HUD-0050556

PB-251 134

ORIGINAL

TCDMS SYSTEM PROGRAMMER'S MANUAL

(Data Management Section)

January 1976

Department of Housing and Urban Development
Office of the Assistant Secretary for
Policy, Development & Research

HUD Contract H-2073-R

INTER-REGIONAL INFORMATION SYSTEM

Regional Information Systems Department
Lane County Courthouse
Eugene, Oregon 97401

and

Data Processing Authority
4747 East Burnside
Portland, Oregon 97215

BIBLIOGRAPHIC DATA SHEET	1. Report No. USACLCG20014B	2.	3. Recipient's Accession No

4. Title and Subtitle	5. Report Date
TCDMS System Programmer's Manual Volume II (Data Management Section)	January 30, 1976
	6.
7. Author(s)	8. Performing Organization Report No.
9. Performing Organization Name and Address Lane County Government Lane County Courthouse Eugene, Oregon 97401	10. Project/Task/Work Unit No.
	11. Contract/Grant No. H-2073-R
12. Sponsoring Organization Name and Address U.S. Department of Housing & Urban Development Office of Policy, Development & Research 451 7th St., S.W. Washington, D.C. 20410	13. Type of Report & Period Covered Special Technical Report
	14.

15. Supplementary Notes

16. Abstracts

This manual is from a USAC series produced by the Regional Information Systems Department of Lane County. It contains a description of the data base definition process for the Data Management component of the Telecommunications Data Management System.

17. Key Words and Document Analysis. 17a. Descriptors

Information System
Local Government
Computer System Programs
Data Retrieval

17b. Identifiers/Open-Ended Terms

Urban Information Systems Inter-Agency Committee
Municipal Information System
Lane County
Data Management System

17c. COSATI Field/Group 5B

18. Availability Statement	19. Security Class (This Report)	21. No. of Pages
released for distribution by NTIS	unclassified	87
	20. Security Class (This Page)	22. Price
	unclassified	

FORM NTIS-35 (REV. 3-72)

USCOMM-D.C. 14952-P72

THE CITATION OF TRADE NAMES OR MANUFACTURERS IN THIS REPORT DOES NOT CONSTITUTE AN OFFICIAL ENDORSEMENT OR APPROVAL OF THE USE OF SUCH EQUIPMENT OR SOFTWARE.

0.1 PREFACE

In 1972, the Data Processing Authority (representing the city of Portland and Multnomah County, Oregon) and the Regional Information Systems Department of the Lane County government (representing the cities of Eugene, Springfield, Albany, Cottage Grove, and Florence; and Lane, Linn and Benton Counties, Oregon) formed an organization called the Inter-Regional Information System (IRIS). Its purpose was manifold:

> to solve some of the complex problems of public information handling through cooperative planning and development of hardware and software environments;
>
> to minimize the duplication of effort involved in writing application systems;
>
> to reduce the cost of governmental data processing; and
>
> to increase the quality of service to the taxpayer.

Since its inception, the IRIS organization has grown to represent over one hundred different city, county, state, and federal agencies serving over 70% of Oregon's population. Current projects include the Fleet Management System, the Assessment and Taxation System, and the Telecommunications Data Management System. Future involvement is anticipated in the areas of criminal justice, management analysis, human resources, geo-coding, and financial systems.

Much of the inter-regional success enjoyed by the IRIS organization has been facilitated by a cost-reimbursement contract with the Urban Information Systems Inter-Agency Committee (USAC). USAC is a consortium of ten federal agencies formed in 1968 to work together

0.1 PREFACE

In 1972, the Data Processing Authority (representing the city of Portland and Multnomah County, Oregon) and the Regional Information Systems Department of the Lane County government (representing the cities of Eugene, Springfield, Albany, Cottage Grove, and Florence; and Lane, Linn and Benton Counties, Oregon) formed an organization called the Inter-Regional Information System (IRIS). Its purpose was manifold:

> to solve some of the complex problems of public information handling through cooperative planning and development of hardware and software environments;
>
> to minimize the duplication of effort involved in writing application systems;
>
> to reduce the cost of governmental data processing; and
>
> to increase the quality of service to the taxpayer.

Since its inception, the IRIS organization has grown to represent over one hundred different city, county, state, and federal agencies serving over 70% of Oregon's population. Current projects include the Fleet Management System, the Assessment and Taxation System, and the Telecommunications Data Management System. Future involvement is anticipated in the areas of criminal justice, management analysis, human resources, geo-coding, and financial systems.

Much of the inter-regional success enjoyed by the IRIS organization has been facilitated by a cost-reimbursement contract with the Urban Information Systems Inter-Agency Committee (USAC). USAC is a consortium of ten federal agencies formed in 1968 to work together with local governments across the United States in an effort to improve urban governance through more effective use of computer-based processing systems. USAC is sponsoring several research and development projects which will result in transferable, computerized information systems available to local governments throughout the United States.

With the support of USAC, IRIS is developing the system software foundation for the application programs which control these computerized systems. This foundation is called TeleCcommunications/Data Management System (TCDMS). This system contains two components which bring together the state-of-the-art features in both telecommunications and data base/date management systems.

The telecommunications component of TCDMS extends the power of the modern computer to the desk of each user. Its facilities include such features as terminal independent I/O functions, user-specified security, multiprogramming, priority scheduling, message switching, print-out spooling, on-line debugging, and remote job entry.

The data base/data management component of TCDMS optimizes the efficiency of data file construction and minimizes data redundancy by combining all files in the system into an integrated data base. Its facilities include data access flexibility, file and data element security, and application program independence from the physical file structure.

Perhaps the most important feature of TCDMS is the transferability of application systems it allows. Application programs running under TCDMS control are isolated from changes in the hardware or software configuration of the installation. This means that TCDMS-controlled application systems can be transferred between IRIS installations without the costly conversion efforts usually necessitated by such exchanges.

TCDMS may be implemented on any IBM System 360/370 computer having 252K bytes or more of storage capacity. It will support IBM System 360/370 BAL, FORTRAN, COBOL, and DL/1 user languages. TCDMS will run in real core under the control of IBM OS or VS operating systems. The modular construction of TCDMS makes it hardware independent; it can operate with any IBM terminal hardware configuration.

The joint software development and maintenance by means of the regional and interregional cooperation of IRIS and the integrated data base/data communications approach of TCDMS are becoming an increasingly popular solution to the problems of information handling in the public domain.

TABLE OF CONTENTS

0.2 INTRODUCTION

This manual is a reference guide for TCDMS system programmers and data base administrators at TCDMS installations. It contains a description of the TCDMS functions which define a data base and which create a data base control block (DBCB). It is written with the assumption that its readers have a working knowledge of data base design. For this reason, it offers no instruction in general data base concepts. The user of this manual should have designed the structure of his installation's data base. The information in this manual enables him to create the TCDMS system files which both identify this structure to TCDMS and allow the application programmers at the installation to access the data within the structure.

This manual has three main sections. They are:

> OVERVIEW OF THE DATA MANAGEMENT SYSTEM - there is a general overview at the beginning of the manual which describes the data management component of TCDMS.
>
> DEFINING THE DATA BASE STRUCTURE - this section describes the TCDMS functions which define data elements, segments, and the relationships between them. These functions create the Data Dictionary and the Segment Dictionary which define the data base structure to TCDMS.
>
> GENERATING THE DATA BASE CONTROL BLOCK - this section describes the process by which the data base control block is created.

In addition, this manual also contains a glossary of terms that are of particular interest to TCDMS programmers.

INTRODUCTION Continued

For other information about TCDMS, the reader is referred to the following documents:

TCDMS SYSTEM SUMMARY - a conceptual overview of TCDMS.

TCDMS APPLICATION PROGRAMMER'S MANUAL (Data Management Section) - a description of the TCDMS data base access functions available to application programmers at TCDMS installations.

TCDMS UTILITIES MANUAL - a description of the TCDMS utility programs which load and maintain a TCDMS data base.

TCS SYSTEM PROGRAMMER'S MANUAL - a description of the TCDMS SYSGEN process.

1.0 OVERVIEW OF THE DATA MANAGEMENT SYSTEM

1.0 Overview of the Data Management System

The data management component of TCDMS provides to an installation both a hierarchically structured data base with extensive capabilities for inter-relating data, and a data base access system which permits application programmers to retrieve, insert, change, or delete data in the data base.

1.1 The TCDMS Data Base

The TCDMS data base contains one or more data files which can be separately indexed, loaded, and reorganized. Data in one TCDMS file can be a pointer to data in another file, thus providing extensive data inter-relatability within the data base.

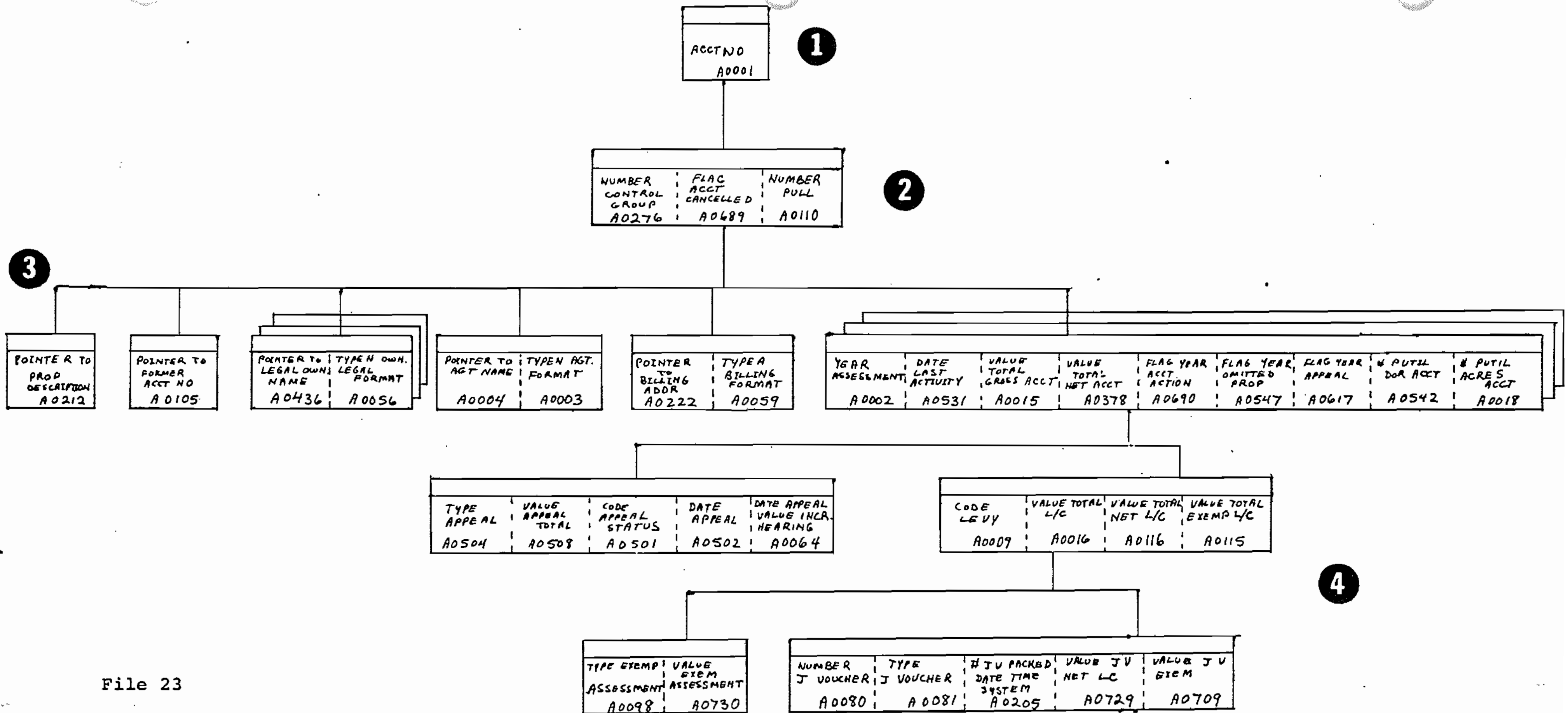
A data element is the unit of data handled by an application programmer who uses files organized and accessed by TCDMS. One or more related data elements are stored within a segment on the file. Only one occurrence of a particular data element is allowed on a segment. Any other occurrences of that data element require additional occurrences of the segment. Generally there is a one-to-one correspondence among the data elements in a multi-element segment. TCDMS treats the segment as the smallest unit of data it reads from or writes to the files. When an application program requests retrieval of a data element which is stored in a multi-element segment, TCDMS reads the segment, extracts the particular data element desired, and passes this value to the requesting program. Similarly, when TCDMS writes data to the data base it writes an entire segment.

Segments are arranged in a hierarchical structure into a family. There can be up to 256 hierarchical levels in a family. Each family contains data of one type and structure. This data is all logically related to, and hierarchically dependent on, one segment - the root segment. Each family in a file is identified by the presence of this root segment. A root segment occurs only once in each family. Generally these segments are indexed. Groups of families make up a TCDMS file.

Data in the TCDMS data base can be related by pointers. These are segments (or data elements) in one file which define the location of a segment in another file. Rather than actually containing the data, a pointer segment contains information which describes where the data is stored in another file. Any segment which is the "target" of a pointer from another file contains back pointers which identify the segments which point to it. Pointers in one file always identify root segments of families in the pointed to file.

Diagram 1 shows one family in a file. This example file, the Public Utility Property File in an assessment and taxation data base, will be used throughout this manual for illustration. The structure of the families which comprise this file is explained briefly.

❶ This box represents the root segment for this particular family. The root segment is a single-element segment. It contains only an account number data element. The data element name A0001 which appears in the lower-right corner of the box has been assigned to this data element. Only one account number is stored in this segment on the file. All the data hierarchically dependent on this root segment pertains to the one account number stored in the segment.



File 23

Public Utility Property File

Diagram 1

2 The next lower hierarchical level on the file contains one multi-element segment. This segment contains three data elements: A0276, A0689, and A0110.

3 There are six segments on this hierarchical level. Each is dependent on the segment above (described in 2). Several of these segments have multiple occurrences. The current owner name segment can have several occurrences. There can be more than one owner for a piece of property and each owner's name requires a separate occurrence of the current owner name segment. There are multiple occurrences of the year segment as well. There is one for each year that the account number has been in use.

The first five segments on this hierarchical level contain pointers. These pointer data elements do not contain the referenced data, but they identify the root segment in another file which does contain the data. The pointed to files are illustrated in Diagram 2.

4 Dependent on each occurrence of the year segment there are two multi-element segments - the appeal segment and the levy code assessment segment. There are also two multi-element segments dependent on the levy code assessment segment.

The Public Utility Property file contains many occurrences of this account family structure. Each account in the file is represented by one family with the structure illustrated in Diagram 1. Because there are many accounts in the file, there are many occurrences of the family structure. Diagram 3 shows a simplified schematic view of this file. For clarity, only certain segments are depicted and the data elements within these segments are not illustrated. Notice that the entire account family structure is repeated for each


PROP
DESCR
RT0011
FORMER
ACCT
NUMBER
RT0012
NAME
RT0013
ADDR
RT0014

Diagram 2

Account families in File 23
Public Utility Property File

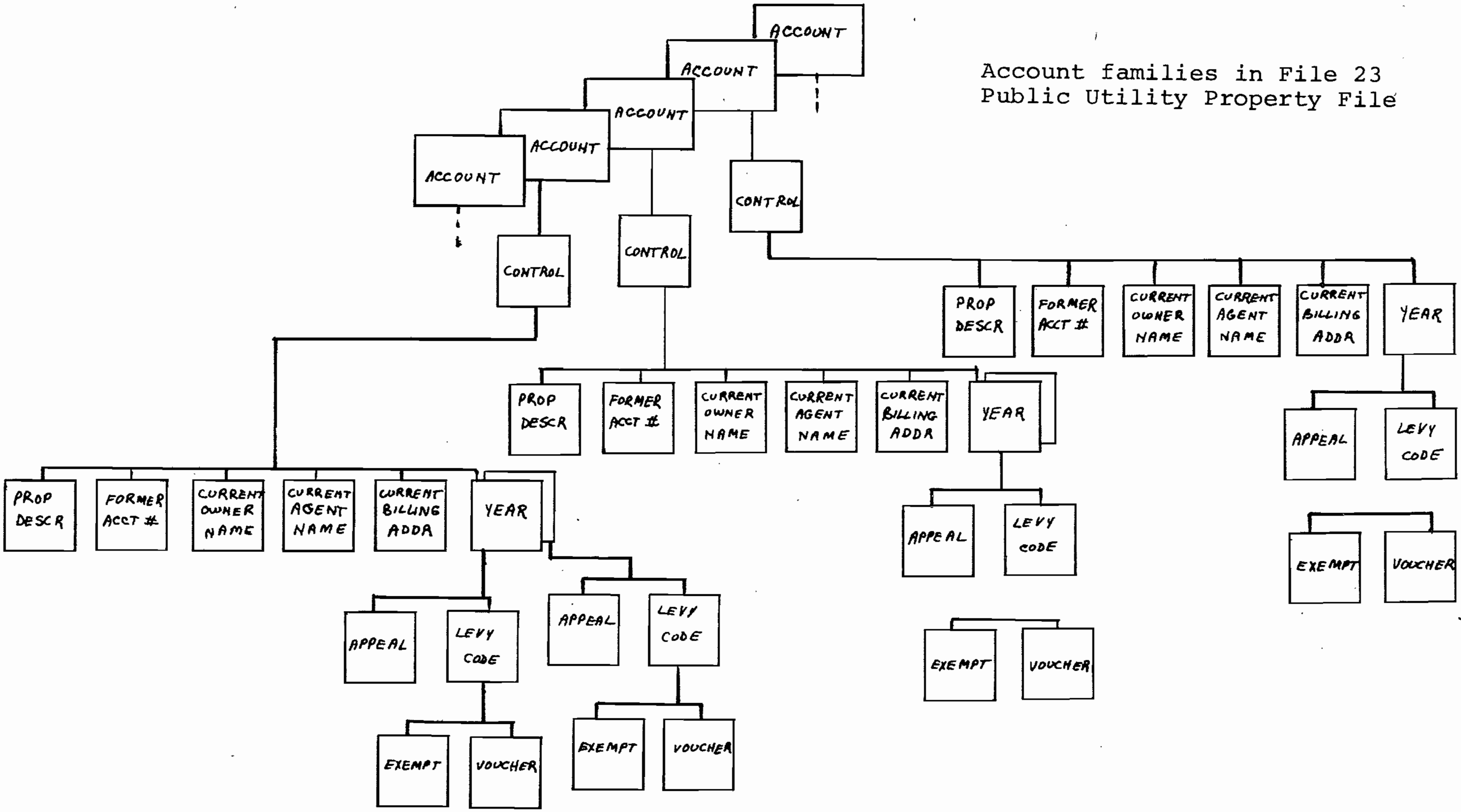


Diagram 3

account segment in the file. This three-dimensional view of the hierarchical data base is the view used by an application programmer who accesses the data base.

Within the TCDMS data base there may be two basic types of files. The shared or multiple chain files contain data that is common to several users. Individual users view this data through pointers in their own files. Therefore, the shared files can be "pointed to" from several applications. Notice that many of the segments in the Public Utility Property file contain pointers. Diagram 2 shows the pointed to files in this sample assessment and taxation data base. These are shared files. They are accessed through the Public Utility Property file and through other files not described here. They contain data (names, addresses, and property descriptions) used by many files within the data base.

The other variety of file, the root chain file, contains data for an individual application. A root chain file contains within it not only data relating to the particular application, but also pointers to related data in other files. The Public Utility Property file is a root chain file. It contains the data which relates specifically to the accounts for property owned by public utilities. It contains pointers to the related data (names, addresses, etc.) contained in other files.

1.2 Data Storage and Description

When a data base is created and installed at a data center, the exact descriptions of the files and the definitions of the data elements and segments contained in these files are established. The Data Dictionary, which defines the names and descriptive attributes for all data elements in the data base, is built. The Segment

Dictionary, which defines the hierarchical structure of the files, is built. These two system files, the Segment and Data Dictionaries, define the data base to TCDMS. Each installation determines the particular structure of its data base - which types of TCDMS files to include, what access limitations are imposed, and so forth. This data base definition process is a main topic of this System Programmer's manual. The process is described in detail in Chapter 2.0 Defining the Data Base Structure.

Data in a TCDMS file can be stored on disk in a variety of formats. TCDMS can either write data to the file exactly as it is supplied, or it can compress the data before storage. There are thirteen types of data compression available. When the data base is established at an installation, each data element is defined, and the compression type, if any, is set. TCDMS converts data to and from these compressed formats as it transfers the data to and from the data base. An application program normally handles only an external, or uncompressed form of the data element.

All the data elements in a TCDMS file are referenced by data element name. These data element names are unique names, determined by each installation to suit their particular data and storage requirements. TCDMS keeps a data element descriptor for each data element name, which contains both the external length and the stored length of the data element. Each data element descriptor contains the compression type and any accessibility specifications for the data element. The data element descriptor also contains information which describes the relationship between the data element and the segment and the location of the data element within the segment. These data element descriptors are stored in the Data Dictionary.

The Segment Dictionary defines the hierarchical structure of the files. Each entry in the Segment Dictionary relates a segment to the segment on the next higher hierarchical level. For pointer segments, each entry identifies the "pointed to" file. For root segments which are "pointed to" from another file, the Segment Dictionary identifies the file and segment which are pointing to the root segment. If there are several segments which point to the root segment, there are several Segment Dictionary entries.

A data base control block (DBCB) defines which data elements can be accessed by a particular application program. DBCBs are created at each installation by a data base administrator in consultation with users and application programmers. Each application program which accesses the data base must have a DBCB. The DBCB generation process is described in Chapter 3.0, Generating the DBCB.

The DBCB relates the physical structure of the data to the logical structure which the application program will use. The data base control block is associated with an application program either as the program is loaded or during execution before any data base access. It occupies a portion of the thread the program is using. The DBCB includes a list of all the data elements the program can use and a segment table TCDMS uses to locate the segments which contain these data elements. The DBCB also includes and formats an area of storage, the segment work area (SWA). The SWA is the area that TCDMS uses either when it extracts data elements from a retrieved segment before passing them to the program, or when it groups data elements into segments before inserting them into the data base. The DBCB also contains other TCDMS work areas.

The TCDMS data files are formatted and loaded onto direct access storage devices using the Data Management System file load utility programs. These utilities are described in the TCDMS Utilities Manual, Data Management Component.

1.3 TCDMS Data Base Access

There are four major functional areas in the TCDMS data base access system: access method, segment processor, request manager, and the file and family protection system.

Data base access requests from an application program are initially handled by the request manager modules. These modules determine the nature of the request, convert the request to the appropriate format for the access method, and handle conversion of data from its stored format to the format desired by the user. In addition, the request manager modules maintain positioning within the data base for direct or sequential access requests. The request manager modules handle data elements.

Within the data base, data elements are stored in segments. The request manager modules call the segment processor modules to handle data segments. The segment processor modules manage the segment work area (SWA) where segments are constructed from data elements (for insertion) or held during data element retrieval. The segment processor also determines whether segments are pointers and performs special processing for them. The segment processor modules call the access method modules for actual data base retrievals, insertions, or deletions.

The access method modules map the complex data structures to and from their physical representations on direct access storage. The access method groups segments into families and stores each family in one physical block on disk. A block can contain several families. If a family will not fit in one block, the access method splits it between two blocks and constructs the necessary linkages to relate the two areas of storage. The access method is designed so that the physical structuring of families is independent of the technique used to access families. This means that a family can be accessed in several ways. The access method index handling routines provide direct or sequential access by index. A family can be accessed through a pointer from another file. The access method also manages any buffers needed during data base physical access. The access method performs the actual read and write operations which access the data files on disk.

The file and family protection modules provide data base protection and restoration facilities. Data base protection assures data integrity during data access operations. In addition, modules within the data base protection system handle processing in the event that an application program which has accessed the data base terminates abnormally. The data base restoration facility includes the capture module and the file recovery modules. The capture module provides the basis for data base recovery in the event of a system failure. It writes a record of each data base update transaction (inserts, deletes, and changes) to the system capture file. The file recovery modules provide the capability to restore the data base to its original condition in the event it becomes damaged during a system failure. Incomplete transactions can be removed. Transactions from the capture file can be applied to a backup copy of the data base.

2.0 DEFINING THE DATA BASE STRUCTURE

2.0 Defining the Data Base Structure

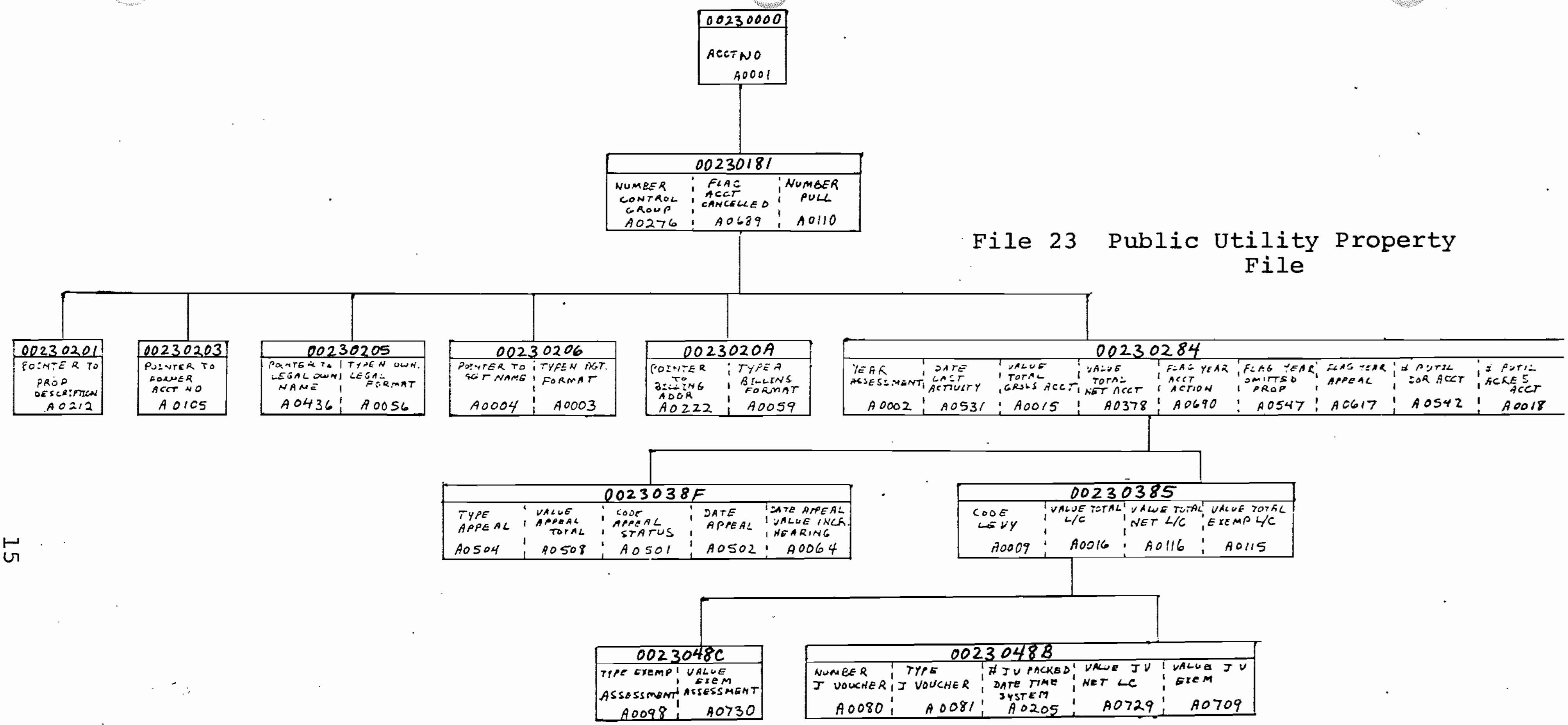
There are three things you must do to define the data base structure. (1) You must define the data elements to be included in the data base. (2) You must define the segments which contain these data elements. (3) You must identify the segments which are referenced by pointers from other files and redefine them to associate them with the pointers. TCDMS then sorts these definitions to form the Data Dictionary and Segment Dictionary.

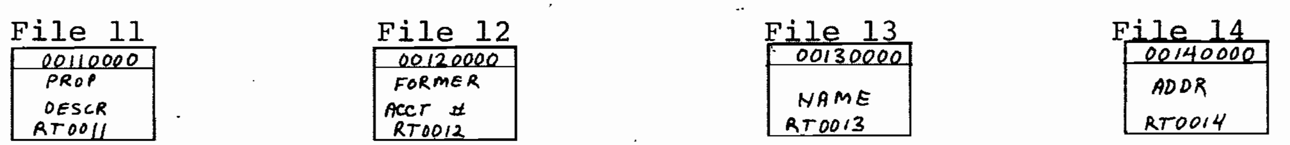
Note that when you define the structure of the data base you are only concerned with the two-dimensional view of the data base. Multiple occurrences of segments can be ignored. The data base structure defines only the possible relationships between segments on two hierarchical levels. Diagram 4 shows the two-dimensional view of the example data base. In File 23 there are 37 data elements in 12 segments which must be defined. Files 11 through 14 contain single-element segments. These files can all be accessed through pointers from the root chain file, File 23. Each segment must be defined and the linkages to the root chain file must also be defined. The sections which follow describe data element definition, segment definition, and segment redefinition for segments referenced by pointers.

2.1 Defining the Data Elements

When you define a data element you create an entry for the Data Dictionary. This entry identifies the data element name, and includes descriptive information about the data element. You define the data element compression type, the external length (used by an application program), and the internal length (used by TCDMS

File 23 Public Utility Property File





Property Description File

Former Account Number File

Name File

Address File

Diagram 4

when it stores or retrieves the data element). You can indicate the location of the data element within its containing segment. Any access limitations you specify for the data element are included in the Data Dictionary entry. Data element validation and sort information which you define is also included in the Data Dictionary entry.

Each data element and the corresponding Data Dictionary entry are identified by the data element name. Each data element name is a unique name, determined by each installation to suit its particular data and storage requirements. Data element names can be from one to six characters long. They must begin with an alphabetic character (A through Z). The remaining 5 characters can be either alphabetic (A-Z) or numeric (0-9). It is suggested that each installation develop data element naming standards for the data elements in their data base. For example, a one- or two-character prefix can be used to identify the particular application area which uses the data element, and the remaining four or five digits could be used for a sequential numbering scheme within that programming area. The example data base depicted in Diagram 4 uses such a scheme. It is an assessment and taxation data base; the data element names begin with the character A and are numbered sequentially. (The root segments to the multiple chain files, Files 11 through 14, are apparent exceptions to this standard. Because these data elements are accessed via pointers from File 23, they are referenced by the data element names illustrated in File 23. These names follow the standard. The relationships between pointers and the "pointed to" segments are described in Section 2.3).

Data in a TCDMS file can be stored on disk in a variety of formats. TCDMS can either write data to the file exactly as it is supplied, or it can compress the data before storage. There are thirteen types of data compression available. These represent combinations of ten external formats and seven internal storage formats. Table I, Data Compression Types and Codes, lists the data compression types.

Data compression allows TCDMS to use disk storage space more efficiently. Some savings are also realized in channel usage, because less data is transferred to and from the disk. TCDMS can validate compressed data. For example on insert requests, TCDMS can check that the characters supplied are valid for the compression type for the data element whose value is being inserted. TCDMS handles all conversion for data which is stored in a compressed format.

When you define the compression type for a data element you also specify an external length. For the numeric compression types you must also specify an internal length. The internal length values depend on the compression type you choose and the alignment requirements for that type.

For certain compression types, TCDMS computes the internal length. For example, suppose you want to define a Name data element that can be a maximum of 42 EBCDIC characters. The external length is 42 bytes. If you use the 6-bit alphanumeric compression type (code 6), the Name data element occupies 32 bytes in the segment. A 42 character name, when compressed, can be stored in 31-1/2 bytes. Because the data compressed to the 6-bit compression type is byte-aligned, TCDMS allocates 32 bytes for the data element.

For numeric compression types, you must specify both the external length and the internal length. For example, suppose you want to define a numeric data element which contains a 5-digit unsigned zoned number. The external length is 5 bytes. The internal length

TABLE I DATA COMPRESSION TYPES AND CODES

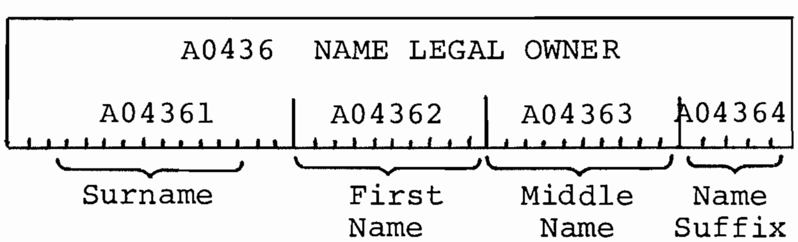
<table>
<tr><th>Code</th><th>Compression Type
Internal Representation</th><th>External Representation</th><th>Maximum External
Length (bytes)</th><th>Internal Length</th><th>Internal
Alignment</th><th>Valid Data Types</th></tr>
<tr><td>8</td><td>not compressed</td><td>Any code</td><td>32,767</td><td>same as external</td><td>byte</td><td>Any 8 bit code, e.g. EBCDIC,
Packed numbers, floating
point numbers</td></tr>
<tr><td>6</td><td>6-bit alphanumeric</td><td>Alphanumeric</td><td>32,767</td><td>3/4 external length</td><td>byte</td><td>A to Z 0 to 9 blank X'00'
,()¢+|&!$*.;-/<%_>?:'"=¬</td></tr>
<tr><td>5</td><td>5-bit alphabetic</td><td>Alphabetic (EBCDIC)</td><td>32,767</td><td>5/8 external length</td><td>byte</td><td>A to Z blank X'00'
.,-'</td></tr>
<tr><td>4</td><td>4-bit character numeric</td><td>Numeric (EBCDIC)</td><td>32,767</td><td>1/2 external length</td><td>byte</td><td>0 to 9 blank .,-$%</td></tr>
<tr><td>D</td><td>2-byte date</td><td>mmddyy (EBCDIC)</td><td>6</td><td>2 bytes</td><td>byte</td><td>dates in mmddyy format</td></tr>
<tr><td>DS</td><td>2-byte date</td><td>mm/dd/yy (EBCDIC)</td><td>8</td><td>2 bytes</td><td>byte</td><td>dates in mm/dd/yy format</td></tr>
<tr><td>PZW</td><td>Packed unsigned</td><td>Zoned numeric ≥ 0</td><td>31</td><td>number of bytes required
(16 bytes maximum)</td><td>byte</td><td>0 to 9 blank</td></tr>
<tr><td>PZI</td><td>Packed signed</td><td>Zoned numeric</td><td>32</td><td>number of bytes required
(16 bytes maximum)</td><td>byte</td><td>0 to 9 blank -</td></tr>
<tr><td>BZW</td><td>Binary unsigned</td><td>Zoned numeric ≥ 0</td><td>19</td><td>number of bits required
(8 bytes maximum)</td><td>bit</td><td>0 to 9 blank</td></tr>
<tr><td>BZI</td><td>Binary signed</td><td>Zoned numeric</td><td>20</td><td>number of bits required
(8 bytes maximum)</td><td>bit</td><td>0 to 9 blank</td></tr>
<tr><td>BPW</td><td>Binary unsigned</td><td>Packed decimal numeric ≥ 0</td><td>10</td><td>number of bits required
(8 bytes maximum)</td><td>bit</td><td>Packed decimal values ≥ 0</td></tr>
<tr><td>BPI</td><td>Binary signed</td><td>Packed decimal numeric</td><td>10</td><td>number of bits required
(8 bytes maximum)</td><td>bit</td><td>Packed decimal values</td></tr>
<tr><td>BW</td><td>Binary unsigned</td><td>Binary ≥ 0</td><td>8</td><td>same as external</td><td>bit</td><td>Binary halfwords, fullwords
or doublewords with values ≥ 0</td></tr>
<tr><td>BI</td><td>Binary signed</td><td>Binary</td><td>8</td><td>same as external</td><td>bit</td><td>Binary halfwords, fullwords
or doublewords</td></tr>
</table>

is 17 bits - the number of bits which can represent the largest non-negative value that can be expressed in 5 zoned digits.

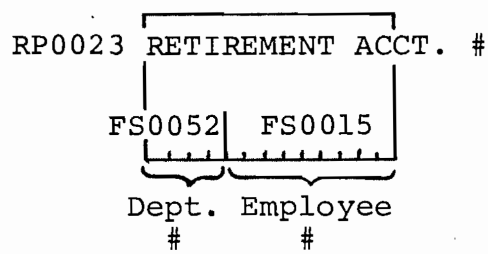
Another descriptive attribute you can specify for the Data Dictionary entry is the offset to the data element within its containing segment. It defines the position of the data element in the segment. For byte-aligned data elements the offset is specified by the number of bytes from the beginning of the segment. For bit-aligned data elements, the offset is expressed by both the number of bytes from the beginning of the segment to the byte in which the data element begins and the number of bits within that byte to the beginning of the data element. Remember that the offsets are computed from the internal (compressed) length.

You can use the offset parameter to define several data elements which reference the same or overlapping fields in a segment. This feature allows you great flexibility in the way you use segments and data elements. There are two main ways to use this feature.

(1) You can define one data element to include fields which are also defined by several other data elements. Example 1: You create a segment which contains a 40-byte name data element A0436. You can also define a 15-byte surname data element, a 10-byte first name data element, a 10-byte middle name data element, and a 5-byte name suffix data element to reference the same 40-byte segment.



Example 2: Your installation uses a combination of department number and employee number as the employee retirement account number. You can define one segment to contain the account number data element RP0023 to reference the 13-byte field in its entirety, and a 4-byte department number data element and 9-byte employee number data element (FS0052 and FS0015) to reference these fields separately.



(2) You can define several data element names with different compression types for one field in the segment. When you do this the internal (stored) form for the several data elements MUST BE THE SAME. The external format can vary. For example, a numeric value is stored in binary in an 8-bit field within a segment. Application programs can access this through three data elements with different external formats and external lengths. The table below shows three data elements which could reference this field.

Element Name	Compression Type	External Format	Internal Format
DR0728	BI	Signed Binary	Signed Binary
DR0942	BZI	Zoned Numeric	Signed Binary
FA6843	BPI	Packed Decimal	Signed Binary

You can define the access allowed for a data element. If no access specifications are included in a Data Dictionary entry for a data element, the data element can be accessed only for retrieval. If the data element is to be updated (change, insert, or delete) you must indicate this when you define the data element. In addition, you can indicate whether the segment which contains the data element is sorted by the value of the data element. Note that data elements which establish the collating position for sorted segments cannot be changed. They must be deleted and re-inserted with a new value. For these data elements you should define the access as insert and delete only.

In addition to the validation which TCDMS performs for compressed data, you can specify two data checks for the data element values. You can specify that the first digit of the compressed form always be 1, or you can specify that it must always be 0. The other data validation which you can request is that compressed form of the data element never be binary 0.

TCDMS provides a function which enables you to create a data element description for the Data Dictionary. The DDE macro instruction creates a Data Dictionary entry for one data element. Section 2.4 contains a description of the DDE macro instruction, and an explanation of the operands.

2.2 Defining the Segments

You must define each segment within the data base. You identify the segment and its hierarchical positon within the data base. The descriptions of the hierarchical relationships among the segments define the structure of the data base. When you define a segment you create a Segment Dictionary entry which contains descriptive information about the segment.

Each segment has a segment descriptor which you create when you define the segment. The segment descriptor contains a four-digit file number, a two-digit number to indicate the hierarchical level that the segment occupies, and a two-digit segment identification. The data base administrator assigns file numbers and segment IDs.

Each file within the data base has a four-digit hexadecimal file number. For the files in the sample data base (Diagram 4 on page 15) these file numbers are 0023 for the Public Utility Property file, 0011 for the Property Description file, 0012 for the Former Account Number file, 0013 for the Name file, and 0014 for the Address file.

The two-digit hexadecimal level number can be from 00 to FF. The root segment of a file is always on level 00. During the SYSGEN process, each installation sets the maximum number of hierarchical levels for their data base. The sample data base depicted here has 5 hierarchical levels, including the root segment. Thus the maximum level number for any segment on this data base is 04. This level contains the exemption segment and the voucher segment.

The segment IDs identify the segments within one hierarchical level on the data base. These two-digit hexadecimal numbers are assigned by the data base administrator. Segments which are pointers have segment IDs from 01 to 7F. There can be 127 types of pointers in a file. The segment IDs 80 to FF are used for non-pointer segments. There can be up to 128 segments on each level in the file. In the sample file 23, the maximum number of segments on one level is 6, on level 02. Remember that you are concerned only with the two-dimensional view of the data base. There can be many occurrences of any one segment in the three-dimensional view; but there is always a maximum of 128 segments per level.

For root segments, the information you supply in the segment descriptor is sufficient to uniquely describe the segment to TCDMS. When you describe the segments at lower hierarchical levels within a file you must include information which defines how the segments relate to the ones on the next hierarchical level. For each segment, you must specify the segment ID of its upward-related segment. For example, when you define the appeal segment which is on level 03, you specify that its upward-related segment is the year segment.

When you define a pointer segment, there is one additional relationship you must specify. You include the four-digit hexadecimal file number of the file to which the pointer segment is linked. Pointers in one file always identify root segments in another file. Thus the file number is sufficient to create the relationship between the two files. In the example data base, the first five segments illustrated on level 02 are pointers. When these segments are defined, the files to which they "point" are specified.

The TCDMS function which allows you to define the segments for your data base is the DSEG macro instruction. This function creates a Segment Dictionary entry for one segment. Section 2.4 contains a description of the DSEG macro instruction and an explanation of the operands.

2.3 Redefining the Segments Which are Referenced by Pointers

After you have defined the data elements and the segments, and established the hierarchical structure within each file, you must complete the data base definition process by creating the inter-file pointer relationships. The inter-file linkages in the

TCDMS data base are composed of corresponding pointers and back pointers.

The inter-file linkages are established in two steps. When a pointer segment is defined, the linkages are placed in the Segment Dictionary entry for the pointer segment. These linkages identify the file the segment references. Section 2.2 describes how these pointer segments are defined. Linkages (back pointers) must also be defined in the Segment Dictionary entries for segments which are referenced by pointers. A back pointer in one segment identifies the particular segment in another file which references the first segment. This section describes how the back pointers are created to complete data base definition.

You must identify each segment which is referenced by pointers from other files. These segments are always root segments in multiple-chain files. Diagram 5 shows a portion of file 23, a root chain file, and all the multiple-chain files associated with it in the sample data base. Notice that the root segments in files 11, 12, 13, and 14 are referenced by pointers from file 23. The root segment for file 13, the Name file, is referenced by pointers in two segments in the root chain file.

Segment redefinition links a "pointed to" root segment with the appropriate pointer segment in another file. When you redefine a segment you specify the segment descriptor of the segment which references it. For example, the root segment for the Property Description file is pointed to by the first segment on level 02 in file 23. Thus when you redefine the root segment for file 11, you specify the segment descriptor of the pointer. This includes the four-digit hexadecimal file number (0023), the two-digit hexadecimal level number (02), and the two-digit hexadecimal segment ID (01).

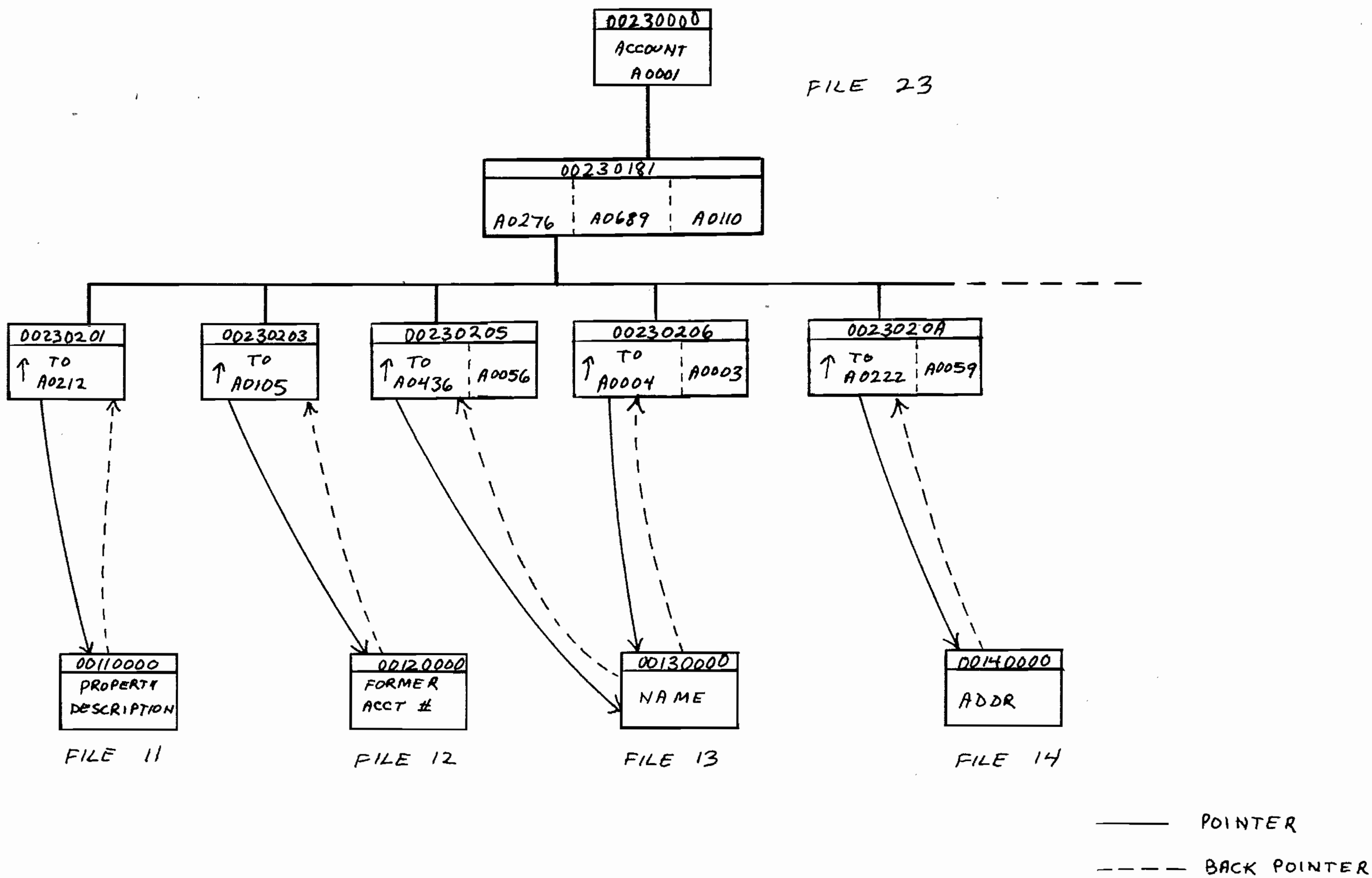

00230000
ACCOUNT
A0001
FILE 23
00230181
A0276
A0689
A0110
00230201
TO
A0212
00230203
TO
A0105
00230205
TO
A0436
A0056
00230206
TO
A0004
A0003
0023020A
TO
A0222
A0059
00110000
PROPERTY
DESCRIPTION
00120000
FORMER
ACCT #
00130000
NAME
00140000
ADDR
FILE 11
FILE 12
FILE 13
FILE 14
POINTER
BACK POINTER

Diagram 5

When a segment is referenced from pointers in several other files, or from several pointer segments in one file, you must include segment redefinitions which describe each linkage. Notice that the root segment in the Name file is "pointed to" by both the Owner Name segment and the Agent Name segment. You should redefine this segment twice. Once to create the linkage to the Owner Name segment, and once again for the Agent Name.

The TCDMS function which redefines segments is the RSEG macro instruction. RSEG defines the necessary linkages in the Segment Dictionary entry for a segment which is referenced from another file. Section 2.4 describes the RSEG macro instruction and its operands.

2.4 How to Use the DSEG, DDE, and RSEG Macro Instructions to Define a Data Base

To define the data base you create a module for assembly and execution. This module contains the TCDMS macro instructions which create the Data Dictionary and Segment Dictionary entries. There are five macro instructions which make up this assembly.

- DBSTRT - creates the TCDMS module which sorts the data element descriptions and segment descriptions into the Data Dictionary and Segment Dictionary.

- DSEG - defines a segment and its hierarchical location within the data base.

- DDE - defines a data element and its descriptive attributes.

RSEG - redefines a segment referenced by a pointer segment in another file.

DBEND - terminates the Data Dictionary and Segment Dictionary generation.

The assembly you create consists of a DBSTRT macro instruction, a series of segment layouts (composed of DSEG, DDE and RSEG macro instructions), and a DBEND macro instruction. This section describes how to code each of these instructions, and explains the operands available. It includes a data base generation assembly for the example data base depicted in Diagram 4 on page 15.

2.4.1 The DBSTRT Macro Instruction

The DBSTRT macro instruction creates the TCDMS module which generates and loads the Data and Segment Dictionaries. These dictionaries are TCDMS system files. The dictionary entries which are loaded are created by DSEG, RSEG, and DDE macro instructions in the same assembly. The module also produces an alphabetical listing of all the data elements defined for the data base. The DBSTRT macro instruction is the first instruction in your data base definition module. There are no operands available with DBSTRT.

Format for Coding the DBSTRT Macro Instruction

[symbol]	DBSTRT	

Return Codes

TCDMS places a completion status code for the Data and Segment Dictionary load into register 15 when the load completes.

Code	Explanation
0	NORMAL LOAD - The Data and Segment Dictionaries have been loaded.
4	LOAD ERROR OCCURRED - An ISAM load error has occurred. The Data and Segment Dictionaries have not been loaded. An ISAM error message is produced which indicates the nature of the error. Some possible errors are: ·duplicate key - you loaded more than one DDE with the same name or more than one DSEG with the same segment ID. ·disk I/O error.
16	MNOTE ISSUED BY DSEG, RSEG, AND/OR DDE MACRO. LOAD PROGRAM TERMINATED - One or more of the DSEG, RSEG, or DDE macro instructions could not be assembled. Check the assembly listing for MNOTE error messages. The Data and Segment Dictionaries have not been loaded.

2.4.2 Segment Layouts

Your data base definition module for assembly is composed chiefly of segment layouts. Each segment layout consists of a segment definition and, for segments referenced by pointers from another file, a segment redefinition. All the segment layouts for one file must be grouped together in the assembly, but they can occur in any order within that group.

A segment definition is composed of one DSEG macro instruction followed by one or more DDE macro instructions which identify the data elements in the segment. In general, you should arrange these DDE instructions in the same order as the order that the data elements occur in the segment.

A segment redefinition is composed of one RSEG macro instruction followed by one DDE macro instruction for each pointer to the segment being redefined. Segment redefinitions occur in the segment layouts for a multiple-chain file. A segment redefinition does not change the segment descriptor or the upward-related segment for the segment being redefined. It describes a different way the segment can be accessed. A segment redefinition allows the use of a data element name which can be accessed only through a pointer to the file which contains the referenced data. TCDMS allows up to 5 intermediate linkages for data referenced by pointer.

Both the Owner Name and Agent Name segments illustrated in Diagram 5 on page 25, are pointers. The segment layout for the root segment of the Name file should contain a segment definition which identifies the root segment, and two segment redefinitions - one for the pointer from the Owner Name segment and one for the pointer from the Agent Name segment. Notice that this allows the segment to be accessed via two data element names - Owner Name and Agent Name.

2.4.2.1 The DSEG Macro Instruction

The DSEG macro instruction generates a Segment Dictionary entry. This entry defines the segment and its hierarchical position within the file. The segment defined by the DSEG macro instruction contains all the data elements defined by DDE macro instructions which follow the DSEG until the next DSEG macro instruction or the DBEND macro instruction in the assembly.

There are three operands available with the DSEG macro instruction. These define the segment and its hierarchical position. The first operand, the segment descriptor, is specified as an eight-character hexadecimal value. It has the format ffffllss. This operand is required for all segment definitions. The four-digit hex file number (ffff) is specified first. These file numbers are assigned by the data base administrator at each installation. The file number for the Public Utility Property file is 0023. The segment descriptor for each segment in this file begins with 0023.

The next field in the segment descriptor contains the hex level number (ll). The root segment of a file is always level 00. Each hierarchical level below that is assigned a number from 01 to FF. The data base administrator sets the maximum number of hierarchical levels at an installation. In the sample data base there are five hierarchical levels, numbered 00 to 04.

The last two digits in the segment descriptor contain the segment ID (ss). These two-digit hex segment IDs are assigned by the data base administrator. They identify the segments within one hierarchical level. Segments which are pointers have segment IDs from 00 to 7F. There is a maximum of 127 pointer segments in each file (on <u>all</u> levels). Thus within a file, a segment ID for a pointer segment is unique. Segment IDs for the non-pointer segments are from 80 to FF. There can be up to 128 segments on <u>each level</u> of a TCDMS file, and the segment IDs are unique within that level.

The next operand identifies the upward-related segment. This is the segment, on the next higher level, on which the segment being defined is dependent. For all segments except root segments, you must include this operand to define the upward-related segment. The operand is specified as the two-digit segment ID, ss, of the upward-related segment. For example, in Diagram 5 the upward-related segment for all the segments on level 02 is the single segment on level 01 which contains the three data elements A0276, A0689, and A0110. The segment ID for this segment on level 01 is 81. Thus the value of the upward-related segment operand for the segments on level 02, all of which are dependent on segment 81, is 81.

You code the third operand for the DSEG macro instruction only when you define a pointer segment. This operand designates the "pointed to" file. You specify this operand by including the four-digit file number (ffff) of the file referenced by this pointer segment. Pointers in one file always reference the root segments of the "pointed to" file. Thus the four-digit file number is sufficient to establish the linkage to the "pointed to" file. (The linkage is completed by information in the segment redefinition contained in the segment layouts for the "pointed to" file. This is described in the section The RSEG Macro Instruction on page 49.

Each DSEG macro instruction must be followed by the DDE macro instructions which define the data elements contained in that segment.

Format for Coding the DSEG Macro Instruction

DSEG	seg descript,up rel seg ID[,ptfile]

Explanation of Operands

Operands are positional and must be coded in the order illustrated.

seg descript — is the eight-digit hexadecimal segment descriptor. It has the format ffffllss.

- -ffff is the four-digit hexadecimal file number of the file which contains this segment.
- -ll is the two-digit hexadecimal level number. It designates the hierarchical level on which this segment occurs. Level numbers are 00 to FF. A root segment is always on level 00.
- -ss is the two-digit hexadecimal segment ID. It identifies the particular segment within a hierarchical level. Segment ID 00 is used for the root to a file. Segment IDs 01-7F identify pointer segments. Segment IDs 80-FF are used for all other segments.

up rel seg ID — is the two-digit hexadecimal segment ID of the upward-related segment. You must code this operand for all except root segments.

ptfile — is the four-digit hexadecimal file number of the file referenced by this segment. It is used only for pointer segments.

Examples:

These examples show how the DSEG macro instruction is used to define the segments marked with large numbers in the sample data base depicted in Diagram 6.

❶ To define the root segment you code:

```
DSEG 00230000
```

This creates a Segment Dictionary entry for the root segment for file 23. A root segment is always on level 00 of a file, and its segment ID is always 00.

❷ To define the segment on the second hierarchical level (01) you code:

```
DSEG 00230181,00
```

Notice that this segment definition includes the up rel seg ID operand with a value of 00. It specifies that the segment being defined is hierarchically dependent on segment ID 00 on the next superior hierarchical level (in this case, the root segment). The first operand, 00230181, specifies that the segment being defined is segment ID 81 on level 01 in file 0023. Notice that this segment is <u>not</u> a pointer segment because the segment ID is between 80 and FF.

❸ To define the segment which references the name of the legal owner for the property you code:

```
DSEG 00230205,81,0013
```

All three operands are present in this pointer segment definition. The segment descriptor specifies that the segment is a

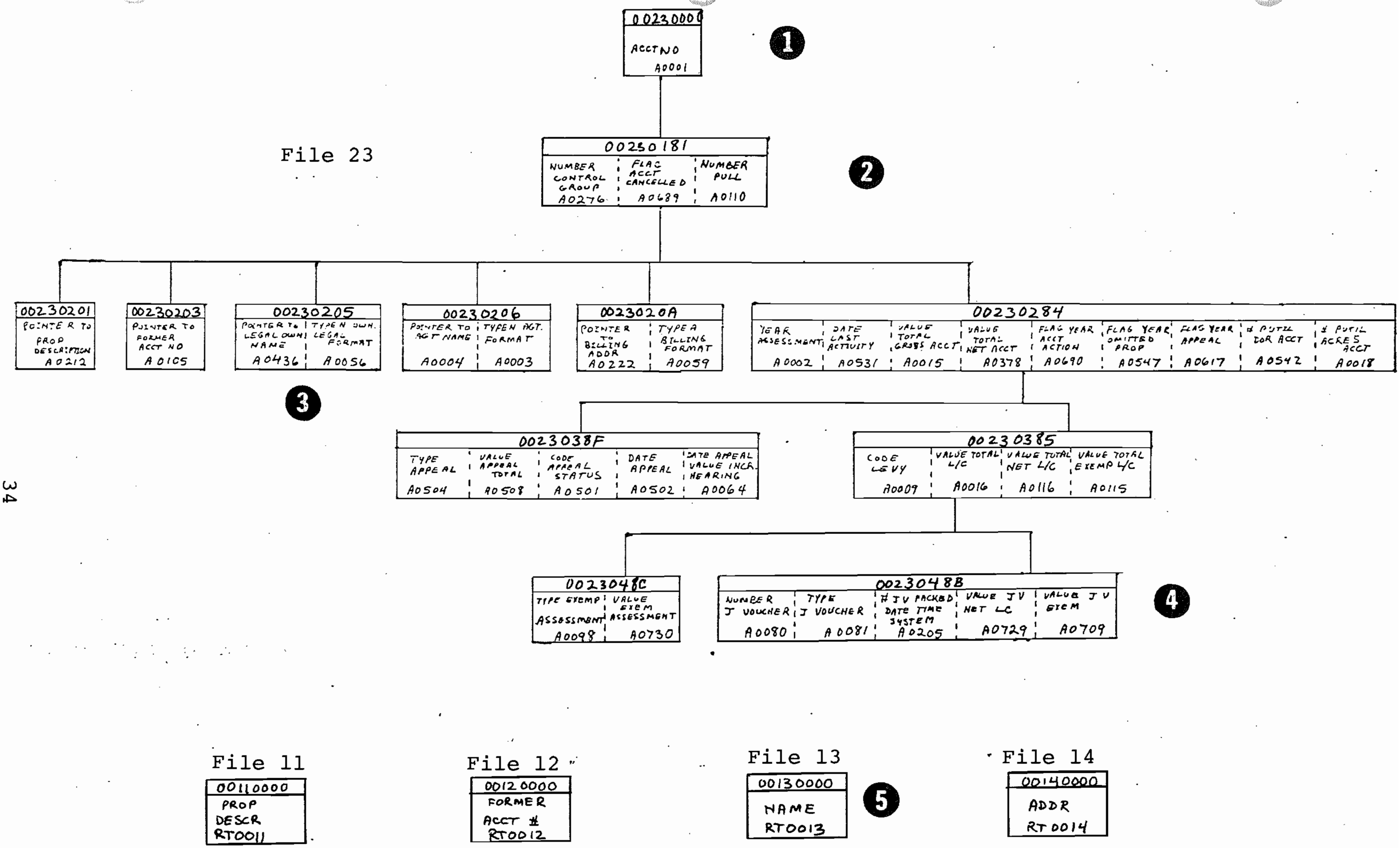

File 23
1
00230000
ACCT NO A0001
2
00250181
NUMBER CONTROL GROUP A0276
FLAG ACCT CANCELLED A0689
NUMBER PULL A0110
00230201
POINTER TO PROP DESCRIPTION A0212
00230203
POINTER TO FORMER ACCT NO A0105
00230205
POINTER TO LEGAL OWN NAME A0436
TYPE N OWN. LEGAL FORMAT A0056
3
00230206
POINTER TO AGT NAME A0004
TYPE N AGT. FORMAT A0003
0023020A
POINTER TO BILLING ADDR A0222
TYPE A BILLING FORMAT A0059
00230284
YEAR ASSESSMENT A0002
DATE LAST ACTIVITY A0531
VALUE TOTAL GROSS ACCT A0015
VALUE TOTAL NET ACCT A0378
FLAG YEAR ACCT ACTION A0690
FLAG YEAR OMITTED PROP A0547
FLAG YEAR APPEAL A0617
PUPIL FOR ACCT A0542
PUPIL ACRES ACCT A0018
0023038F
TYPE APPEAL A0504
VALUE APPEAL TOTAL A0508
CODE APPEAL STATUS A0501
DATE APPEAL A0502
DATE APPEAL VALUE INCR. HEARING A0064
00230385
CODE LEVY A0009
VALUE TOTAL L/C A0016
VALUE TOTAL NET L/C A0116
VALUE TOTAL EXEMP L/C A0115
0023048C
TYPE EXEMP ASSESSMENT A0098
VALUE EXEM ASSESSMENT A0730
0023048B
NUMBER J VOUCHER A0080
TYPE J VOUCHER A0081
JV PACKED DATE TIME SYSTEM A0205
VALUE JV NET LC A0729
VALUE JV EXEM A0709
4
File 11
00110000
PROP DESCR RT0011
File 12
00120000
FORMER ACCT # RT0012
File 13
00130000
NAME RT0013
5
File 14
00140000
ADDR RT0014

Diagram 6

pointer segment (segment ID 05) on level 02 in file 0023. It is hierarchically dependent on segment 81 on level 01. The file to which segment 00230205 points is file 0013, the Name file.

4 To define the journal voucher segment, you code:

```
DSEG 0023048B,85
```

This segment is not a pointer. Its definition contains only the segment descriptor and upward-related segment operands. The segment descriptor, 0023048B, specifies that the segment is segment ID 8B on level 04 in file 0023. It is hierarchically dependent on segment ID 85 on level 03.

5 You define the root segment for the Name file in a manner similar to the way you defined the root segment for the Public Utility Property file. You code:

```
DSEG 00130000
```

This creates a Segment Dictionary entry for the root segment of file 0013, the Name file.

Error Messages for the DSEG Macro Instruction

The DSEG macro instruction creates a Segment Dictionary entry when your module is assembled. It contains no executable coding. Thus any error conditions TCDMS encounters are flagged by the assembler and appear as MNOTEs in your assembly listing. These messages are listed on the following page.

- SEGMENT DESCRIPTOR MUST BE PRESENT

- SEGMENT DESCRIPTOR MUST BE EIGHT HEX CHARACTERS

- UPWARD-RELATED SEGMENT ID MUST BE PRESENT ON NON-ROOT SEGMENTS

- UPWARD-RELATED SEGMENT ID MUST BE 2 HEX CHARACTERS

- UPWARD-RELATED SEGMENT ID NOT ALLOWED ON ROOT SEGMENTS

- ROOT CANNOT BE A POINTER

- NO DDE SINCE PREVIOUS DSEG OR RSEG

- POINTED TO FILE MUST BE FOUR HEX CHARACTERS

2.4.2.2 The DDE Macro Instruction

Each DSEG macro instruction you code should be followed by a series of DDE macro instructions to define the data elements within the segment. Each DDE creates a Data Dictionary entry which defines one data element for the data base.

You designate the name for the data element by coding this name as the label on your DDE macro instruction. Data element names are one to six alphanumeric characters and they must begin with an alphabetic character (Section 2.1 discusses a data element name convention.)

There are six operands available with the DDE macro instruction. These define the compression type, the position of the data element within the segment, the allowed forms of access, sort information, and data validation information.

The compression operand defines the compression type for the data element and the external length. There are thirteen types of data compression available. Table 1, Data Compression Types and Codes on page 18, lists these compression types. For the numeric compression types (PZW, PZI, BZW, BZI, BPW, BPI, BW, and BI) you must also specify the internal length when you code the compression operand. TCDMS computes the internal length for the non-numeric compression types.

Suppose you want to define a Name data element that can be a maximum of 38 EBCDIC characters. The external length is 38 bytes. Alphabetic character data can have compression type 8 (no compression), 6 (6-bit alphanumeric), or 5 (5-bit alphabetic). TCDMS computes the internal length for the compression type you request.

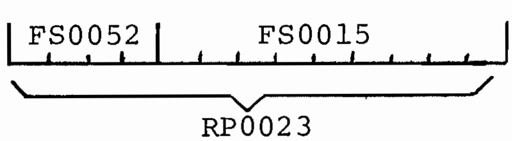
For numeric types you must specify both the external and internal length. For the PZW and PZI compression types the stored values are byte-aligned and the internal length is expressed in bytes. For the BZW, BZI, BPW, BPI, BW, and BI compression types the stored value is not aligned (or is "bit-aligned") and the internal length is expressed in bits. Suppose you want to define a data element which you use as a 5-digit unsigned zoned number. The data element is stored as an unsigned binary value. The compression code is BZW. For external length you specify 5 bytes; the internal length is 17 bits. This is the maximum number of bits needed to represent the largest value which can be expressed in 5 zoned digits.

The offset operand allows you to specify the position of the data element within the segment. This operand is optional. If you arrange the DDE macro instructions in the same order as the "left-to-right" order that the data elements occur in the segment (see Diagram 6 on page 34), you can omit the offset operand from all these DDEs. When you include the offset operand, remember that the data elements are placed in the segment in their compressed format. You should calculate the offset values accordingly.

The offset operand has two suboperands. The first, bytes offset, is used for all the data compression types. It defines the number of bytes within the segment to the byte which contains the data element. The second suboperand, bits offset, is used only for the data elements with a binary compression type (BZW, BZI, BPW, BPI, BW, and BI). It defines the number of bits within the byte to the beginning of the data element.

The offset also allows you to define several data elements which reference the same or overlapping fields in the segment. There are two main ways to use this feature. (1) You can define the data element to include fields which are also defined by several other data elements. In this case you use the offset operand to specify the location in the segment at which the field accessed by each data element begins.

Example:



Element	Offset	Length
FS0052	0	4
FS0015	4	9
RP0023	0	13

These three data elements are all contained in one segment. Example 3 on page 45 shows the DDEs which define the data elements in a similar segment. The example also shows how data compression can be used for these data elements. (2) You can define several data element names with different compression types which all reference one field in the segment. You can use the offset operand to specify that each of these data elements begins at the same location in the segment. When you do this you must be certain that the internal (stored) format of the several data element names IS THE SAME. The external format can vary. Example 4 on page 46 shows how you can code these data element definitions.

The access operand specifies whether the data element can be accessed for update (insert, delete, or change). All data elements are automatically available for retrieval. You use the access operand to specify the other types of allowed access.

The sort operand specifies that the data element is used to determine the sort sequence for the segment. Only one data element in a segment can be used to determine the sort. Thus only one DDE macro instruction in each segment layout can include the sort operand.

The remaining two operands are used for data validation. One specifies that the compressed form of the data element must have a specified value (either 0 or 1) for the first digit. The other operand specifies that the value of the data element must not be zero. TCDMS performs the data validation whenever the designated data elements are accessed.

FORMAT FOR CODING THE DDE MACRO INSTRUCTION

```
element name | DDE | (compression)[,(offset)][,A={I|C|D|ID|IC|CD|ICD}]
             |     | [,S=Y][,F={0|1}][,Z=N]
```

Note that you must code the 1 to 6 character data element name as the label on the DDE macro instruction.

Operands

The first two operands are positional and must be coded in the order illustrated.

(compression) is the compression information for the data element. This operand is required. It has several suboperands. The set of suboperands must be enclosed in parentheses. The format of these suboperands is:

(compression code,external length[,internal length])

compression code is the 1 to 3 character TCDMS code for the compression type of the data element.

external length is the maximum length in bytes of the uncompressed form of the data element. This value cannot exceed the maximum for the compression type.

internal length is the maximum length of the compressed form of the data element. For all the compression types stored in binary format (BZW, BZI, BPW, BPI, BW, and BI) the internal length is number of bits. For all the other compression types, internal length is number of bytes. The internal length value cannot exceed the maximum stored length for the compression type.

Table 1 on page 18 lists all the TCDMS compression types and the codes associated with them. It specifies the maximum external and internal lengths, and indicates whether the compressed from of the data element is byte-aligned. When you code the internal length suboperand for a bit-aligned data element you specify this length in number of bits.

(offset) is the offset within the segment to the data element. This operand is optional. If it is omitted, TCDMS places the data element immediately following the one defined by the previous DDE. If the DDE is the first for the segment, TCDMS places the data element first in the segment. There are two suboperands for the offset operand. They must be enclosed in parentheses, even if only one suboperand is present. The format of the offset operand is:

(bytes offset[,bits offset])

bytes offset is the number of bytes within the segment to the byte which contains the data element.

bits offset is the number of bits within the byte to the first bit of the data element. This suboperand must be coded only for data elements with a bit-aligned compression type (BZW, BZI, BPW, BPI, BW, and BI).

A= { I | C | D | ID | IC | CD | ICD } This operand specifies the types of access allowed for this data element. It is optional. If the access operand is omitted the data element is restricted to retrieval only. You can specify any combination of the three update access codes. These can be in any order.

- I specifies that the value for the data element can be inserted (MCALL INSx).
- C specifies that the value for the data element can be changed (MCALL CHG).
- D specifies that the value for the data element can be deleted (MCALL DEL).

S=Y specifies that the segment which contains this data element is sorted on the compressed value of this data element. This operand is optional. If it is omitted, the data element does not determine a sort sequence for the segment. Note that you must not specify A=C for data elements which determine sort sequence.

F= { 1 | 0 } This operand specifies that the first digit of the compressed form of the data element must have the specified value (either 0 or 1). When you include this operand, TCDMS performs data validation on the first digit whenever the data element is accessed. This operand is optional. If you omit it, TCDMS does not check the first digit of the data element value.

Z=N This operand specifies that the compressed form of the data element value can never be all zeros. This operand is optional. When you include Z=N, TCDMS checks the data element for zero values whenever the data element is accessed. If you omit this operand TCDMS does not check the data element.

Examples

Example 1: To define the data element contained in the root segment for file 23, you code:

```
A0001 DDE (8,6),S=Y,A=ID
```

This defines the data element A0001, which contains the account number. The data element name A0001 appears as the label for the DDE macro instruction. The compression information, (8,6), specifies that the data is stored uncompressed (code 8) and that its maximum external length is 6 bytes. There is only one data element in this segment. The offset operand has been omitted. The S=Y operand has been included. This means that this root segment is sorted by the value of this data element. These segments are stored in a sequence based on the value of the account number. In general, root segments in TCDMS files are sorted. The access operand A=ID specifies that this data element can be accessed for retrieval, insert, or delete. Note that because the data element determines the sort sequence, access to change it is prohibited. No data validation has been requested for this data element.

When you assemble your data base definition module, the internal length of each data element is computed and printed below the DDE which defines the element. Thus this DDE in the assembly listing of your module will appear as:

```
A0001  DDE  (8,6),S=Y,A=ID
            *,48 BITS LONG FROM BYTE 0, BIT 0 TO BYTE 5, BIT 7
```

Example 2: To define all the data elements in the year segment 84 on level 2 in file 23 (segment number 00230284) you code:

```
A0002  DDE  (8,2),A=ICD           YEAR ASSESSMENT
A0531  DDE  (8,8),A=C             DATE LAST ACTIVITY
A0015  DDE  (8,5),A=C             TOTAL GROSS ACCT VALUE
A0378  DDE  (8,5),A=C             TOTAL NET ACCT VALUE
A0690  DDE  (8,1),A=C             FLAG YEAR ACCT ACTION
A0547  DDE  (8,1),A=C             FLAG YEAR OMITTED PROP
A0617  DDE  (8,1),A=C             FLAG YEAR APPEAL
A0542  DDE  (8,3),A=ICD           #PUTIL DOR ACCT
A0018  DDE  (8,4),A=C             #PUTIL ACRES ACCT
```

These DDE instructions are coded in the same order as the data elements occur in the segment (see Diagram 6 on page 34), so the offset operand is omitted. When you assemble your module, the internal length for each data element is computed and printed below each DDE. These values provide you with a map of the year segment. This segment layout will appear in your listing as:

```
A002   DDE  (8,2),A=ICD                YEAR ASSESSMENT
            *,16 BITS LONG FROM BYTE 0, BIT 0 TO BYTE 1, BIT 7
A0531  DDE  (8,8),A=C                  DATE LAST ACTIVITY
            *,64 BITS LONG FROM BYTE 2, BIT 0 TO BYTE 9, BIT 7
A0015  DDE  (8,5),A=C                  TOTAL GROSS ACCT VALUE
            *,40 BITS LONG FROM BYTE 10, BIT 0 TO BYTE 14, BIT 7
A0378  DDE  (8,5),A=C                  TOTAL NET ACCT VALUE
            *,40 BITS LONG FROM BYTE 15, BIT 0 TO BYTE 19, BIT 7
A0690  DDE  (8,1),A=C                  FLAG YEAR ACCT ACTION
            *,8 BITS LONG FROM BYTE 20, BIT 0 TO BYTE 20, BIT 7
A0547  DDE  (8,1),A=C                  FLAG YEAR OMITTED PROP
            *,8 BITS LONG FROM BYTE 21, BIT 0 TO BYTE 21, BIT 7
A0617  DDE  (8,1),A=C                  FLAG YEAR APPEAL
            *,8 BITS LONG FROM BYTE 22, BIT 0 TO BYTE 22, BIT 7
A0542  DDE  (8,3),A=ICD                #PUTIL DOR ACCT
            *,24 BITS LONG FROM BYTE 23, BIT 0 TO BYTE 25, BIT 7
A0018  DDE  (8,4),A=C                  #PUTIL ACRES ACCT
            *,32 BITS LONG FROM BYTE 26, BIT 0 TO BYTE 29, BIT 7
```

Example 3: You want to define a segment which contains a 13-digit retirement account number which is composed of a 4-digit department number, and a 9-digit employee number. You can use compression type 4 to reduce storage space in the file. You use the offset operand to "overlay" the three data element names in the segment. The series of DDE macro instructions you code is:

```
RP0023  DDE  (4,13),A=ICD
FS0052  DDE  (4,4),(0),A=ICD
FS0015  DDE  (4,9),(2),A=ICD
```

When your module assembles, the internal length for the data elements is computed. Your assembly listing appears as:

```
RP0023  DDE  (4,13),A=ICD
             *,56 BITS LONG FROM BYTE 0, BIT 0 TO BYTE 6, BIT 7
FS0052  DDE  (4,4),(0),A=ICD
             *,16 BITS LONG FROM BYTE 0, BIT 0 TO BYTE 1, BIT 7
FS0015  DDE  (4,9),(2),A=ICD
             *,40 BITS LONG FROM BYTE 2, BIT 0 TO BYTE 6, BIT 7
```

Notice that the internal length has been used to produce the listed segment layouts. A 13-byte external length can be stored in 7 bytes when it is compressed using compression type 4. (The value can in fact be stored in 6-1/2 bytes, but compression type 4 is byte-aligned, thus the data element is allocated 7 bytes in the segment.)

The second and third DDEs in the segment layout use the offset operand to specify the location at which the defined data elements begin. The DDE for data element FS0052 begins at 0, the beginning of the segment. Its external length is 4 and since it is stored using compression type 4, the internal length is 2. Although you do not code the internal length for data elements which use compression code 4, you must calculate it for this example because you need it to compute the offset for the next data element, FS0015. The offset value for the DDE which defines FS0015 is 2.

Example 4: You want to define an area within a segment that can be accessed from several different applications in several different formats. The value is stored in binary. One application uses the value in its binary format, another uses it in a zoned numeric format and a third uses it in packed decimal format. You define three data elements which reference this field by coding:

```
DR0728  DDE  (BI,8,8),(1,0),A=IC,Z=N
DR0942  DDE  (BZI,4,8),(1,0),A=ICD,Z=N
FA6843  DDE  (BPI,2,8),(1,0),Z=N
```

The first operand specifies the compression type, the external length in bytes, and the internal length in bits (because the compression type is "bit-aligned"). Notice that all these data elements have the same internal length. Because the compression types are different, the external lengths vary. The location of the data elements within the segment is expressed by the second operand, the offset. These data elements are all located one byte from the beginning of the segment. They all reference the same area of the segment. Notice that because these data elements have a binary internal format, the offset (1,0) is specified by both the number of bytes to the byte which contains the data element and the number of bits within that byte to the beginning of the data element.

The A operand specifies the allowed access. When the field is accessed with the data element name DR0728, it can be retrieved, inserted, or changed. When this field is accessed by data element DR0942, any type of access (retrieval, insert, delete, or change) is permitted. For access by data element name FA6843, the value can be retrieved only. No other access is allowed for that data element.

The last operand in all three DDE macro instructions requests that TCDMS check that the value of the field is not zero.

Error Messages for the DDE Macro Instruction

The DDE macro instruction creates a Data Dictionary entry when your module is assembled. It contains no executable coding. Thus any error conditions TCDMS encounters are flagged by the assembler and appear as MNOTEs in your assembly listing. These messages are listed below:

- NAME MUST BE PRESENT

- name IS TOO LONG - SIX CHARACTERS MAXIMUM
  where name is the data element name you supplied.

- COMPRESSION INFORMATION REQUIRED

- COMP TYPE, EX LEN REQUIRED FOR xxx COMP
  where xxx is the compression code you supplied.

- EXTERNAL LENGTH MUST BE NUMERIC

- INVALID COMPRESSION TYPE

- EXTERNAL LENGTH MUST NOT BE GREATER THAN nn

  where nn is the maximum external length for the compression type you supplied.

- EXTERNAL LENGTH MUST BE EQUAL TO nn

  where nn is either 8 or 6 depending on the date compression (DS or D) you specified.

- COMP TYPE, EX LEN, INT LEN REQUIRED FOR xxx COMP

  where xxx is the compression code you supplied.

- INTERNAL LENGTH MUST BE NUMERIC

- INTERNAL LENGTH MUST NOT BE GREATER THAN mm

  where mm is the maximum internal length for the compression type you supplied.

- EXTERNAL LENGTH MUST BE 1, 2, 4, OR 8

- OFFSET SUBOPERAND(S) MUST BE NUMERIC

- ONLY BYTE OFFSET ALLOWED FOR xxx COMP

  where xxx is the compression code you supplied.

- BYTE AND BIT OFFSET REQUIRED FOR xxx COMP

  where xxx is the compression code you supplied.

- BIT OFFSET MUST BE NUMERIC AND LESS THAN 8

- INTERNAL LENGTH PLUS BIT OFFSET MUST NOT BE GREATER THAN 64

- NO MORE THAN n SUBOPS ALLOWED FOR xxx COMP

  where xxx is the compression code you supplied and n is the maximum number of suboperands (2 or 3) allowed for that compression type.

- DSEG MUST PRECEDE DDE

- ONLY ONE DDE ALLOWED PER ROOT SEGMENT

- INVALID ACCESS TYPE OPERAND

- OFFSET MUST BE ZERO FOR ROOT SEGMENT

2.4.2.3 The RSEG Macro Instruction

Each segment which is referenced through a pointer from another file must be identified and connected with the pointer segment.

Segments which are referenced by pointers from other files are always root segments in multiple-chain files (File 0011, 0012, 0013 and 0014 in the sample data base depicted in Diagram 6 on page 34). Segment redefinition links a "pointed to" root segment with the appropriate pointer in another file. You use the RSEG macro instruction to do this. The RSEG macro instruction redefines the access to data elements within the segment. It specifies the segment which points to the root of the file which contains the data element.

For example, the root segment for the Property Description file is pointed to by the first segment on level 02 in file 23. The segment descriptor for this segment is 00230201. When a segment is referenced from pointers in several files, or from several pointers in one file, you must include segment redefinitions which describe <u>each</u> linkage.

The RSEG macro instruction has one operand. There can be up to 5 suboperands. Each of these suboperands contains a segment descriptor for the "pointing segment". For direct linkages, such as are illustrated in the sample data base, only one suboperand is used in each RSEG macro instruction. The value of the operand for the RSEG macro instruction for the root segment of File 12, the Former Account Number file, is 00230203, the segment descriptor for the segment in File 23 which is identified as the "pointer to former account number".

There can also be indirect linkages between TCDMS files. In this case, each suboperand in the RSEG macro instruction defines the segment which points to the root of the previous file in the linkage. Example 3 on page 52 illustrates this type of linkage.

You code the RSEG macro instruction following the segment layout for the root segment being redefined. The RSEG macro instruction is followed by a DDE macro instruction to identify the data element name for the "pointed to" data. If there are additional redefinitions for the root segment, you code these following the first one.

FORMAT FOR CODING THE RSEG MACRO INSTRUCTION

RSEG	(ptfile[,ptfile,ptfile,ptfile,ptfile])

Operand

There can be up to 5 suboperands for the RSEG macro instruction. For direct linkages you code one. It identifies a segment which points to the root of the file which contains the data element defined by the DDE which follows the RSEG. For indirect linkages, each suboperand specifies the segment which points to the root of the previous file in the linkage.

ptfile is the segment descriptor which identifies the pointing segment. It has the format ffffllss where

- ffff is the four-digit hexadecimal file number
- ll is the two-digit hexadecimal level number
- ss is the two-digit hexadecimal segment ID

Example 1: To redefine the root segment for the Property Description file, file 11, which is pointed to from file 23, you code:

```
       RSEG   (00230201)
 A0212 DDE    (8,18),A=ID              DESCR PROPERTY
```

Notice that the data element name by which the "pointed to" data is referred is defined in the "pointed to" file, not the pointing file.

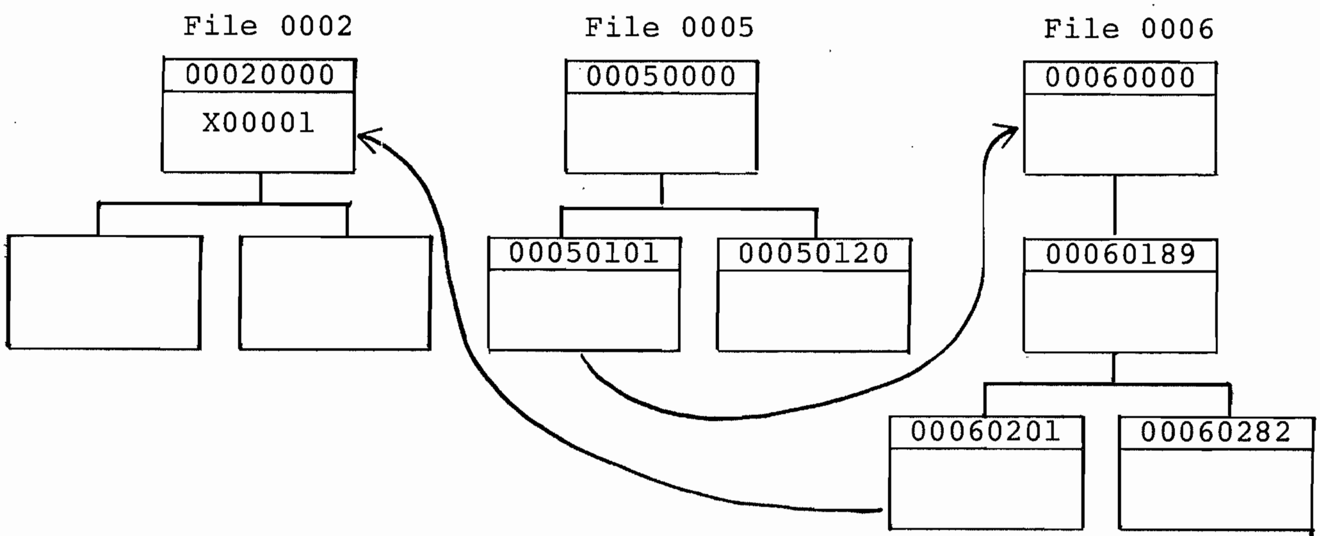
Example 2: The coding below defines, and redefines, the root segment for the Name file, which is pointed to by two segments in the root file, file 23.

```
         DSEG   00130000                  ROOT FILE 13--NAME FILE
RT0013   DDE    (8,40),A=ID               NAME
         RSEG   (00230205)
A0436    DDE    (8,40),A=ID               NAME OWNER LEGAL
         RSEG   (00230206)
A0004    DDE    (8,40),A=ID               NAME AGENT
```

Example 3: To redefine a segment which is connected through intermediate linkages you can code:

```
         RSEG   (00060201,00050101)
X00001   DDE    (6,2),A=ICD
```

This redefinition means that the root of the file which contains data element X00001 is pointed to by segment ID 01 on level 02 in file 0006. The root segment for file 0006 is, in turn, pointed to by segment ID 01 on level 01 in file 0005.



Error Messages for the RSEG Macro Instruction

When your module is assembled, the RSEG macro instruction creates a completed linkage in the Segment Dictionary entry for the segment defined by the preceding DSEG macro instruction. It contains no executable coding. Thus any error condition TCDMS encounters is flagged by the assembler and appears as an MNOTE in your assembly listing. These messages are listed below.

- NO DDE SINCE PREVIOUS DSEG OR RSEG
- POINTING SEGMENT DESCRIPTOR NOT PRESENT
- NO MORE THAN 5 POINTING SEGMENTS ALLOWED
- POINTING SEGMENT DESCRIPTOR MUST BE EIGHT HEX CHARACTERS
- RSEG MUST BE PRECEDED BY A DSEG
- RSEG NOT ALLOWED ON POINTING SEGMENTS

2.4.3 The DBEND Macro Instruction

The DBEND macro instruction terminates a data base definition assembly module. You code this macro instruction after all the segment layouts for all the files in your data base. The DBEND macro instruction sets the counters for the number of data elements and segments you have defined. There are no operands available with the DBEND macro instruction.

FORMAT FOR CODING THE DBEND MACRO INSTRUCTION

DBEND

There are no operands or return codes from DBEND.

2.4.4 Sample Data Base Definition

The sample assembly listing on the following page is the entire data base definition module for the sample data base depicted in Diagram 4 on page 15.

```
STMT     SOURCE STATEMENT

   1               PRINT NOGEN
   2               DBSTRT
 864               DSEG   00230000               ROOT FILE 23--UTILITY
 871   A0001       DDE    (8,6),S=Y,A=ID         NUMBER ACCT IDENTIFICATION
 884                      *,48 BITS LONG FROM BYTE 0, BIT 0 TO BYTE 5, BIT 7
 886               DSEG   00230181,00
 893   A0276       DDE    (8,2),A=C              NUMBER CONTROL GROUP
 906                      *,16 BITS LONG FROM BYTE 0, BIT 0 TO BYTE 1, BIT 7
 908   A0689       DDE    (8,1),A=C              FLAG ACCT CANCELLED
 921                      *,8 BITS LONG FROM BYTE 2, BIT 0 TO BYTE 2, BIT 7
 923   A0110       DDE    (8,3),A=C              NUMBER PULL
 936                      *,24 BITS LONG FROM BYTE 3, BIT 0 TO BYTE 5, BIT 7
 938               DSEG   00230201,81,0011       POINTER TO DESCR PROPERTY
 945               DSEG   00230203,81,0012       POINTER TO NUMBER ACCT FORMER
 952               DSEG   00230205,81,0013       POINTER TO NAME OWNER LEGAL
 959   A0056       DDE    (8,1),A=C              TYPEN OWNER LEGAL FORMAT
 972                      *,8 BITS LONG FROM BYTE 4, BIT 0 TO BYTE 4, BIT 7
 974               DSEG   00230206,81,0013       POINTER TO NAME AGENT
 981   A0003       DDE    (8,1),A=C              TYPEN AGENT FORMAT
 994                      *,8 BITS LONG FROM BYTE 4, BIT 0 TO BYTE 4, BIT 7
 996               DSEG   0023020A,81,0014       POINTER TO ADDR BILLING
1003   A0059       DDE    (8,1),A=C              TYPEA BILLING FORMAT
1016                      *,8 BITS LONG FROM BYTE 4, BIT 0 TO BYTE 4, BIT 7
1018               DSEG   00230284,81            YEAR SEGMENT
1025   A0002       DDE    (8,2),A=ICD            YEAR ASSESSMENT
1038                      *,16 BITS LONG FROM BYTE 0, BIT 0 TO BYTE 1, BIT 7
1040   A0531       DDE    (8,8),A=C              DATE LAST ACTIVITY
1053                      *,64 BITS LONG FROM BYTE 2, BIT 0 TO BYTE 9, BIT 7
1055   A0015       DDE    (8,5),A=C              VALUE TOTAL GROSS ACCT
1068                      *,40 BITS LONG FROM BYTE 10, BIT 0 TO BYTE 14, BIT 7
1070   A0378       DDE    (8,5),A=C              VALUE TOTAL NET ACCT
1083                      *,40 BITS LONG FROM BYTE 15, BIT 0 TO BYTE 19, BIT 7
1085   A0690       DDE    (8,1),A=C              FLAG YEAR ACCT ACTION
1098                      *,8 BITS LONG FROM BYTE 20, BIT 0 TO BYTE 20, BIT 7
1100   A0547       DDE    (8,1),A=C              FLAG YEAR OMITTED PROPERTY
1113                      *,8 BITS LONG FROM BYTE 21, BIT 0 TO BYTE 21, BIT 7
1115   A0617       DDE    (8,1),A=C              FLAG YEAR APPEAL
1128                      *,8 BITS LONG FROM BYTE 22, BIT 0 TO BYTE 22, BIT 7
1130   A0542       DDE    (8,3),A=ICD            NUMBER PUTIL DOR ACCT
```

```
STMT      SOURCE STATEMENT

1143                     *,24 BITS LONG FROM BYTE 23, BIT 0 TO BYTE 25, BIT 7
1145  A0018   DDE    (8,4),A=C            NUMBER PUTIL ACRES ACCT
1158                 *,32 BITS LONG FROM BYTE 26, BIT 0 TO BYTE 29, BIT 7
1161          DSEG   0023038F,84          APPEAL SEGMENT
1168  A0504   DDE    (8,2),A=ICD          TYPE APPEAL
1181                 *,16 BITS LONG FROM BYTE 0, BIT 0 TO BYTE 1, BIT 7
1183  A0508   DDE    (8,5),A=C            VALUE APPEAL TOTAL
1196                 *,40 BITS LONG FROM BYTE 2, BIT 0 TO BYTE 6, BIT 7
1198  A0501   DDE    (8,1),A=C            CODE APPEAL STATUS
1211                 *,8 BITS LONG FROM BYTE 7, BIT 0 TO BYTE 7, BIT 7
1213  A0502   DDE    (8,8),A=C            DATE APPEAL
1226                 *,64 BITS LONG FROM BYTE 8, BIT 0 TO BYTE 15, BIT 7
1228  10064   DDE    (8,8),A=C            DATE APPEAL VALUE INCR HEARG
1241                 *,64 BITS LONG FROM BYTE 16, BIT 0 TO BYTE 23, BIT 7
1243          DSEG   00230385,84          L\C ASSESSMENT SEGMENT
1250  A0009   DDE    (8,3),A=ICD          CODE LEVY
1263                 *,24 BITS LONG FROM BYTE 0, BIT 0 TO BYTE 2, BIT 7
1265  A0016   DDE    (8,5),A=C            VALUE TOTAL L\C
1278                 *,40 BITS LONG FROM BYTE 3, BIT 0 TO BYTE 7, BIT 7
1280  A0116   DDE    (8,5),A=C            VALUE TOTAL NET L\C
1293                 *,40 BITS LONG FROM BYTE 8, BIT 0 TO BYTE 12, BIT 7
1295  A0115   DDE    (8,5),A=C            VALUE TOTAL EXEMP L\C
1308                 *,40 BITS LONG FROM BYTE 13, BIT 0 TO BYTE 17, BIT 7
1310          DSEG   0023048C,85          EXEMPT SEGMENT
1317  A0098   DDE    (8,2),A=ICD          TYPE EXEM ASSESSMENT
1330                 *,16 BITS LONG FROM BYTE 0, BIT 0 TO BYTE 1, BIT 7
1332  A0730   DDE    (8,5),A=C            VALUE EXEM ASSESSMENT
1345                 *,40 BITS LONG FROM BYTE 2, BIT 0 TO BYTE 6, BIT 7
1347          DSEG   0023048B,85          VOUCHER SEGMENT
1354  A0080   DDE    (8,5),A=ICD          NUMBR JVOUCHER
1367                 *,40 BITS LONG FROM BYTE 0, BIT 0 TO BYTE 4, BIT 7
1369  A0081   DDE    (8,2),A=C            TYPE JVOUCHER
1382                 *,16 BITS LONG FROM BYTE 5, BIT 0 TO BYTE 6, BIT 7
1384  A0205   DDE    (8,8),A=C            NUMBR JVPACKD DATE TIME SYSTEM
1397                 *,64 BITS LONG FROM BYTE 7, BIT 0 TO BYTE 14, BIT 7
1399  A0729   DDE    (8,5),A=C            VALUE JV NET LC
```

```
STMT      SOURCE STATEMENT

1412                  *,40 BITS LONG FROM BYTE 15, BIT 0 TO BYTE 19, BIT 7
1414  A0709   DDE     (8,5),A=C            VALUE JV EXEM
1427                  *,40 BITS LONG FROM BYTE 20, BIT 0 TO BYTE 24, BIT 7
1430          DSEG    00110000             ROOT FILE 11--DESCR PROPERTY
1437  RT0011  DDE     (8,18),A=ID          DESCR PROPERTY
1450                  *,144 BITS LONG FROM BYTE 0, BIT 0 TO BYTE 17, BIT 7
1452          RSEG    (00230201)
1453  A0212   DDE     (8,18),A=ID          DESCR PROPERTY
1466                  *,144 BITS LONG FROM BYTE 0, BIT 0 TO BYTE 17, BIT 7
1469          DSEG    00120000             ROOT FILE 12--NUMBER ACCT FRMER
1476  RT0012  DDE     (8,6),A=ID           NUMBER ACCT FORMER
1489                  *,48 BITS LONG FROM BYTE 0, BIT 0 TO BYTE 5, BIT 7
1491          RSEG    (00230203)
1492  A0105   DDE     (8,6),A=ID           NUMBER ACCT FORMER
1505                  *,48 BITS LONG FROM BYTE 0, BIT 0 TO BYTE 5, BIT 7
1508          DSEG    00130000             ROOT FILE 13--NAME FILE
1515  RT0013  DDE     (8,40),A=ID          NAME
1528                  *,320 BITS LONG FROM BYTE 0, BIT 0 TO BYTE 39, BIT 7
1530          RSEG    (00230205)
1531  A0436   DDE     (8,40),A=ID          NAME OWNER LEGAL
1544                  *,320 BITS LONG FROM BYTE 0, BIT 0 TO BYTE 39, BIT 7
1546          RSEG    (00230206)
1547  A0004   DDE     (8,40),A=ID          NAME AGENT
1560                  *,320 BITS LONG FROM BYTE 0, BIT 0 TO BYTE 39, BIT 7
1563          DSEG    00140000             ROOT FILE 14--ADDR FILE
1570  RT0014  DDE     (8,40),A=ID          ADDR
1583                  *,320 BITS LONG FROM BYTE 0, BIT 0 TO BYTE 39, BIT 7
1585          RSEG    (0023020A)
1586  A0222   DDE     (8,40),A=ID          ADDR BILLING
1599                  *,320 BITS LONG FROM BYTE 0, BIT 0 TO BYTE 39, BIT 7
1601          DBEND
1604          END
```

3.0 GENERATING THE DATA BASE CONTROL BLOCK

3.0 Generating the Data Base Control Block

A data base control block (DBCB) defines which data elements within the data base can be accessed by a particular application program. It includes a list of all the data elements which the program can use, and a segment table which TCDMS uses to locate the segments which contain these data elements. The DBCB also contains other TCDMS work areas.

DBCBs are created at each TCDMS installation by the data base administrator in consultation with users and application programmers. These DBCBs are kept in a TCDMS library. Each application program which accesses the data base must have a DBCB associated with it. The data base control block from the library is associated with an application program either as the program is loaded or during execution before any data base access. This DBCB is loaded into an unused portion of the thread which the program is using. In addition to the tables and control blocks which were stored on the library, the DBCB includes and formats the remaining space in the thread into tables and a Segment Work Area (SWA). The SWA is the area that TCDMS uses either when it extracts data elements from a retrieved segment before passing them to the program, or when it groups data elements into segments before inserting them into the data base.

This chapter describes the component tables, work areas, and fields in the DBCB and the procedures for generating a data base control block.

3.1 DBCB Components

A data base control block (DBCB) contains two types of areas: those TCDMS builds when the DBCB is created and those it builds when the DBCB is loaded into a user thread.

The three tables or areas constructed when the DBCB is created are the DMS communications area (DCA), the segment descriptor table (SDT), and the data element area (DEA). These three fields are in the DBCB that is stored in the TCDMS library. When the DBCB is loaded into a user thread, six additional areas are allocated and formatted from the available storage in the thread. These are the back pointer occurrence table (BOT), the scan point element (SPE), the access request (ARQ), the segment processor work area (SPWA), a register save area, and the segment work area.

The DMS communications area (DCA) contains address constants for DMS tables and routines as well as parameters and work areas. It contains the addresses of the other tables and areas in the DBCB. TCDMS uses the information in the DCA when it handles data base access requests for an application program. The DCA is 524 bytes long. TCDMS creates this field when the DBCB is generated.

The segment descriptor table (SDT) maps out the physical structure of the portion of the data base which the application program will use. It contains one segment descriptor table entry for each segment which includes a data element listed in the data element area. In addition there is one entry for each higher segment along the vertical hierarchical path from this segment to the root segment of the file. Thus the SDT contains the information which enables TCDMS to locate any data element which the application program can request.

Each SDTE (segment descriptor table entry) contains the segment descriptor (file number, level number and segment ID), and the file number of any pointed to segment. In addition it contains the offset to the SDTE for the upward-related segment, and the offset to the first data element entry (in the DEA) which is associated with this segment. It contains other linkage and status fields used by TCDMS. Each SDTE is 36 bytes long. TCDMS creates the segment descriptor table entries from information contained in the Data and Segment Dictionaries. It groups them into the SDTE when the DBCB is generated.

The data element area (DEA) contains one data element entry (DEE) for each data element which is specified in the DBCB generation process. Only the data elements listed in the DEA can be accessed by an application program which has been associated with the DBCB.

Each data element entry contains the data element name, the location of the SDTE for the segment which contains this data element, and the location of the data element within the segment. In addition it contains status bits to indicate the data element external format and length and the internal (compressed) format and length. It also contains a work area used by both the scan processor and the data element processor during data access. Each DEE is 20 bytes long. TCDMS creates the DEEs from information in the Data Dictionary when a DBCB is generated. You specify only the data element names for the data elements you want included.

The three tables just described are stored as a DBCB in the system library. When the DBCB is associated with an application program, TCDMS creates and initializes the remaining six tables and areas. They are described briefly here, and estimates of the length of each are given. A programmer should be aware of the size of the DBCB which is associated with his program to be sure that both will fit in the thread and that there is ample storage for the segment work area.

The back pointer occurrence table (BOT) is a table TCDMS uses to identify the back pointer file linkages for data elements at the current data point. The BOT is 12 bytes long, plus 12 bytes for each pointer segment. The scan point element (SPE) is a control block used to identify and describe the current scan point. The SPE is 99 bytes long. The access request (ARQ) is a control block used to pass requests to the access method for physical data access. This control block is 20 bytes long, plus 8 times the maximum segment level in any file referenced by the DBCB. For example, if the "lowest" segment in the hierarchy is level 04, then the ARQ is 20 bytes + (8 bytes/level) x (5 levels) = 60 bytes. The segment processor work area (SPWA) is used by the segment processor. It is 20 bytes long plus 4 times the maximum segment level in any file referenced by the DBCB. Thus for the same DBCB just used as an example for the ARQ length, the SPWA is 40 bytes.

The register save area is used by all the data management system routines which handle an application program request. Control passes through many DMS routines before the request is satisfied. The register save area is 1000 bytes.

TCDMS uses all the remaining space in the user's thread as the segment work area. The SWA is the area that TCDMS uses during application program execution either when it extracts data elements from a segment before passing them to the program, or when it groups data elements into segments before inserting or updating a segment on the data base.

The SWA contains three areas - the data position tables (DPT), the segment headers and data, and the update control blocks. TCDMS creates and modifies these fields while handling the application program data access requests. The data point table is used to keep track of multiple data points within the data base. There can be up to 256 data points used by the application program. TCDMS does not use the data point table if only one data point (data point 0) is being used. The DPT contains one entry for each data point number (other than data point 0) which is used by the associated application program. This entry is a copy of the SDT status at the time when the data point was set. It contains 4 bytes, plus 6 bytes for each segment in the SDT, plus a copy of the BOT and SPE from that time.

The main use of the SWA is to contain the segments being accessed by the application program. Each segment that is kept in the SWA is preceded by a segment header. The size of the segment header depends on the type of segment. Segment headers for root segments are 8 bytes long. For a pointer segment, the segment header is 20 bytes plus 4 bytes for each level from the root segment to the pointer. For example, a segment header for a pointer on level 02 of a file is 32 bytes long (20 bytes + (4 bytes/level) x 3 levels). Segment headers for other segments are 16 bytes plus 4 bytes per level.

When TCDMS retrieves data it keeps only the most recently accessed copy of a segment in the SWA. TCDMS keeps all segments for update (insert, delete, or change) in the SWA until the physical write to the data base is performed. If the SWA fills up with these updated segments, TCDMS attempts to obtain more SWA space by freeing space occupied by retrieved segments. If there are no retrieved segments to release, the application program has no more storage to use to update the data base. TCDMS terminates this application program abnormally. For this reason, it is important that programmers understand the sizes of storage available to them in the SWA.

TCDMS keeps in the SWA only the amount of the segment which contains the data elements specified by the application program. For root segments, it keeps the entire segment (the key). For pointer segments it is the pointer portion (RBF address of the pointed to segment) and the "pointed to" value.

TCDMS creates update control blocks in the SWA only when the application program requests a file update. Each update control block contains an ARQ and a file request table which identifies the files affected by the update. The FRT is 4 bytes plus 2 bytes for each file referenced in the DBCB.

3.2 How to Create a DBCB

You create a DBCB using the TCDMS DBCB generation module DUDBCB. You execute this program as part of a job. The job also should include a link-edit step to prepare the DBCB for the library.

The SYSIN input data set for DUDBCB contains the one- to six-character name of the DBCB, and the names of all the data elements which the application program which uses the DBCB can access. You specify the DBCB name by coding DBCB=name and you separate this from the data element names by a comma. Each data element name is separated from the next by a comma. You must also include DD statements which define the system libraries which contain the Data Dictionary and the Segment Dictionary. DDname DMSDADIC identifies the

Data Dictionary. DDname DMSSEDIC identifies the Segment Dictionary.

The DUDBCB module reads the data element names you supply, and looks up the appropriate entries in the Data Dictionary. From that information it can create the data element entries for the DEA. The information in each Data Dictionary entry also identifies the segment descriptor for the segment which contains the data element. Using this, DUDBCB looks in the Segment Dictionary to find the upward-related segment for this segment. It continues this process until it has identified all the segment descriptors for segments in the path from the segment which contains the data element to the root segment of the file. DUDBCB does this for each data element name input. These segment descriptors form the segment descriptor table (SDT) in the DBCB. DUDBCB also sets up the DCA. It puts the completed DBCB onto the system library, and produces a listing of the DBCB and all the data elements in it.

The sample job control statements below show the DBCB creation step in a DBCB generation job. This job generates a DBCB for the data base defined in chapter 2. The DBCB identifies all the data elements in this sample data base. They are all accessible to any program which is associated with this DBCB.

```
//GEN       EXEC PGM=DUDBCB
//STEPLIB   DD   DSN=USER.TCDMSGO,DISP=SHR    LIBRARY FOR DUDBCB
//SYSIN     DD   *
 DBCB=ADBCB1,A0001,A0276,A0689,A0110,A0056,A0003,A0059
 A0002,A0531,A0015,A0378,A0690,A0547,A0617,A0542,A0018
 A0504,A0508,A0501,A0502,A0064,A0009,A0016,A0116,A0115
 A0098,A0730,A0080,A0081,A0205,A0729,A0709,A0212,A0105
 A0004,A0222,A0436
/*
//DMSDADIC DD   DSN=USER.DMS.ATDBDD,DISP=OLD,DCB=DSORG=IS
//DMSSEDIC DD   DSN=USER.DMS.ATDBSD,DISP=OLD,DCB=DSORG=IS
//SYSPRINT DD   SYSOUT=A
//SYSLIB   DD   DSN=&&DBCB,DISP=(NEW,PASS),DCB=(DSORG=PS,
                BLKSIZE=800,LRECL=80,RECFM=FB),SPACE=(TRK,(1)),
                UNIT=SYSDA
```

The six DDnames define the data sets used for DBCB generation. STEPLIB describes the library which contains the DUDBCB module. The SYSIN data set contains the input to the DUDBCB program, the name of the DBCB and the data element names. DMSDADIC defines the Data Dictionary and DMSSEDIC defines the Segment Dictionary. SYSLIB specifies the output data set for the DBCB which is created. SYSPRINT contains the messages generated by this step.

The next component of your DBCB generation job link-edits the DBCB and puts it on the system library. The sample job control statements below show the link-edit step.

```
//LINKDBCB EXEC UPLKED,NAME=ADBCB1
//SYSLIN   DD   DSN=&&DBCB,DISP=OLD
```

The DUDBCB program produces a list of all the data elements in the DBCB and a map of the DBCB structure. The sample JCL below shows the entire DBCB generation. It is followed by the DBCB map produced by this job. The DBCB which is generated is for the sample A&T data base used throughout this manual.

```
//ATDBCB   JOB    accounting information
//GEN      EXEC PGM=DUDBCB
//STEPLIB  DD   DSN=USER.TCDMSGO,DISP=SHR
//SYSIN    DD   *
DBCB=ADBCB1,A0001,A0276,A0689,A0110,A0056,A0003,A0059
A0002,A0531,A0015,A0378,A0690,A0547,A0617,A0542,A0018
A0504,A0508,A0501,A0502,A0064,A0009,A0016,A0116,A0115
A0098,A0730,A0080,A0081,A0205,A0729,A0709,A0212,A0105
A0004,A0222,A0436
/*
//DMSDADIC DD   DSN=USER.DMS.ATDBDD,DISP=OLD,DCB=DSORG=IS
//DMSSEDIC DD   DSN=USER.DMS.ATDBSD,DISP=OLD,DCB=DSORG=IS
//SYSPRINT DD   SYSOUT=A
//SYSLIB   DD   DSN=&&DBCB,DISP=(NEW,PASS),DCB=(DSORG=PS,
                BLKSIZE=800,LRECL=80,RECFM=FB),SPACE=(TRK,(1)),
                UNIT=SYSDA
//LINKDBCB EXEC UPLKED,NAME=ADBCB1
//SYSLIN   DD   DSN=&&DBCB,DISP=OLD
//
```

TCDMS DATA BASE CONTROL BLOCK GENERATION FOR DBCB ADBCB1
DBCB MAP

LOGICAL DBCB1	UP REL SEG	GEN	SRTD	PTR TO
0023-00-00	**ROOT**			
A0001			X	
0023-01-81	0023-00-00			
A0276				
A0689				
A0110				
0023-02-01	0023-01-81			0011
A0212				
0023-02-03	0023-01-81			0012
A0105				
0023-02-05	0023-01-81			
A0436				
A0056				
0023-02-06	0023-01-81			0013
A0004				
A0003				
0023-02-0A	0023-01-81			0014
A0222				
A0059				
0023-02-84	0023-01-81			
A0002				
A0531				
A0015				
A0378				
A0690				
A0547				
A0617				
A0542				
A0018				
0023-03-85	0023-02-84			
A0009				
A0016				
A0116				
A0115				
0023-04-8B	0023-03-85			
A0080				
A0081				
A0205				
A0729				
A0709				
0023-04-8C	0023-03-85			
A0098				
A0730				
0023-03-8F				
A0504				
A0508				
A0501				
A0502				
A0064				

Interpreting the DBCB Map

The DBCB map shows the logical structure of the data base defined by the DBCB. The field on the left contains the segment descriptors for all the segments in the portion of the data base described by this DBCB. They are arranged in a hierarchical structure which parallels the data base hierarchy. Each segment which is dependent on another segment is listed beneath its upward-related segment and is indented two spaces. This allows you to see the data base hierarchy more clearly. All the segments on one level have the same indentation (for example segments A0023-03-85 and 0023-03-8F). All the data elements which are contained in a segment are listed immediately following the segment descriptor for that segment. The data elements which are the "target" of pointers (for example A0004 and A0003 in files 0013 and 0014) are listed as if they were directly contained in the pointing segment. This is the logical view of the data base. Remember that when you defined the physical view of the data base using the data base definition process described in chapter 2, you listed the DDEs for "pointed to" data elements in the files to which they belonged.

The four columns on the right define the upward-related segment (UP REL SEG column) and specify whether the segment was generated for the DBCB (GEN column), whether the segment is sorted (SRTD column) and if it is a pointer (PTR TO). The UP REL SEG column contains the segment descriptor for the upward-related segment for any segment. If the segment is a root segment, it has no upward-related segment and **ROOT** appears in this column. (See segment 0023-00-00).

The GEN column contains an X for any segments which are required to complete the hierarchical path to the root but which do not contain data elements listed in the input to DUDBCB. Such segments lie on the path between a segment which contains a listed data element and the root segment, but since no data elements in these segments are listed, they cannot be accessed.

The SRTD column contains an X for any data element that determines the sort sequence for its containing segment. For example, data element A0001 in segment 0023-00-00 is marked X in the SRTD column. This means that the many occurrences of segment 0023-00-00 are stored in a collating sequence, arranged by the values of the A0001 data elements within them. The PRT TO column indicates the file number of the file pointed to by any pointer segments.

Cataloging a DBCB

After DUDBCB creates a DBCB and it is link-edited, the DBCB is placed in a TCDMS library. Before the DBCB can be used by an application program the DBCB name must be cataloged. This creates a TCDMS library directory entry for the DBCB. When a DBCB is cataloged the names of all application programs which can use this DBCB are specified. DBCB cataloging is a secured ULIB CAT function available only to a data base administrator. The format of the ULIB command to catalog a DBCB is

```
*ULIB CAT,DBCB=dbcbname,PGMS=(progname,...)
```

where dbcbname	is the one- to six-character name of the DBCB being cataloged.
progname	is the one- to six-character name of a program which can use the DBCB. There can be several program names; they must be separated by commas and the entire list must be enclosed in parentheses.

4.0 GLOSSARY

4.0 GLOSSARY

ACCESS METHOD The set of modules within the Data Management component of TCDMS which locate and access data in a TCDMS data base.

ACCESS REQUEST (ARQ) A control block within a DBCB which is used to pass a request to the access method for physical data base access.

ARQ See access request.

BACK POINTER A linkage contained in a segment in a multiple-chain file which identifies a particular segment (a pointer segment) in another file. TCDMS file linkages are composed of corresponding pointers and back pointers.

BACK POINTER OCCURRENCE TABLE (BOT) A table in a DBCB that TCDMS uses to identify the back pointer file linkage during data base access.

DATA BASE ADMINISTRATOR A person or group responsible for design, creation, and maintenance of the data base at an installation.

DATA BASE CONTROL BLOCK (DBCB) A TCDMS control block which identifies both the data elements accessible to an application program and the logical relationships between those elements. The DBCB also contains the segment work area and other TCDMS work areas.

DATA COMPRESSION A TCDMS method for reducing the direct access storage requirements for data in TCDMS files.

DATA DICTIONARY A TCDMS system file which contains the data element descriptors for all the data elements in the data base.

DATA ELEMENT The unit of data handled by an application programmer who accesses TCDMS files. It is the smallest unit of data in a TCDMS file.

DATA ELEMENT AREA (DEA) A TCDMS control block within a DBCB which identifies the names and location information for the data elements which can be accessed using that DBCB.

DATA ELEMENT DESCRIPTOR (DED) An entry in the Data Dictionary which contains a data element name, the external and stored lengths and the compression type of the data element, the accessibility and security attributes, and in-information relating the data element to a segment. There is one DED for each data element in the data base.

DATA ELEMENT ENTRY (DEE) An entry in the data element area. It identifies one data element name and its location and status information.

DATA ELEMENT NAME A unique name assigned to a data element in the data base.

DATA POINT (1) A collection of segments in the data base which are available for access. A data point includes only one occurrence of each segment type on each hierarchical level in the file. A data point identifies a unique location within the data base.

(2) A number (0 to 255) which specifies a scan point and data point.

DATA POINT TABLE (DPT) A table in the segment work area which TCDMS uses to identify multiple data points during data base access.

DBCB See data base control block.

DBEND The TCDMS macro instruction which terminates a data base definition assembly module.

DBSTRT The TCDMS macro instruction which initiates a data base definition assembly module.

DCA See DMS communications area.

DDE The TCDMS macro instruction which creates a Data Dictionary entry for one data element in the data base.

DED See data element descriptor.

DEA See data element area.

DEE See data element entry.

DMS COMMUNICATIONS AREA A control block in the DBCB which contains address constants for routines and tables TCDMS uses to handle data base access requests.

DPT See data point table.

DSEG The TCDMS macro instruction which creates a Segment Dictionary entry to define a segment and its hierarchical position within the data base.

FAMILY A collection of segments which are hierarchically dependent on one segment, the root segment. All the data in a family pertains to the root segment.

HIERARCHICAL LEVEL An occurrence or set of occurrences of segments which occupy the same relative vertical position in the data base. Root segments always form the first hierarchical level in a file. The set of segments directly dependent on the root segments forms the second hierarchical level. The segments directly dependent on these form the third hierarchical level, etc.

HIERARCHICAL PATH A set of segments which includes one segment on each level between the lowest-level segment requested and the root segment for that file. Also called vertical path.

LOGICAL FILE The view of the data base defined by a DBCB. This is the view used by application programmers who access the data base using that DBCB.

LOGICAL ROOT The root segment for the logical view of the file.

MCALL CHG The TCDMS data base access macro instruction which changes one or more data element values on the data base.

MCALL DEL The TCDMS data base access macro instruction which deletes a data element value from the data base.

MCALL GETx The TCDMS data base access macro instructions which retrieve one or more data elements values from the data base.

MCALL INSx The TCDMS data base access macro instructions which insert data element values into the data base.

MULTIPLE CHAIN FILE A TCDMS file which contains data that is common to several users. Individual users view this data from pointers in other files.

PHYSICAL FILE The actual data base file. The physical view of the file is connected to the application program's logical view by the DBCB. Compare with logical file.

POINTER A segment in a root chain file which identifies a root segment (and its associated family) in another file. TCDMS file linkages are composed of corresponding pointers and back pointers.

POINTER SEGMENT See pointer.

RBF The relative block address of a segment within a family. This address is used by the access method routines to locate a requested segment.

REQUEST MANAGER The set of modules within the Data Management component of TCDMS which interpret an application program request for data base access.

ROOT CHAIN FILE A TCDMS file which contains both data for an individual user and pointers to data in shared, or multiple-chain, files.

ROOT SEGMENT The one segment in a family which identifies that family. All the data in the family is logically related to, and hierarchically dependent on, the root segment.

RSEG The TCDMS macro instruction which redefines a segment to complete the pointer-back pointer interfile linkages.

SCAN POINT A collection of hierarchically related segments containing data elements that uniquely identify the particular data of interest to an application program. The segments included in a scan point form a hierarchical path in the data base.

SCAN POINT ELEMENT (SPE) A control block within a DBCB which identifies the scan point being used by an application program.

SDT See segment descriptor table.

SDTE See segment descriptor table entry.

SEGMENT A set of data elements grouped physically together in a TCDMS file. The data elements in a segment all occupy the same logical position in the data base hierarchy.

SEGMENT DESCRIPTOR An eight-character hexadecimal valve which identifies a segment within the data base.

SEGMENT DESCRIPTOR TABLE (SDT) A table within a DBCB which maps out the physical location of each segment which an application program using that DBCB can access.

SEGMENT DESCRIPTOR TABLE ENTRY (SDTE) An entry in the segment descriptor table which contains a segment descriptor and linkage information for one segment.

SEGMENT DICTIONARY A TCDMS system file which contains the segment descriptions for the segments in the data base.

SEGMENT ID A two-digit hexadecimal valve which identifies a segment on one hierarchical level within a TCDMS file.

SEGMENT LAYOUT A segment definition, followed by data element definitions for each data element in the segment. For segments "pointed to" from other files, a segment layout also includes one or more segment redefinitions to complete the linkages.

SEGMENT PROCESSOR The set of modules within the Data Management component of TCDMS which handle segment level requests from the request manager.

SEGMENT PROCESSOR WORK AREA (SPWA) A work area within a DBCB used by the segment processor.

SEGMENT WORK AREA (SWA) A portion of the DBCB which TCDMS uses to construct segments from data elements for insertion, or to hold segments from which it retrieves individual data elements for an application program.

SHARED FILE A multiple-chain file.

SORTED SEGMENTS Segments stored in a collating sequence based on one data element within each segment.

SPE See scan point element.

SPWA See segment processor work area.

SWA See segment work area.

THREAD An area of main storage available for an executing on-line application program.

VERTICAL PATH A set of segments which includes one segment on each level between the lowest-level segment requested and the root segment for that file. Also called hierarchical path.