3H or later.

Database ID of CA-DATADICTIONARY

More than one CA-DATADICTIONARY may exist in a CA-DATACOM/DB information base.

The DDID can be specified with the DDID translate-time option. The default is 000, indicating that the value should be retrieved from the system resource table DDSRTLM.

If more than one CA-DATADICTIONARY database is active at your installation, you must be sure that the CA-DATADICTIONARY database ID (DDID) specified in the CA-DATADICTIONARY :System Resource Table is the CA-DATADICTIONARY your application must access. Alternatively, you must explicitly specify the CA-DATADICTIONARY database ID on the DDID option.

Program Name

In a CA-DATACOM/DB Facility application, the PROGRAM-ID statement must specify a CA-DATADICTIONARY program entity-occurrence name. That is, the program-name given in the PROGRAM-ID header must be defined to CA-DATADICTIONARY as a program entity-occurrence.

Program Status

The program entity-occurrence status may be TEST or PROD. The program status is specified with the PSTAT translate-time option.

Program Version

If the program status in CA-DATADICTIONARY is TEST, the PVER translate-time option must specify the program's version number as specified in CA-DATADICTIONARY.

Dataview Name

The dataview name may be 1 to 30 characters for CA-DATACOM/DB Release 7.5 or 1 to 15 characters for CA-DATACOM/DB Release 7.4.

Dataview Status

For CA-DATACOM/DB 7.5, valid dataviews in TEST or PROD status may be executed. A database structure related to a TEST dataview does not need to be cataloged. However, it must be cataloged and have a valid structure if it is to be executed.

For CA-DATACOM/DB 7.4, the dataview entity-occurrence status must be PROD in order to execute a program using this dataview. A program containing TEST dataviews cannot be executed against the database.

Dataview Version

If the status is TEST, the CA-DATADICTIONARY version number of a dataview entity-occurrence must be specified in the TEST-VERSION clause of the CA-DATACOM/DB Facility DATAVIEW statement.

Dataview Key Names(s)

The key name may be 1 to 30 characters for CA-DATACOM/DB Release 7.5 or 1 to 15 characters for CA-DATACOM/DB Release 7.4.

COBOL Compiler Names

The COBOL data descriptions defined for the fields contained in the dataview.

Required Relationships

The DBA must relate your CA-DATACOM/DB Facility program and the dataview(s) it accesses with the relationship PGM-DVW-USE. This is not automatically done when you run your program.

Additional relationships must also be established between dataviews and elements. For record-at-a-time access, additional relationships must be established between dataviews and keys. For CA-DATACOM/DB 7.4, these relationships are named DVW-ELM-ACCESS and DVW-KEY-ACCESS.

CA-DATADICTIONARY System Resource Table

To interface the CA-DATACOM/DB Facility with CA-DATADICTIONARY, the NSTLVL parameter must be set to nine for the CA-DATADICTIONARY :System Resource Table option DDSYSTBO during installation. (Failure to do this results in translator error DLBA241F and CA-DATADICTIONARY :Service Facility return code MSS.)

5.2 Record-at-a-time Versus Set-at-a-time Processing

An advantage of record-at-a-time processing is that it can be faster because it allows the index to be searched for the location of a record without actually retrieving the record. One use of index searching is to quickly determine whether a record exists before it is processed. Another use of index searching is to locate data records that exist in different areas of the database but are often processed together. Therefore, record-at-a-time processing is more advantageous when an application is able to specify the data it requires with a key.

An advantage of set-at-a-time processing is that it is *key-independent*; that is, it can be used to access records for which no keys have been defined. This means that a CA-DATACOM/DB Facility application program using set-at-a-time processing can specify any valid single or multiple fields in a record as search criteria. However, you can specify key-fields as a selection criteria; if a selection-criteria field is not a key, the CA-DATACOM/DB Facility builds a temporary index to the set of records it is creating.

The English-like syntax of the FOR statement makes a program using set-at-a-time processing easy to code and maintain. For example, to retrieve the names of employees living in Texas, you could code a FOR statement similar to the following:

```
FOR EACH EMPLOYEE
       WHERE (EM-STATE-ADDRESS EQ 'TX')
                   DISPLAY EM-IDENTIFICATION-NUMBER
                   DISPLAY EM-IDENTIFICATION-NAME
        WHEN ERROR
                   DISPLAY DB-STATUS-CODE
ENDFOR
```

With set-at-a-time processing, application programs do not need to "look" at all potential records. CA-DATACOM/DB makes the *set* of records satisfying the search criteria available to the program. For example, an application program using set-at-a-time processing could retrieve the employees living in Texas with employee numbers greater than 1000 in a single access (by including "AND EM-IDENTIFICATION-NUMBER > 1000" in the WHERE clause of the above example). On the other hand, a program using record-at-a-time processing must access the database for each "Texas" record, and then it must check the matching record for an employee number greater than 1000.

5.3 Translating an Application Program

A CA-DATACOM/DB Facility program is translated into standard COBOL. The resulting compiler-ready source can then be compiled, link-edited, and executed. Certain translate-time options, macro sets, and translator-directing statements must be specified in your translation JCL. The following sections describe JCL basics for translating, compiling, link-editing, and executing your application program.

If you are developing your application using the CA-MetaCOBOL+ Dialog, the translation JCL is generated automatically; the dialog uses the Profile Table to retrieve the required values for translate-time options and macro sets.

5.3.1 Translate-time Options

Translate-time options control specific functions for the current execution of the CA-MetaCOBOL+ Translator. These options must be the first entry in the primary input stream. They can be specified:

- in one or more OPTION cards,
- For MVS only - in the PARM field of the JCL EXEC statement
- For MVS only - in both places.

MVS Examples

The translate-time options below are specified in the PARM field of the JCL EXEC statement.

```
//sample  JOB ...
//STEP   EXEC PGM=metacbl,
//        PARM='IXIT=ADRXIXIT,DDID=013,PSTAT=TEST,PVER=110'
         .
         .
         .
```

Below, the translate-time options are given in the OPTION parameter.

```
// sample   JOB ...
//STEP    EXEC PGM=metacbl
//CARDF  DD *
  OPTION IXIT=ADRXIXIT,DDID=013,PSTAT=TEST,PVER=110
         .
         .
         .
```

VSE Example

```
//EXEC METACOB,SIZE=AUTO
OPTION IXIT=ADRXIXIT,DDID=013,PSTAT=TEST,PVER=110
...
```

Refer to the CA-MetaCOBOL+ documentation for a description of the OPTION card and a complete list of the translate-time options that are relevant to the CA-DATACOM/DB Facility.

5.3.2 Translator-Directing Statements

Translator-directing statements control certain CA-MetaCOBOL+ Translator functions. They begin with an asterisk and a dollar sign (*$) in columns 7 and 8.

CA-DATACOM/DB Facility programs require the *$COPY (or *$LIBED) translator-directing statements. These statements are used to load macro sets and are specified immediately before the CA-DATACOM/DB Facility input source.

A list and description of CA-MetaCOBOL+ translator-directing statements is provided in the CA-MetaCOBOL+ *User Guide*.

5.3.3 CA-DATACOM/DB Facility Macro Sets

CA-DATACOM/DB Facility macro sets enable the CA-MetaCOBOL+ Translator to provide the high-level language facilities that allow your application to access CA-DATACOM/DB. These macro sets must be loaded before the input source program.

There are three macro sets that may be used when executing CA-DATACOM/DB Facility programs:

- SP Facility macros (SPP)
- Online Programming Language (OPLPS)
- Data Language Macros (DLM)

All macro sets must be loaded before the CA-DATACOM/DB Facility source. For CA-DATACOM/DB Facility programs containing SP Facility statements and/or OPL statements, the macro sets must be loaded in the following order:

1. SPP macro set
2. OPLPS
3. DLM macro set

For CA-DATACOM/DB Facility programs using conventional COBOL, only the DLM macro set is required.

Macro sets are loaded with the *$COPY translator-directing statements. The name of the macro set immediately follows the *$COPY statement.

5.4 Compiling a Program

The procedures for compiling a CA-DATACOM/DB Facility program are the same as for compiling a standard COBOL program. This is because the input to this step is standard COBOL. Any COBOL compiler diagnostics will appear in the output listing at the end of this step. For MVS systems, the input to the compile step is the Translator PUNCHF file; for VSE, it is the generated COBOL source file, which is created in the preceding translation step.

5.5 Link-Editing a Program

This section presents some considerations for link-editing CA-DATACOM/DB Facility programs.

5.5.1 Batch Programs

A CA-DATACOM/DB Facility batch program needs to be link-edited with a CA-DATACOM/DB User Requirements Table (URT).

There are two ways to interface a CA-DATACOM/DB Facility batch program with the operating system.

1. *Program is mainline, CA-DATACOM/DB URT is a subroutine of the program.* The program is responsible for opening and closing the URT with the and CLOSE statements. Link-edit the CA-DATACOM/DB batch programs and the URT with the ENTRY *progname*.

2. *CA-DATACOM/DB URT is mainline, program is a subroutine of CA-DATACOM/DB.* A URT defined with the parameter OPEN=DB is opened automatically before the program is called, and closed automatically when the COBOL program returns control to the CA-DATACOM/DB URT.

   A URT defined with the parameter OPEN=USER requires the program to open and close the URT. ENTER-DATACOM-DB, which generates the user program entry point DBMSCBL, must be the first CA-DATACOM/DB Facility statem~~ent for a~~, which generates the user ~~successful linkage~~-edit. Link-edit with the ENTRY BEGIN *progname*.

Loading the URT

The User Requirements Table is loaded in the linkage-editor step. There are two ways to load the URT in the link-edit step.

1. *Assemble the URT with the CA-DATACOM/DB interface.* With this technique, a single INCLUDE linkage-editor statement includes a URT that has been assembled with the CA-DATACOM/DB interface.

2. *Assemble the URT separately from the CA-DATACOM/DB interface and link it with your program.* Separate INCLUDE statements are necessary for loading the URT: one for the CA-DATACOM/DB interface and the other for the URT. This option allows a common CA-DATACOM/DB interface to be used with different URTs without re-assembling the CA-DATACOM/DB interface.

5.5.2 CICS Programs

CICS programs accessing CA-DATACOM/DB with CA-DATACOM/DB Facility statements function as subroutines to CICS. The program accesses CA-DATACOM/DB via the CA-DATACOM/DB CICS Service Facility. The CICS Service Facility opens and closes all URTs.

The module DBCSVPR links the program with the CICS Service Facility. An INCLUDE card is used to include the module in the link-edit step.

```
          INCLUDE DBCSVPR
```

5.6 Executing a Batch Program

After the program has been link-edited with the CA-DATACOM/DB interface, it may be executed.

If the URT is loaded at run-time, it must be stored in a load library available at execution. If a LOADPRM= is specified in the DBURINF macro assembly, the specified name must be included in the EXEC= parameter.

If single-user mode is in effect at your installation (and without dynamic allocation under MVS), define CA-DATACOM/DB data sets in the JCL. When the Multi-User Facility is used, the CA-DATACOM/DB files only need to be present in the multi-user JCL, not in the application program's JCL.

6. The CA-DATACOM/DB Facility Data Manipulation Language

This chapter describes CA-DATACOM/DB Facility statements and shows how to implement basic application tasks. For a complete description of statement syntax, see the CA-MetaCOBOL+ *Program Development Reference - CA-DATACOM/DB* manual.

6.1 Preliminary Considerations

Before you begin coding, plan the application using structured programming techniques. We recommend that you read the CA-MetaCOBOL+ *Structured Programming Guide*.

After designing the application, determine the CA-DATACOM/DB and CA-DATADICTIONARY specifications in effect at your installation. These specifications are listed in Chapter 5.

6.2 CA-DATACOM/DB Facility Statement Categories

The CA-DATACOM/DB Facility Data Manipulation Language (DML) is an *extension* to conventional COBOL. This means that conventional COBOL statements are valid in a CA-DATACOM/DB Facility program, and CA-DATACOM/DB Facility source input has the same structure as a conventional COBOL program.

CA-DATACOM/DB Facility statements are categorized by function in the following table. Appendix A displays the syntax of the statements listed below:

Statement Function	Division	Statement
Translation Environment	Environment	DATACOM SECTION
Database Administration	Data	DATAVIEW
Set-at-a-time Processing	Procedure	FOR
SKIP	Procedure	SKIP
Record-at-a-time Processing		
Sequential Retrieval	Procedure	LOCATE SEQUENTIAL
		READ/OBTAIN SEQUENTIAL
Random Retrieval	Procedure	READ/OBTAIN
		READ/OBTAIN NEXT
		READ/OBTAIN NEXTDUPLICATE
		READ/OBTAIN PREVIOUS
		READ/OBTAIN WITHIN RANGE
		READ/OBTAIN NEXT WITHIN RANGE
Database Maintenance	Procedure	WRITE/INSERT
		REWRITE/UPDATE
		DELETE
Index Search	Procedure	LOCATE
		LOCATE NEXT
		LOCATE NEXT DUPLICATE
		LOCATE NEXT KEY
		LOCATE PREVIOUS
		LOCATE...AT
		LOCATE WITHIN RANGE
		LOCATE NEXT WITHIN RANGE
Physical Sequential Proc.	Procedure	LOCATE PHYSICAL
		READ/OBTAIN PHYSICAL
Program Service		
Establish Entry Point	Procedure	ENTER-DATACOM-DB
Open/Close URT	Procedure	BACKOUT LOG
Logging	Procedure	CHECKPOINT LOG
		READ LOG
		WRITE LOG
Exclusive Control	Procedure	FREE LAST
		FREE ALL
		FREE SET
Diagnostic Assistance	Procedure	SET TEST OPTIONS
		ABEND

6.3 Identification Division

The program name specified with the PROGRAM-ID statement has special meaning to the CA-DATACOM/DB Facility. This program name must exist in CA-DATADICTIONARY as the program entity-occurrence name if CA-DATADICTIONARY dataviews are used in the program. In addition, this program entity-occurrence name must have TEST status in CA-DATADICTIONARY when you are testing and debugging your program. For more information on CA-DATADICTIONARY requirements, see Section 5.1.2, "CA-DATADICTIONARY Requirements."

In the following example, *TESTPGM* is the CA-DATADICTIONARY program entity-occurrence name. This statement enables the interface between the program and CA-DATADICTIONARY.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. TESTPGM.
```

6.4 Environment Division

The CA-DATACOM/DB Facility extends the Environment Division to include the DATACOM Section. The DATACOM Section specifies the program identification area, online monitor, and an output listing option. The program identification area is defined in the Data Division as a 01-level group item of 32 bytes containing the name of the program. The monitor is specified when the program is an online CICS program. The print option specifies whether you want generated code to appear in the translation source output listing.

The following example shows an Environment Division for a batch program.

```
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER.  IBM-370.
OBJECT-COMPUTER.  IBM-370.
DATACOM SECTION.
    ID-AREA IS ID-AREA-1
    PRINT NOGEN.
        .
        .
        .
DATA DIVISION.
01  ID-AREA-1      PIC X(32) VALUE 'TESTPGM'.
```

In this example, the ID-AREA statement indicates the data name of the programmer-defined program identification area, which is defined in the Working-Storage Section. The name specified by ID-AREA must be defined as a 32-byte group data item and must be given the value of the program name in the PROGRAM-ID statement above. PRINT NOGEN indicates that CA-DATACOM/DB Facility-generated code will not appear on the Input listing of the translation step.

The following example shows an Environment Division for an online CICS program.

```
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER.  IBM-370.
OBJECT-COMPUTER.  IBM-370.
DATACOM SECTION.
    MONITOR IS CICS
    ID-AREA IS ID-AREA-1
    PRINT NOGEN.
```

6.5 DATAVIEW Statement

The DATAVIEW statement identifies the CA-DATADICTIONARY dataview to be accessed by the program. The entity-occurrence name for the dataview, which is obtained from the (DBA), is specified in the DATAVIEW statement. At translate time, the dataview is retrieved from CA-DATADICTIONARY and used to generate a data area containing the formatted request for CA-DATACOM/DB services (request area and element list) and the data returned by CA-DATACOM/DB (workarea). You must code a DATAVIEW statement for each dataview accessed by the program.

DATAVIEW statements are coded in columns 8-72 of the Working-Storage or Linkage section. The DATAVIEW statement has three formats:

1. set-at-a-time
2. record-at-a-time
3. physical sequential

These formats correspond to the CA-DATACOM/DB access techniques supported by the CA-DATACOM/DB Facility and are shown in the following examples. The formats are further described in the CA-MetaCOBOL+ *Program Development Reference - CA-DATACOM/DB manual*.

The "dataview-name" referenced in the first two examples is the DATAVIEW statement name. DATAVIEW statement names may consist of 1 to 30 characters. However, the first 15 must be unique; no two DATAVIEW statement names may have the same characters in positions 1 through 15 within a program. If you plan to use a dataview twice within the same program for different types of processing, you can specify a prefix on the DATAVIEW statement.

The set-at-a-time DATAVIEW statement format follows:

```
        DATAVIEW  dataview-name

         [ {DD
         [ {DICTIONARY    } NAME IS dd-dataview-name]
         [ {DATADICTIONARY}

         [                  {LAST          }  ]
         [  TEST-VERSION IS {LATEST        }  ]
         [                  {version-number}  ]

         [  PREFIX IS 'data-name-prefix-literal']
                                                ]
         [ {DBID                  }              ]
         [ {DATA-BASE-ID          } [IS data-name]  ]   .
         [ {DATA-BASE-IDENTIFICATION}              ]
```

The record-at-a-time DATAVIEW statement format follows:

```
     DATAVIEW  dataview-name

     [ {DD            }                         ]
     [ {DICTIONARY    } NAME IS dd-dataview-name]
     [ {DATADICTIONARY}                         ]
     [ {              }                         ]
     [                  {              }  ]
     [                  {LAST          }  ]
     [  TEST-VERSION IS {LATEST        }  ]
     [                  {version-number   ]
     [                  {              }  ]
     [  PREFIX IS 'data-name-prefix-literal']
                            {        }
        ACCESS IS [GENERIC] {KEY IS  }
                            {KEYS ARE} dd-key-name [dd-key-
  name]...

     [ {DBID                  }                 ]
     [ {DATA-BASE-ID          } [IS data-name]  ]   .
     [ {DATA-BASE-IDENTIFICATION}               ]
```

The physical sequential DATAVIEW statement format follows:

```
    DATAVIEW  dataview-name

        [ {DD            }                     ]
        [ {DICTIONARY    } NAME IS dd-dataview-name]
        [ {DATADICTIONARY}                     ]

        [                  {LAST          }  ]
        [  TEST-VERSION IS {LATEST        }  ]
        [                  {version-number}  ]

        [  PREFIX IS 'data-name-prefix-literal']
           ACCESS IS PHYSICAL

        [ {DBID                   }                 ]
        [ {DATA-BASE-ID           } [IS data-name]  ]   .
        [ {DATA-BASE-IDENTIFICATION}                ]
```

The DATAVIEW statement formats share the following optional clauses: CA-DATADICTIONARY, TEST-VERSION, PREFIX, and DATA-BASE-IDENTIFICATION.

CA-DATADICTIONARY
The CA-DATADICTIONARY clause specifies the CA-DATADICTIONARY entity-occurrence name for the dataview accessed by the application program. When this clause is not specified, the CA-DATACOM/DB Facility will verify that the DATAVIEW statement name is defined as an entity-occurrence in CA-DATADICTIONARY. This allows you to choose a name for the dataview that has more meaning to your program than the dataview entity occurrence name. (For CA-DATACOM/DB 7.4, the dd-dataview-name must not exceed 15 characters; for CA-DATACOM/DB 7.5, 30 characters.) You can also use this clause to code two DATAVIEW statements that function logically as separate dataviews in your program, but actually reference the same CA-DATADICTIONARY dataview.

TEST-VERSION
The TEST-VERSION clause specifies the version number for a dataview in TEST status. If you specify a TEST version-number in a DATAVIEW statement, its CA-DATADICTIONARY dataview must be in TEST status and must have the same version-number. If you omit the TEST-VERSION clause, the CA-DATACOM/DB Facility assumes that the dataview it accesses in CA-DATADICTIONARY has PROD status. Remember that for CA-DATACOM/DB 7.4, TEST dataviews may only be translated and compiled; they may not be used to access a database at run time.

PREFIX-
With the PREFIX clause you can specify a prefix of up to five characters, including a hyphen, for data names in the workarea generated for the dataview. You can use this feature to give unique names to data items generated by the CA-DATACOM/DB Facility.

DATA-BASE-IDENTIFICATION

The DATA-BASE-IDENTIFICATION clause specifies the database that the program will access when a statement is issued using the dataview. This clause is required if SYNONYM=YES is specified in the User Requirements Table or if a generic search is requested with the dataview.

The database ID may be controlled at run time by coding the DBID IS data-name clause in the DATAVIEW statement. You must make sure that a valid value for the database ID is available as input to the program at run time. If DBID is coded in the DATAVIEW statement without IS *data-name*, the database ID is retrieved from CA-DATADICTIONARY.

6.5.1 Set-at-a-time DATAVIEW Statement

In the set-at-a-time format of the DATAVIEW statement, only the CA-DATADICTIONARY entity-occurrence name of the dataview is required. Therefore the dataview statement DATAVIEW DLEMPLOYEE is valid if DLEMPLOYEE is the CA-DATADICTIONARY entity-occurrence name for the desired dataview. In this case, DLEMPLOYEE is coded in the dataview-name position of the FOR statement.

You can also specify another name for the dataview to use in your program. In the following example, EMPLOYEE is the name that is coded in the FOR statement, and DLEMPLOYEE is the entity-occurrence name as before.

```
DATA DIVISION.
WORKING-STORAGE SECTION.
        .
        .
        .
DATAVIEW EMPLOYEE
    DATADICTIONARY NAME IS DLEMPLOYEE.
        .
        .
        .
PROCEDURE DIVISION.
        .
        .
        .
    FOR EACH EMPLOYEE...
```

6.5.2 Record-at-a-time DATAVIEW Statement

As with the set-at-a-time DATAVIEW statement, code the record-at-a-time DATAVIEW statement with the DATADICTIONARY clause to identify the dataview occurrence name and allow reference to a name of your choice, coded as the DATAVIEW statement name in your program. If you do not specify the DATADICTIONARY clause, you must use the CA-DATADICTIONARY entity-occurrence name for the dataview. (For CA-DATACOM/DB 7.4, the name must not exceed 15 characters; for CA-DATACOM/DB 7.5, 30 characters.)

The ACCESS clause is required in the record-at-a-time DATAVIEW statement. The ACCESS clause specifies whether a single table (specific search) or multiple tables (generic search) are accessed.

For a specific search, ACCESS KEY IS *dd-key-name* must be coded. (If the URT entry for the table to be searched specifies SYNONYM=YES, the DATAVIEW statement must also contain the DBID clause.)

For a generic search, code ACCESS IS GENERIC KEY IS *dd-key-name* in the DATAVIEW statement. The presence or absence of the DBID clause affects which tables are searched. If the DBID clause is not specified, all table names for the first DBID specified in the URT are searched. If the DBID clause is specified, only the table names listed in the URT for the specified DBID, regardless of their position in the URT, are searched.

The following are examples of specific and generic search DATAVIEW statements. Note that both DATAVIEW statements reference the same CA-DATADICTIONARY dataview (DLEMPLOYEE).

```
DATA DIVISION.
WORKING-STORAGE SECTION.
        .
        .
        .
DATAVIEW EMPLOYEE
    DATADICTIONARY NAME IS DLEMPLOYEE
    ACCESS KEY IS NUMBER.
        .
        .
        .
DATAVIEW EMPLOYEE-GEN
    DATADICTIONARY NAME IS DLEMPLOYEE
    ACCESS IS GENERIC KEY IS NUMBER
    DBID.
```

6.5.3 Physical Sequential Dataview

This format of the DATAVIEW statement designates a dataview for physical access. CA-DATACOM/DB Facility statements issued with the physical sequential dataview access data in its physical sequence through multiple block reads. This technique is used when an entire area must be read and the key sequence is not important.

A physical sequential DATAVIEW statement must contain the ACCESS IS PHYSICAL clause. The DATADICTIONARY NAME IS clause can also be coded if you wish to use a different name for the dataview in your program.
The following is an example of the PHYSICAL SEQUENTIAL format of the DATAVIEW statement.

```
DATA DIVISION.
WORKING-STORAGE SECTION.
       .
       .
       .
DATAVIEW EMPLOYEE
    DATADICTIONARY NAME IS DLEMPLOYEE
    ACCESS IS PHYSICAL.
```

6.6 Procedure Division for Set-at-a-time Processing

This section shows how to code CA-DATACOM/DB Facility statements to implement some of the functions listed in the table at the beginning of this chapter. For a complete definition of the syntax for the CA-DATACOM/DB Facility statements presented here, refer to the CA-MetaCOBOL+ *Program Development Reference - CA-DATACOM/DB* manual.

6.6.1 Preliminary Statements

If your program executes as a subroutine of CA-DATACOM/DB, you must code the ENTER-DATACOM-DB statement before any other CA-DATACOM/DB Facility statement. This statement provides the entry point (DBMSCBL) that CA-DATACOM/DB URT uses to transfer control to your program. (Refer to Section 5.5, "Link-Editing a Program," for other link-edit considerations.)

If your program executes as the main program and CA-DATACOM/DB as a subprogram, do not specify the ENTER-DATACOM-DB statement. However, OPEN=USER must be specified in the URT. When OPEN=USER is specified in the URT, the CA-DATACOM/DB Facility OPEN statement must be coded before the first statement accessing CA-DATACOM/DB. In addition, the CLOSE statement must be coded before the GOBACK statement.

6.6.2 FOR Statement

The FOR statement allows you to select records without specifying keys. The set of records meeting the selection criteria may be returned to the program for processing.

The syntax of the SP Facility FOR statement is shown in the following figure. The conventional COBOL format of the FOR statement is given in the CA-MetaCOBOL+ *Program Development Reference - CA-DATACOM/DB* manual. The only significant difference in syntax between the two formats is that the SP Facility FOR statement terminates with ENDFOR, whereas the conventional COBOL FOR statement terminates with a period. In addition, nested FOR statements may be coded in SP Facility programs.

The WHERE clause conditionally specifies the selection criteria. The WHERE clause is a simple or complex COBOL condition. In a simple condition, a subject and object are joined by one of the following relations.

Relation	**Short Forms**	
IS EQUAL TO	IS = TO	EQ
IS LESS THAN	IS < THAN	LT
IS GREATER THAN	IS > THAN	GT
IS NOT EQUAL TO	IS NOT = TO	NE
IS NOT LESS THAN	IS NOT < THAN	NL
IS GREATER THAN OR EQUAL TO	> OR =	GE, >=
IS EQUAL TO OR GREATER THAN	= OR >	=>
IS NOT GREATER THAN	IS NOT > THAN	NG
IS LESS THAN OR EQUAL TO	< OR =	LE, <=
IS EQUAL TO OR LESS THAN	= OR <	=<

A complex condition joins simple conditions by AND and/or OR. Also, NOT can be used to negate a simple or complex condition.

FOR statement syntax follows:

```
FOR {EACH  {                        }  }
    {FIRST {record-count-identifier}  }
    {ANY   {record-count-literal   }  }

     dataview-name {RECORD }
                   {RECORDS}

    [WHERE (condition)]
    [     {        } ]
    [HOLD {RECORD  } ]
    [     {RECORDS} ]

    [COUNT IN retrieval-count-identifier]

    [ORDER [UNIQUE] RECORDS ON
    [ { [          ]                       }      ]
    [ { [ASCENDING ]                       }      ]
    [ { [DESCENDING] dataview-identifier } ... ]

         [process-1] ...

    [WHEN END          ]
    [    process-2 ...]

    [WHEN ERROR        ]
    [    process-3 ...]

    [WHEN NONE         ]
    [    process-4 ...]
ENDFOR
```

The EACH/FIRST/ANY keywords allow you to restrict the records that CA-DATACOM/DB returns with the set. EACH specifies that every record matching the selection criteria in the WHERE clause is returned. FIRST returns the first n matching records, and ANY returns any *n* matching records.

The ORDER clause specifies an ordering of ascending/descending on any data-item. There is no difference in the records returned with either FOR FIRST or FOR ANY if the ORDER clause is not specified. When the ORDER clause is coded in a FOR FIRST statement, matching records are first identified and sorted, and then the first *n* records from the sorted set are returned to the program; that is, the sort is performed *before* the set is selected. When the ORDER clause is coded in a FOR ANY statement, the first *n* matching records are identified, and then the resulting set is sorted and returned to the program; that is, the sort is performed *after* the set has been selected.

Compare the records retrieved from the PMF table in the HUMAN RESOURCE database with the FOR EACH, FOR FIRST, and FOR ANY statements shown in the following example. The FOR EACH statement shows the entire set of records matching the selection criteria EM-CITY-ADDRESS EQ 'HOUSTON'. Because the sort specified by the ORDER clause occurs before records are selected in the FOR FIRST statement and after records are selected in the FOR ANY statement, different records are returned to the program.

6.6.3 Status Checking

Example 1

```
      FOR EACH EMPLOYEE RECORD
          WHERE (EM-CITY-ADDRESS EQ 'HOUSTON')
                DISPLAY EMPLOYEE-WORKAREA
          WHEN ERROR
                DISPLAY DB-STATUS-CODE
      ENDFOR
00006     SNOW TERRI             1810 VAN CR           HOUSTON    TX 77506
00017     DAVENPORT JACQUELINE   1913 GETTYSBURG PL    HOUSTON    TX 77506
00028     LOVELL RAYMOND         700 CENTER ST         HOUSTON    TX 77506
00038     BREEDEN JERRY          11023 TASCOSA RD      HOUSTON    TX 77506
00046     LANE GRACE             3880 DURANGO RD       HOUSTON    TX 77506
00053     GRAHAM PATRICK         3221 HUDNALL ST       HOUSTON    TX 77506
00061     GANTZ CAROLINE         1814 WALNOT ST        HOUSTON    TX 77506
00069     ROBERTS MELVIN         2222 GRAYCLIFF AVE    HOUSTON    TX 77506
00077     BREEN GEORGE           5919 BIRCHBROOK LA    HOUSTON    TX 77506
00081     MORGAN JOSEPH          1306 GAUCHO DR        HOUSTON    TX 77506
00089     HOUSLEY EARL           3312 REGENT DR        HOUSTON    TX 77506
00097     BRET JOHN              6647 AINTREE DR       HOUSTON    TX 77506
00105     CHURCH PHILLIP         1900 LANETTE CR       HOUSTON    TX 77506
00115     ABEL PHILIP            3547 BRANDON PL       HOUSTON    TX 77506
00123     NEELY ROY              3100 CONCORD RD       HOUSTON    TX 77506
00130     DIETER RODNEY          2610 CRESTBROOK       HOUSTON    TX 77506
00141     KERBY RICHARD          1153 TRACY LA         HOUSTON    TX 77506
00147     MAYFIELD GERALD        9737 AMBERTON LA      HOUSTON    TX 77506
00150     GEIGER ROBERT          7815 VILLA CLIFF      HOUSTON    TX 77506
00157     TONROY RAY             7307 FILLMORE         HOUSTON    TX 77506
00168     STANYER VERNON         306 PARKVALE          HOUSTON    TX 77506
00176     LANGSTON MELVIN        8825 ELDON CT         HOUSTON    TX 77506
00191     KLEIN CAROL            5904 ARAPAHO          HOUSTON    TX 77506
00200     HOWARD LANE            3120 HARMON           HOUSTON    TX 77506
```

Example 2

```
     FOR FIRST 10 EMPLOYEE RECORDS
         WHERE (EM-CITY-ADDRESS EQ 'HOUSTON')
         ORDER RECORDS ON ASCENDING EM-IDENTIFICATION-NAME
               DISPLAY EMPLOYEE-WORKAREA
         WHEN ERROR
               DISPLAY DB-STATUS-CODE
     ENDFOR
```

```
00115    ABEL PHILIP          3547 BRANDON PL     HOUSTON  TX 77506
00038    BREEDEN JERRY        11023 TASCOSA RD    HOUSTON  TX 77506
00077    BREEN GEORGE         5919 BIRCHBROOK LA  HOUSTON  TX 77506
00097    BRET JOHN            6647 AINTREE DR     HOUSTON  TX 77506
00105    CHURCH PHILLIP       1900 LANETTE CR     HOUSTON  TX 77506
00017    DAVENPORT JACQUELINE 1913 GETTYSBURG PL  HOUSTON  TX 77506
00130    DIETER RODNEY        2610 CRESTBROOK     HOUSTON  TX 77506
00061    GANTZ CAROLINE       1814 WALNOT ST      HOUSTON  TX 77506
00150    GEIGER ROBERT        7815 VILLA CLIFF    HOUSTON  TX 77506
00053    GRAHAM PATRICK       3221 HUDNALL ST     HOUSTON  TX 77506
```

Example 3

```
     FOR ANY 10 EMPLOYEE RECORDS
         WHERE (EM-CITY-ADDRESS EQ 'HOUSTON')
         ORDER RECORDS ON ASCENDING EM-IDENTIFICATION-NAME
               DISPLAY EMPLOYEE-WORKAREA
         WHEN ERROR
               DISPLAY DB-STATUS-CODE
     ENDFOR
```

```
00038    BREEDEN JERRY        11023 TASCOSA RD    HOUSTON  TX 77506
00077    BREEN GEORGE         5919 BIRCHBROOK LA  HOUSTON  TX 77506
00017    DAVENPORT JACQUELINE 1913 GETTYSBURG PL  HOUSTON  TX 77506
00061    GANTZ CAROLINE       1814 WALNOT ST      HOUSTON  TX 77506
00053    GRAHAM PATRICK       3221 HUDNALL ST     HOUSTON  TX 77506
00046    LANE GRACE           3880 DURANGO RD     HOUSTON  TX 77506
00028    LOVELL RAYMOND       700 CENTER ST       HOUSTON  TX 77506
00081    MORGAN JOSEPH        1306 GAUCHO DR      HOUSTON  TX 77506
00069    ROBERTS MELVIN       2222 GRAYCLIFF AVE  HOUSTON  TX 77506
00006    SNOW TERRI           1810 VAN CR         HOUSTON  TX 77506
```

The FOR statements shown in the examples above each contain a WHEN ERROR clause. The WHEN ERROR clause is executed when a CA-DATACOM/DB error code is returned to the program. The statements following the WHEN ERROR clause are then processed. In the examples above, a field named DB-STATUS-CODE is DISPLAYed if the WHEN ERROR clauses are executed. DB-STATUS-CODE is a generated group data item that contains the CA-DATACOM/DB and CBS error codes for the most recent database call. CA-DATACOM/DB error codes are identified in the CA-MetaCOBOL+ *Program Development Reference - CA-DATACOM/DB* manual, which also contains a detailed description of DB-STATUS-CODE and FOR statement error processing.

6.6.4 FOR Statement Nesting

In COBOL programs, FOR statements can be nested with PERFORM...THRU. In SP Facility programs, FOR statements can be nested with the FOR-ENDFOR construct. This type of nesting is structured, unambiguous, and easy to follow.

If you code nested FOR statements and your program abnormally terminates, check the DB-DL-FOR-STATUS field in DB-STATUS-CODE with the WHEN ERROR clause. The field contains NE when a nesting error occurs (i.e., concurrent access of the same dataview has been attempted) or FA when a FOR FIRST/ANY error occurs (i.e., the value of the record-count-identifier is negative).

You can use the FOR statement to compare one record from table A to several records from table B. According to relational database terminology, this technique is known as a *one-to-many table equi-join*. This technique is accomplished by selecting the initial record from table A with the outer FOR statement and then searching table B for related records with the inner FOR statement.

For example, suppose the HUMAN RESOURCE database had an additional table, called DEP, containing records of dependents for each employee in the PMF (employee) table. We could call a dataview for the DEP table DEPENDENT. Further, suppose that a COBOL name for a key field in the dataview is DP-IDENTIFICATION-NUMBER, which has the same format as the five-character employee ID shown in the above example, and that DP-IDENTIFICATION-NUMBER may have duplicate values in the table.

Because the ID field is common to both tables, it can be used to join (match) an employee record with its related dependent record in the PMF table. Thus, the ID is a *join field value* for both tables (i.e., it is a field that can be used to relate the PMF table with the DEP table).

In the following example, the dependents for each employee living in Houston are listed. The COBOL data names beginning with EM- or DP- in the example are the compiler names that you would typically acquire from the DBA.

```
DATA DIVISION.
       .
       .
       .
DATAVIEW EMPLOYEE.
DATAVIEW DEPENDENT.
01  WS-REC-COUNTS      COMP-3.
    05  DEP-COUNT      PIC S9(05)    VALUE  +0.
       .
       .
       .
PROCEDURE DIVISION.
    ENTER-DATACOM-DB
    OPEN
       .
       .
       .
    FOR EACH EMPLOYEE RECORD
        WHERE (EM-CITY-ADDRESS EQ 'HOUSTON')
        FOR EACH DEPENDENT RECORD
            WHERE (DP-IDENTIFICATION-NUMBER EQ
EM-IDENTIFICATION-NUMBER)
            COUNT IN DEP-COUNT
                 DISPLAY DEPENDENT-WORKAREA
        WHEN END
             DISPLAY DEP-COUNT
                     ' DEPENDENTS FOR EMPLOYEE '
                       EM-IDENTIFICATION-NUMBER
        WHEN ERROR
             DISPLAY DB-STATUS-CODE
        WHEN NONE
             DISPLAY ' NO DEPENDENTS FOR EMPLOYEE '
                       EM-IDENTIFICATION-NUMBER
        ENDFOR
    WHEN ERROR
         DISPLAY DB-STATUS-CODE
    ENDFOR
```

In the above example, when the WHEN END clause in the inner FOR statement is executed, the total number of dependents for each employee selected is DISPLAYed. This value is accumulated using the COUNT IN clause. In addition, when the WHEN NONE clause in the inner FOR statement is executed, the employee ID for an employee who has no dependents is DISPLAYed.

6.6.5 Obtaining and Releasing Exclusive Control

The HOLD RECORDS clause of the FOR statement obtains exclusive control over the set of records returned to the application program. Code the HOLD RECORDS clause only if records are to be updated or deleted.

When FOR statement processing terminates, the CA-DATACOM/DB resources for building the set are released. These resources include set definition, temporary indices, and the release of the last set retrieved under exclusive control.

If you transfer control out of the FOR statement, specify the FREE SET statement before exiting the FOR statement. The FREE SET statement releases the resources acquired for the specified dataview. In the previous nested FOR example, we could add the following code just before the inner FOR statement to terminate processing immediately if an employee with an ID number of 00000 is read:

```
IF EM-IDENTIFICATION-NUMBER = '00000'
  FREE SET EMPLOYEE
  EXIT
ENDIF
```

6.7 Procedure Division for Record-at-a-time Processing

There are two ways to retrieve records with record-at-a-time processing: sequential retrieval and random retrieval. Generally, to retrieve a record with either method:

1. Position is established in the index with a prerequisite statement. That is, a key value of the desired record is used to search the index for the pointer to the record.

2. Then the desired record is retrieved for processing.

The following sections describe the statements for implementing index searching, and sequential and random retrieval. In addition, the statements for updating, adding, and processing records with exclusive control are described.

Note that record-at-a-time statements may be used with the DATAVIEW statement shown in the example in Section 6.5, "DATAVIEW Statement." In addition, statements using the generic search technique to search (locate only) multiple tables must use dataviews that have ACCESS IS GENERIC specified in the DATAVIEW statement.

Refer to Chapter 8 for syntax descriptions of CA-DATACOM/DB Facility statements and Section 6.6.1, "Preliminary Statements," for the preliminary statements that may be required prior to execution of record-at-a-time statements.

6.7.1 Index Searching

Index searching locates an occurrence of a record matching a specified key. In this way, index searching "marks" a record for retrieval. A specific or generic index search may be implemented. Refer to Section 6.5.2 for a description of how to specify the DATAVIEW statement for a generic/specific search.

The LOCATE statement searches an index for an equal or next higher value of the specified key. LOCATE only verifies that a record exists; it does not return data to the workarea. The LOCATE statement has the following formats:

- LOCATE dataview-name WHERE condition searches an index for an equal or next higher key value. (A prerequisite statement for establishing position in the index does not need to be issued before LOCATE WHERE.)
- LOCATE NEXT dataview-name searches the index for the record in the next sequential entry. LOCATE NEXT DUPLICATE dataview-name searches the index for the next entry with the same key value. LOCATE NEXT KEY dataview-name searches the index for the next sequential key value in the index, skipping over duplicates.
- LOCATE PREVIOUS dataview-name establishes position on a previous key value.
- LOCATE dataview-name-1 AT dataview-name-2 establishes position for dataview-1 at the current position of dataview-2.
- LOCATE SEQUENTIAL dataview-name establishes the beginning point for reading an index based on an equal or higher match of a specific key value. (This form of the LOCATE statement is used for sequential retrieval only.)
- LOCATE dataview-name WITHIN RANGE WHERE keyname IS m THRU n locates a record key value nearest or matching m (m and n can be identifiers or literals). This statement is a prerequisite of LOCATE NEXT dataview-name WITHIN RANGE, which locates the next record key value within the range matching or greater than the current key value.

The following example illustrates how to use LOCATE statements to implement index searching.

```
                .
                .
                .
     LOCATE EMPLOYEE
         WHERE EM-IDENTIFICATION-NUMBER = INPUT-EMP-ID
     IF EMPLOYEE-RA-STATUS-CODE EQUAL SPACES
         DISPLAY EMPLOYEE
         LOCATE PAYROLL AT EMPLOYEE
         IF PAYROLL-RA-STATUS-CODE EQUAL SPACES
             DISPLAY PAYROLL
         ENDIF
     ENDIF
                .
                .
                .
```

Using the example, assume the value for INPUT-EMP-ID is 00005. The LOCATE WHERE statement searches the EMPLOYEE table for an employee ID that is equal to 00005. When found, the record is displayed.

```
00001  ALLEN ANTHONY    1322 TEMPAST DR    BOSTON         MA 02115
00002  LENZEN RUTH      642 RAY RD         SAN FRANCISCO  CA 94104
00003  MEAD HENRY       902 PATRIA         BOSTON         MA 02115
00004  PARKER WILLIAM   423 ARD RD         CHICAGO        IL 60632
00005  RIGSBY JOHN      9305 MOSS          NEW YORK       NY 10004
00006  SNOW TERRI       1810 VAN CR        HOUSTON        TX 77506
00007  TURNER ARNOLD    418 RUSTIC         BOSTON         MA 02115
00008  WEEKS LEN        4005 ROSA LN       SAN FRANCISCO  CA 94104
00009  LUTHER GARY      13410 ONYX         DALLAS         TX 75243
00010  BRATTON GEORGE   3600 GASTON AVE    ATLANTA        GA 30342
```

The LOCATE AT statement then establishes position in the PAYROLL table at the current position of the EMPLOYEE table; the record is then displayed.

```
00001   A   S      00300000      07800000      02500000     02300000
00002   A   S      00040500      01053000      00100000     00090000
00003   A   S      00040500      01053000      00110000     00090000
00004   A   S      00040000      01040000      00105000     00090000
00005   A   H      00000855      00889200      00150000     00070000
00006   A   S      00041000      01066000      00115000     00095000
00007   I   S      00000000      00000000      00000000     00000000
00008   A   S      00040500      01053000      00105000     00090000
00009   A   H      00000855      00889200      00150000     00070000
00010   A   S      00040000      01040000      00100000     00085000
```

Note that LOCATE NEXT, LOCATE NEXT DUPLICATE, LOCATE AT, and LOCATE NEXT KEY must follow a statement that has established a position in the database. For example, if a LOCATE...WHERE successfully finds a record that meets the search condition, a LOCATE NEXT may be executed. To obtain a list of prerequisite statements, refer to the CA-MetaCOBOL+ *Program Development Reference - CA-DATACOM/DB* manual.

To implement a specific or generic index search, issue a LOCATE statement specifying a dataview which has been defined for a generic or specific search with the ACCESS IS clause on the DATAVIEW statement.

6.7.2 Random Retrieval

As previously shown, the LOCATE statement provides a fast, easy method for "browsing" through the database. The READ statement randomly retrieves, and optionally holds, records for processing. The following statements are used to randomly retrieve records. Exclusive control of a record is obtained with the AND HOLD clause. The verbs READ and OBTAIN are equivalent.

- READ/OBTAIN [AND HOLD] *dataview-name* [WHERE *condition*]

  For CA-DATACOM/DB Release 7.5, this statement retrieves records with key values that are less than or equal to, equal to, or greater than or equal to a key value or record identifier, or if not found, a record *matching* or *nearest* the requested key.

  For CA-DATACOM/DB Release 7.4, this statement retrieves records with key values that are equal to a key value or record identifier.

- READ/OBTAIN [AND HOLD] NEXT *dataview-name* retrieves the next data record in sequence within the table. READ/OBTAIN NEXT DUPLICATE [AND HOLD] *dataview-name* reads the next data record containing the same key value. A successful READ or LOCATE are prerequisites of these statements.

- READ/OBTAIN PREVIOUS [AND HOLD] *dataview-name* retrieves the record preceding the current key value. A successful READ or LOCATE is a prerequisite of this statement.

- READ/OBTAIN [AND HOLD] *dataview-name* WITHIN RANGE WHERE *condition* retrieves the next record with a key value within the range matching or greater than the specified key value. This statement is a prerequisite to READ/OBTAIN [AND HOLD] NEXT *dataview-name* WITHIN RANGE, which retrieves the next record with a key value within the range matching or greater than the current key value.

The following example illustrates how to use the READ statement to randomly retrieve a record; if found, the record is deleted.

```
                  .
                  .
                  .
   READ EMPLOYEE AND HOLD
       WHERE EM-IDENTIFICATION-NUMBER = INPUT-EMP-ID
   IF EMPLOYEE-RA-STATUS-CODE EQUAL SPACES
       DELETE EMPLOYEE
   ENDIF
                  .
                  .
                  .
```

Using the example, assume the value for INPUT-EMP-ID is 00005. If found, the READ statement retrieves the record from the EMPLOYEE table where the ID is equal to 00005; the record is then deleted.

```
00001  ALLEN ANTHONY    1322 TEMPAST DR    BOSTON         MA 02115
00002  LENZEN RUTH      642 RAY RD         SAN FRANCISCO  CA 94104
00003  MEAD HENRY       902 PATRIA         BOSTON         MA 02115
00004  PARKER WILLIAM   423 ARD RD         CHICAGO        IL 60632
00005  RIGSBY JOHN      9305 MOSS          NEW YORK       NY 10004
00006  SNOW TERRI       1810 VAN CR        HOUSTON        TX 77506
00007  TURNER ARNOLD    418 RUSTIC         BOSTON         MA 02115
00008  WEEKS LEN        4005 ROSA LN       SAN FRANCISCO  CA 94104
00009  LUTHER GARY      13410 ONYX         DALLAS         TX 75243
00010  BRATTON GEORGE   3600 GASTON AVE    ATLANTA        GA 30342
```

6.7.3 Sequential Retrieval

Sequential retrieval is a fast record-at-a-time method for retrieving records from the database. These records are valid for batch processing only. Sequential retrieval is implemented with the LOCATE SEQUENTIAL and READ/OBTAIN sequential statements, which are defined as follows.

- LOCATE SEQUENTIAL *dataview-name* establishes the starting point for reading the index based on an equal or higher match on the specified key value. The key value is specified with the WHERE clause. This statement must precede the READ/OBTAIN sequential statement.

- READ/OBTAIN SEQUENTIAL *dataview-name* retrieves records sequentially, beginning with the record position established with the LOCATE SEQUENTIAL statement.

The following example illustrates how to use these statements to implement sequential retrieval. The LOCATE SEQUENTIAL statement searches for an employee record that is greater than or equal to LOW-VALUES (i.e., get first record in the PMF table). The record marked by this index entry is retrieved with the OBTAIN SEQUENTIAL statement.

```
DATAVIEW EMPLOYEE
   ACCESS KEY IS NUMBER.
       .
       .
       .
LOCATE SEQUENTIAL EMPLOYEE
   WHERE NUMBER GREATER THAN OR EQUAL LOW-VALUES.
OBTAIN SEQUENTIAL EMPLOYEE.
```

6.7.4 Database Maintenance

The following statements are used to add, update, or delete records from the database.

- WRITE/INSERT *dataview-name* adds records to the database. Since the key of each record added to the database is defined in the CA-DATACOM/DB control file (CXX), an entry for the new record is automatically added to the index.
- DELETE *dataview-name* removes records from the database and the index entry associated with the record.

  Note:
  Exclusive control must be obtained before a record can be deleted. (See Section 6.7.2, "Random Retrieval.")
- REWRITE/UPDATE *dataview-name* updates records.

Refer to Section 6.7.7, "Record-at-a-time Example," for an example of maintenance statements.

6.7.5 Obtaining and Releasing Exclusive Control

Exclusive control allows you to reserve data records for the exclusive use of your application. That is, when your application retrieves a record from CA-DATACOM/DB under exclusive control, no other task may have access to the record until exclusive control has been released. The AND HOLD clause of the READ/OBTAIN statement places records under exclusive control.

An application can release exclusive control with the FREE statement. The FREE statement releases update protection and has the following formats.

- FREE LAST *dataview-name* releases control of the *last* record held under exclusive control.

- FREE ALL *dataview-name* releases control of *all* records acquired with update protection by the AND HOLD clause of a READ/OBTAIN statement.

For example, the FREE SET statement below releases the resources acquired for the EMPLOYEE dataview if an employee with an ID number of 00000 is read:

```
FOR EACH EMPLOYEE RECORD
    WHERE (EM-CITY-ADDRESS EQ 'HOUSTON')
    IF EM-IDENTIFICATION-NUMBER = '00000'
        FREE SET EMPLOYEE
        EXIT
    ENDIF
         .
         .
         .
ENDFOR
```

6.7.6 Status Checking

You can use the field *dataview-name*-RA-STATUS-CODE field of the *dataview-name*-REQUEST-AREA generated for each dataview to determine the status of a CA-DATACOM/DB request. If a record is successfully located or returned to an application program, this field contains spaces. If no record is found, it contains '14'. (A list of CA-DATACOM/DB error codes are provided in Appendix B of the CA-MetaCOBOL+ *Program Development Reference - DATACOM/DB* manual.)

You can determine the result of each database access by checking the value of *dataview-name*-RA-STATUS-CODE. For example, your program logic could branch to an error processing routine when this field does not contain spaces.

The *dataview-name*-REQUEST-AREA group data item contains other fields that are useful for status checking. One of these is the *dataview-name*-RA-FILE-ID, which contains the three-byte name of the table accessed after a request for record retrieval has been issued. The table name can be checked after a record has been located with a generic search.

An example of CA-DATACOM/DB Facility status checking for record-at-a-time statements is shown in the next section.

6.7.7 Record-at-a-time Example

The following is an SP Facility example of how to code a generic search. In the example, all records that match the employee IDs from an input "transaction" file must be deleted. This example assumes the following:

1. A dataview named GENERIC has been defined to CA-DATADICTIONARY and consists of the five-character employee ID.

2. The database contains two tables: PMF and PAY. PMF contains the employee address, and PAY contains salary information.

3. A valid input file of employee IDs that are to be deleted from the database has been defined. (The paragraph for reading and validating the employee IDs in the input file and PERFORMing the LOCATE-RECORD paragraph is not included in the example.)

In the LOCATE-RECORD paragraph, a generic search is performed to find a record matching the input employee ID, INPUT-EMP-ID, with the GENERIC dataview. When a match is found, the request area field GENERIC-RA-FILE, which is generated for the dataview and contains the three-character table ID, is checked to determine the table containing the record. This is done to identify which dataview can read the record.

Once a match and the table in which it is located have been determined, control passes to the DELETE-EMPLOYEE or DELETE-PAYROLL paragraph. The dataview appropriate to the table (EMPLOYEE or PAYROLL) is positioned over the GENERIC dataview with the LOCATE AT statement. The record is then read and deleted with the CA-DATACOM/DB Facility READ and DELETE statements, respectively.

Note that the request area field *dataview-name*-RA-STATUS-CODE, which is updated after each attempt to access a database, is checked after each LOCATE, READ, and DELETE statement. If this field contains spaces, no CA-DATACOM/DB error code was issued, and normal processing may proceed. In this example, the CA-DATACOM/DB return code is displayed if the status code field does not contain spaces. (A list of CA-DATACOM/DB return codes is given in Appendix E of the CA-MetaCOBOL+ *Program Development Reference - DATACOM/DB* manual).

```
DATA DIVISION.
DATAVIEW GENERIC
   ACCESS IS GENERIC KEY IS NUMBER
   DBID.
DATAVIEW EMPLOYEE
   DICTIONARY NAME IS DLEMPLOYEE
   ACCESS KEY IS NUMBER
   DBID.
DATAVIEW PAYROLL
   DICTIONARY NAME IS DLPAYROLL
   ADDESS KEY IS NUMBER
   DBID.
01  INPUT-RECORD.
    05  INPUT-EMP-ID    PIC X(05).
    05  FILLER          PIC X(75).
PROCEDURE DIVISION.
    ENTER-DATACOM-DB
    OPEN
       .
       .
       .
```

```
LOCATE-RECORD.
    LOCATE GENERIC
       WHERE NUMBER EQUAL INPUT-EMP-ID
    IF GENERIC-RA-STATUS-CODE EQUAL SPACES
       IF GENERIC-RA-FILE EQUAL 'PMF'
          PERFORM DELETE-EMPLOYEE
       ELSE
          IF GENERIC-RA-FILE EQUAL 'PAY'
             PERFORM DELETE-PAYROLL
          ENDIF
       ENDIF
    ENDIF
              .
              .
              .
DELETE-EMPLOYEE.
    LOCATE EMPLOYEE AT GENERIC
    IF EMPLOYEE-RA-STATUS-CODE EQUAL SPACES
       READ EMPLOYEE AND HOLD
       IF EMPLOYEE-RA-STATUS-CODE EQUAL SPACES
          DELETE EMPLOYEE
          IF EMPLOYEE-RA-STATUS-CODE EQUAL SPACES
             DISPLAY 'PMF RECORD DELETED FOR EMPLOYEE'
                      EM-IDENTIFICATION-NUMBER
          ELSE
             DISPLAY 'EMPLOYEE DELETE ERROR '
                      EMPLOYEE-RA-STATUS-CODE
          ENDIF
       ELSE
          DISPLAY 'EMPLOYEE READ ERROR '
                   EMPLOYEE-RA-STATUS-CODE
       ENDIF
    ELSE
       DISPLAY 'EMPLOYEE LOCATE AT ERROR '
                EMPLOYEE-RA-STATUS-CODE
    ENDIF
```

```
DELETE-PAYROLL.
    LOCATE PAYROLL AT GENERIC
    IF PAYROLL-RA-STATUS-CODE EQUAL SPACES
       READ PAYROLL AND HOLD
       IF PAYROLL-RA-STATUS-CODE EQUAL SPACES
          DELETE PAYROLL
          IF PAYROLL-RA-STATUS-CODE EQUAL SPACES
             DISPLAY 'PMF RECORD DELETED FOR PAYROLL '
                      RC-NUMBER
          ELSE
             DISPLAY 'PAYROLL DELETE ERROR '
                      PAYROLL-RA-STATUS-CODE
          ENDIF
       ELSE
          DISPLAY 'PAYROLL READ ERROR '
                   PAYROLL-RA-STATUS-CODE
       ENDIF
    ELSE
       DISPLAY 'PAYROLL LOCATE AT ERROR '
                PAYROLL-RA-STATUS-CODE
    ENDIF
```

6.8 Physical Sequential Statements

Physical sequential processing bypasses the CA-DATACOM/DB index, retrieving data through physical block reads. This technique provides a faster method of passing a large volume of data than record-at-a-time sequential retrieval.

The LOCATE PHYSICAL [AND HOLD] *dataview-name* statement sets the start of a data area for physical block reads. The READ PHYSICAL *dataview-name* statement reads the data area specified in the LOCATE PHYSICAL statement by physical block. The *dataview-name* must be specified using the DATAVIEW statement shown in the example in Section 6.5, "DATAVIEW Statement."

The following is an SP Facility example of how to code physical sequential statements.

```
DATAVIEW EMPLOYEE
    DICTIONARY NAME IS DLEMPLOYEE
    ACCESS IS PHYSICAL
             .
             .
             .
PROCEDURE DIVISION.
    LOCATE PHYSICAL EMPLOYEE
    READ PHYSICAL AND HOLD EMPLOYEE
```

6.9 Logging Statements

The CA-DATACOM/DB logging system consists of the Log Area and Recovery Area, which have the CA-DATACOM/DB designations LXX and RXX, respectively. The Log Area is required to perform transaction backout. Both the Log Area and the Recovery File, which is a tape file spilled from the Log Area, are required to perform recovery.

The Log Area is created and maintained by the DBA. Therefore you can use the CA-DATACOM/DB Facility logging statements only if the DBA has specified LOGGING=YES in the CA-DATACOM/DB Master List; otherwise, the statements that control logging functions are ignored. The Recovery Area, which is required for transaction backout, may be used only if the DBA has specified LOGRCV=YES in the CA-DATACOM/DB Master List.

The following statements control logging functions. Note that as a CA-DATACOM/DB Facility programmer, you are responsible for defining the eight-byte I/O area identifier for the WRITE LOG, READ LOG, and CHECKPOINT LOG statements.

- WRITE LOG *ioarea-identifier* writes records from the Log Area. The identifier contains the length of the data read from the first four bytes, followed by the data itself.

- READ LOG *ioarea-identifier* reads records from the Log Area. The identifier contains the length of the data read from the first four bytes, followed by the data itself.

- CHECKPOINT LOG *ioarea-identifier* issues a checkpoint, indicating that all records before the checkpoint were processed successfully. The checkpoint ioarea-identifier is an eight-byte data item that is written to the log area when the checkpoint is issued.

- BACKOUT LOG reverses all transactions since the last checkpoint. This operation is ignored unless the CA-DATACOM/DB User Requirements Table is defined for transaction backout.

CA-DATACOM/DB maintains log records in the Log Area for transaction backout until your job terminates normally or issues a checkpoint with the CHECKPOINT LOG statement.

CA-DATACOM/DB logging, recovery/restart, and transaction backout function independently of any logging commands in your application program. That is, the logging system, if established in the Master List with LOGGING=YES, already provides these services. Therefore, use the WRITE/READ LOG command only if there is a special need to write data to the Log Area, and use the BACKOUT LOG command only if there is a special need to reverse maintenance changes made by the program to the database.

The CHECKPOINT LOG command is used for checkpoint processing. If transaction backout is used in a batch job that updates a large number of records, a backout can take a long time. Issuing CHECKPOINT LOG periodically limits the size of the transaction backup. Since the CICS Service Facility issues a checkpoint (with the DEQUE command) at the end of the CICS task, a CHECKPOINT LOG command is unnecessary at this point.

The following is an example of how to use the CHECKPOINT LOG command. In this example, CHECKPOINT LOG is issued for every 100 records that have been updated.

```
01  LOG-IOAREA.
    05  LOG-ID  PIC X(06)  VALUE 'TESTPGM'.
    05  LOG-NO  PIC 9(03)  VALUE 0.
              .
              .
              .
UPDATE-RECORDS.
    READ AND HOLD EMPLOYEE
      WHERE NUMBER EQUAL INPUT-EMP-ID
    IF EMPLOYEE-RA-STATUS-CODE EQUAL SPACES
       ADD 1 TO UPDATE-COUNTER
       IF UPDATE-COUNTER > 100
          ADD 1 TO LOG-NO
          CHECKPOINT LOG LOG-IOAREA
          DISPLAY        '   PROCESSED   ' LOG-IOAREA
          MOVE ZEROS TO UPDATE-COUNTER
       ENDIF
    ENDIF
              .
              .
              . logging:example
```

7. CA-DATACOM/DB Facility Sample Program

This section contains three CA-DATACOM/DB Facility programming examples. You can run these programs provided your CA-DATACOM/DB installation contains the HUMAN RESOURCE database as distributed in CA-DATACOM/DB Release 7.4 (DSGEN 1.8) or later.

Before attempting to run one of the sample programs, you must make sure that the dataviews and program contained in the examples are defined as entity-occurrences in CA-DATADICTIONARY and that the appropriate URT parameters have been set. Refer to Chapter 4 and the CA-MetaCOBOL+ *User Guide* for other preliminary considerations, including how to create JCL for translating, compiling, link-editing, and executing CA-DATACOM/DB Facility programs.

These examples are SP Facility programs. For descriptions of SP Facility statements, refer to the CA-MetaCOBOL+ *Structured Programming Guide*.

7.1 Record-at-a-time Example

This example shows the SP Facility-CA-DATACOM/DB Facility input and generated COBOL output for a program requesting CA-DATACOM/DB record-at-a-time processing. The HUMAN RESOURCE database is accessed using the EMPLOYEE and PAYROLL dataviews, which are described in examples throughout this manual.

Note the use of the SP Facility SELECT, LOOP, and IF constructs in the Input listing.

```
 OPTION IXIT=ADRXIXIT
 OPTION DDID=002,PSTAT=TEST,PVER=000
*$COPY SPP
*$COPY DLM
 IDENTIFICATION DIVISION.
 PROGRAM-ID.  DLPROB3.
 AUTHOR.  CAI.
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 SOURCE-COMPUTER.  IBM-370.
 OBJECT-COMPUTER.  IBM-370.
 DATACOM SECTION.
     ID-AREA IS ID-AREA-IDENTIFIER
     PRINT GEN.
 DATA DIVISION.
 WORKING-STORAGE SECTION.
 01  ID-AREA-IDENTIFIER VALUE 'DLPROB3' PIC X(32).
 01  FLAG EMPLOYEE-FOUND IS TRUE.
 DATAVIEW PAYROLL
     DICTIONARY NAME IS DLPAYROLL
     ACCESS KEY IS NUMBER.
 DATAVIEW EMPLOYEE
     DICTIONARY NAME IS DLEMPLOYEE
     ACCESS KEY IS NUMBER.
 PROCEDURE DIVISION.
 MAIN-MODULE.
     ENTER-DATACOM-DB
     DO LOCATE-EMPLOYEE
     IF EMPLOYEE-FOUND
         LOOP
             DO FIND-PAYROLL-INFO
             DO READ-EMPLOYEE-INFO
         UNTIL EMPLOYEE-RA-STATUS-CODE NOT = SPACES
         ENDLOOP
     ELSE
         DISPLAY 'NO EMPLOYEES MEET CRITERIA'
     ENDIF
     GOBACK
```

Continued on next page

```
       LOCATE-EMPLOYEE.
           LOCATE EMPLOYEE
           WHERE NUMBER GE 100
           SELECT FIRST ACTION FOR EMPLOYEE-RA-STATUS-CODE
           WHEN SPACES
               SET-TRUE EMPLOYEE-FOUND
           WHEN '14'
               SET-FALSE EMPLOYEE-FOUND
               DISPLAY 'NO EMPLOYEES MEET CRITERIA'
           WHEN NONE
               SET-FALSE EMPLOYEE-FOUND
               DISPLAY 'ERROR IN LOCATE='
EMPLOYEE-RA-STATUS-CODE
           ENDSELECT
           READ EMPLOYEE
           IF EMPLOYEE-RA-STATUS-CODE NOT = SPACES
               DISPLAY 'ERROR IN 1ST EMPLOYEE READ='
                       EMPLOYEE-RA-STATUS-CODE
               SET-FALSE EMPLOYEE-FOUND
           ENDIF
       READ-EMPLOYEE-INFO.
           READ NEXT EMPLOYEE
           SELECT FIRST ACTION FOR EMPLOYEE-RA-STATUS-CODE
           WHEN SPACES
           WHEN '14'
           WHEN NONE
               DISPLAY 'ERROR=' EMPLOYEE-RA-STATUS-CODE
                       ' ON READ OF EMPLOYEE DATA FOR '
                       EMPLOYEE-RA-KEY-VALUE
           ENDSELECT
       FIND-PAYROLL-INFO.
           READ  PAYROLL
           WHERE NUMBER EQ EM-IDENTIFICATION-NUMBER
           SELECT FIRST ACTION FOR PAYROLL-RA-STATUS-CODE
           WHEN SPACES
               DO DISPLAY-EMPLOYEE-INFO
           WHEN '14'
               DISPLAY 'NO PAYROLL INFORMATION FOR EMPLOYEE '
                       EM-IDENTIFICATION-NUMBER
           WHEN NONE
               DISPLAY 'ERROR= ' PAYROLL-RA-STATUS-CODE
                       ' IN PROCESSING PAYROLL FOR '
                       EM-IDENTIFICATION-NUMBER
           ENDSELECT
       DISPLAY-EMPLOYEE-INFO.
           DISPLAY EM-IDENTIFICATION-NUMBER
EM-IDENTIFICATION-NAME
                   EM-STATE-ADDRESS RC-CURRENT-RATE
RC-ACTIVITY-CODE
                   RC-ACTIVITY-STATUS
                        .  .  .
```

7.2 Set-at-a-time Example

The following is the CA-DATACOM/DB Facility input for a program accessing the sample PARTS database, which contains sample parts and inventory records. (The PARTS database is described in the CA-DATACOM/DB *Application Programming Guide.*)

This program uses the PARM field of the EXEC statement to pass a date, INP-DATE, and a logging interval (i.e., number of transactions between checkpoints), INP-INTERVAL, to the program. The purpose of the program is to delete all records defined by the PART-COST dataview with an effective date, EFFDT, that is greater than INP-DATE.

```
 OPTION IXIT=ADRXIXIT
 OPTION DDID=002,PSTAT=TEST,PVER=000
*$COPY SPP
*$COPY DLM
 IDENTIFICATION DIVISION.
 PROGRAM-ID.  DELPNC.
 AUTHOR.  CAI.
 ENVIRONMENT DIVISION.
 DATACOM SECTION.
     ID-AREA IS ID-AREA-IDENTIFIER
     PRINT NOGEN.
 DATA DIVISION.
 WORKING-STORAGE SECTION.
 DATAVIEW  PART-COST.
 01  ID-AREA-IDENTIFIER    PIC X(32)   VALUE 'DELPNC'.
 01  WS-REC-COUNTS         COMP-3.
     05  DELETE-COUNT      PIC S9(05)  VALUE  +0.
     05  LOG-COUNT         PIC S9(05)  VALUE  +0.
 01  LOG-IOAREA.
     05  LOG-ID            PIC X(06)   VALUE  'DELPNC'.
     05  LOG-NO            PIC 9(02)   VALUE  0.
 LINKAGE SECTION.
 01  INPUT-PARM.
     05  FILLER            PIC S9(04)  COMP.
     05  INP-DATE          PIC X(06).
     05  INP-INTERVAL      PIC 9(05).
```

Continued on next page

```
       PROCEDURE DIVISION.
       MAIN-MODULE.
           ENTER-DATACOM-DB  USING  INPUT-PARM
           FOR EACH  PART-COST
               WHERE (EFFDT > INP-DATE)
               HOLD RECORDS
               COUNT  IN  DELETE-COUNT
               ORDER RECORDS ON ASCENDING PN
                  ADD  1  TO  LOG-COUNT
                  IF  LOG-COUNT  >  INP-INTERVAL
                      DO  LOGGING
                  ENDIF
                  DELETE  PART-COST
               WHEN END
                   DISPLAY 'TOTAL RECORDS DELETED '  DELETE-COUNT
               WHEN ERROR
                   DISPLAY '***CA-DATACOM/DB ERROR'
DB-STATUS-CODE '***'
           ENDFOR
           GOBACK

       LOGGING.
           ADD   1  TO  LOG-NO
           CHECKPOINT LOG  LOG-IOAREA
           DISPLAY  PN  '  PROCESSED  '  LOG-IOAREA
           MOVE  0  TO  LOG-COUNT
```

7.3 CICS Example

The following program is a CICS program using CA-DATACOM/DB Facility statements to display EMPLOYEE records requested from the PMF table of the HUMAN RESOURCE database. The employee number is the input field.

Note the use of the KEY-VALUE keyword in the READ WHERE statement of the LOCATE-EMP paragraph. KEY-VALUE is not a database or CA-DATADICTIONARY key name; rather, it specifies the first or only key defined for the dataview.

```
            OPTION IXIT=ADRXIXIT
            OPTION DDID=002,PSTAT=TEST,PVER=000
           *$COPY SPP
           *$COPY DLM
            IDENTIFICATION DIVISION.
            PROGRAM-ID.  DLPROG.
            AUTHOR.  CAI.
            ENVIRONMENT DIVISION.
            CONFIGURATION SECTION.
            SOURCE-COMPUTER.  IBM-370.
            OBJECT-COMPUTER.  IBM-370.
            DATACOM SECTION.
                MONITOR IS CICS
                ID-AREA IS ID-AREA-IDENTIFIER
                PRINT NOGEN.
            DATA DIVISION.
            WORKING-STORAGE SECTION.
            01  ID-AREA-IDENTIFIER VALUE 'DLPROG' PIC
     X(32).
            01  MAP2I COPY 'ONLMAP2'.
            DATAVIEW EMPLOYEE
                DICTIONARY NAME IS PRIN-PER-DVW
                ACCESS KEY IS NUMBER
                DBID.
            LINKAGE SECTION.
            01  DFHCOMMAREA.
                02  DUMMY          PICTURE  X.
```

Continued on next page

```
        PROCEDURE DIVISION.
        EMPLOYEE-INQ.
            EXEC CICS
                HANDLE CONDITION MAPFAIL(DISPLAY-EMP)
            END-EXEC
            EXEC CICS
                HANDLE AID PF1(END-SESSION)
            END-EXEC
            DO GET-EMP
            DO LOCATE-EMP
            DO DISPLAY-EMP
            GOBACK
        END-SESSION.
            EXEC CICS RETURN END-EXEC
            GOBACK
        GET-EMP.
            EXEC CICS
                RECEIVE MAP('ONLMAP2')
                MAPSET('ONLMAP2')
            END-EXEC
        LOCATE-EMP.
            READ EMPLOYEE
            WHERE KEY-VALUE = EMPNOI
            SELECT FIRST ACTION FOR
 EMPLOYEE-RA-STATUS-CODE
            WHEN SPACES
                MOVE EM-IDENTIFICATION-NAME TO EMPNAMO
            WHEN '14'
                MOVE 'NO SUCH EMPLOYEE' TO EMPNAMO
            WHEN NONE
                MOVE 'ERROR ON READ' TO EMPNAMO
            ENDSELECT
        DISPLAY-EMP.
            EXEC CICS
                SEND MAP('ONLMAP2')
                MAPSET('ONLMAP2')
                ERASE
            END-EXEC
            EXEC CICS
                RETURN TRANSID('ONL1')
            END-EXEC
```

8. CA-DATACOM/DB Facility Statement Formats

The following formats describe CA-DATACOM/DB Facility statement syntax. Note that these formats do not represent complete syntax; they are intended as a quick reference. For additional information on CA-DATACOM/DB Facility syntax, refer to Chapter 6 or the CA-MetaCOBOL+ *Program Development Reference - CA-DATACOM/DB* manual.

Environment Division Statements

```
ENVIRONMENT DIVISION.
DATACOM SECTION.
   [MONITOR IS CICS]
   [PRINT {GEN   } ]
   [      {NOGEN} ]
   [ID-AREA IS id-area-identifier].
DATA DIVISION.
WORKING-STORAGE SECTION.
01 program-id-area VALUE 'program-name' PIC X(32).
         .
         .
         .
```

Data Division Statements

Record-at-a-time DATAVIEW Statement

Generic Search

```
DATAVIEW dataview-name
   [CA-DATADICTIONARY NAME IS dd-dataview-name]
   [PREFIX IS 'data-name-prefix-literal']
    ACCESS IS GENERIC KEY IS dd-key-name [dd-key-name-2 ...]
   [DATA-BASE-ID [IS data-name]].
```

Specific Search

```
DATAVIEW dataview-name
   [DATADICTIONARY NAME IS dd-dataview-name]
   [PREFIX IS 'data-name-prefix-literal']
    ACCESS KEY IS dd-key-name [dd-key-name-2 ...]
   [DATA-BASE-ID [IS data-name]].
```

Physical Sequential DATAVIEW Statement

```
DATAVIEW dataview-name
   [DATADICTIONARY NAME IS dd-dataview-name]
   [PREFIX IS 'data-name-prefix-literal']
    ACCESS IS PHYSICAL
   [DATA-BASE-ID [IS data-name]].
```

Set-at-a-time DATAVIEW Statement

```
DATAVIEW dataview-name
   [DATADICTIONARY NAME IS dd-dataview-name]
   [PREFIX IS 'data-name-prefix-literal']
   [DATA-BASE-ID [IS data-name]].
```

Procedure Division Statements

```
PROCEDURE DIVISION.
```

Entrance from batch URT interface

```
ENTER-DATACOM-DB
```

OPEN batch URT if URT defined as OPEN=USER

```
OPEN
```

Note: This option applies to batch only

Physical Sequential Processing

```
LOCATE PHYSICAL [AND HOLD] dataview-name
READ PHYSICAL dataview-name
```

Record-at-a-time Processing

- **Database Maintenance**

```
REWRITE dataview-name [FROM alternate-workarea-identifier]

WRITE dataview-name [FROM alternate-workarea-identifier]

DELETE dataview-name
```

- **Index Search**

```
LOCATE dataview-name

          {'db-key-name-literal'}  {<=}
    WHERE { dd-key-name          }  {>=}  {'key-value-literal'  }
          { KEY-VALUE            }  {= }  { key-value-identifier}

LOCATE dataview-name-1 AT dataview name-2

LOCATE NEXT [DUPLICATE]
            [KEY      ] dataview-name


LOCATE PREVIOUS dataview-name

LOCATE dataview-name

          {'db-key-name-literal'}
    WHERE { dd-key-name          } IS {ident1} THRU {ident2}
          { KEY-VALUE            }    {'lit1'}      {'lit2'}

LOCATE [NEXT] dataview-name
```

- **Random Retrieval**

```
 READ [AND HOLD] dataview-name

[       {'db-key-name-literal'}  {<=}  {'key-value-literal'  } ]
[ WHERE { dd-key-name          }  {>=}  { key-value-identifier} ]
[       { KEY-VALUE            }  {= }  {'record-id-literal'  } ]
[       { ID                   }        { record-id-identifier} ]


 READ NEXT [DUPLICATE] [AND HOLD] dataview-name

 READ [AND HOLD] PREVIOUS dataview-name

 READ [AND HOLD] dataview-name WITHIN RANGE

          {'db-key-name-literal'}
    WHERE { dd-key-name          } IS {ident1} THRU {ident2}
          { KEY-VALUE            }    {'lit1'}      {'lit2'}

 READ [AND HOLD] NEXT dataview-name WITHIN RANGE
```

- **Sequential Retrieval (batch processing only)**

```
    LOCATE SEQUENTIAL dataview-name
             {'db-key-name-literal'}
       WHERE { dd-key-name         }  >=  {'key-value-literal'  }
             { KEY-VALUE           }      { key-value-
   identifier}

    READ SEQUENTIAL dataview-name
```

Set-at-a-time Processing

```
FOR {EACH
    {FIRST {record-count-identifier}  }
    {ANY   {record-count-literal   }  }

     dataview-name {RECORD }
                   {RECORDS}
    [WHERE (condition)]

    [HOLD {RECORD } ]
    [     {RECORDS} ]

    [COUNT IN retrieval-count-identifier]

    [ORDER [UNIQUE] RECORDS ON
    [
    [ { [ASCENDING ]                    }     ]
    [ { [DESCENDING] dataview-identifier } ... ]

        [process-1] ...
    [WHEN END         ]
    [    process-2 ...]

    [WHEN ERROR       ]
    [    process-3 ...]

    [WHEN NONE        ]
    [    process-4 ...]

ENDFOR (if SPP)

SKIP dataviewname {FORWARD  } <BY> dataname/literal <RECORDS>
                  {BACKWARD }

        {FIRST}
SKIP TO {LAST } dataviewname <RECORD>
        (SAME }
```

Releasing Exclusive Control

- **Record-at-a-time Processing**

  FREE LAST *dataview-name*

  FREE ALL *dataview-name*

- **Set-at-a-time Processing**

  ```
  FREE SET dataview-name .   .   .
       .
       .
       .
  ```

- **CLOSE batch URT if URT defined as OPEN=USER**

  ```
  CLOSE
  GOBACK
  ```

Index