April 1971

A Manual with Examples for the Data Description Language (DDL)

Diane P. Smith
University of Pennsylvania

Follow this and additional works at: http://repository.upenn.edu/cis_reports

Recommended Citation
http://repository.upenn.edu/cis_reports/801


This paper is posted at ScholarlyCommons. http://repository.upenn.edu/cis_reports/801
For more information, please contact repository@pobox.upenn.edu.
A Manual with Examples for the Data Description Language (DDL)

Abstract
A Data Description Language (DDL) for describing the organizations of data in files and data bases, is specified. This language has been developed as part of a utility which will process data bases or data files, with existing formats and organizations, and which will produce these data in new desired forms.

The DDL is sufficiently rich and expressive to be readily used to describe the organization of existing data bases, the structure of desired data bases and the transformations between the existing ones to the desired ones. The DDL is specified in the form of an extensive manual containing specifications and a set of detailed examples of the use of the DDL.

Keywords
data description, data structure, storage structure, interfacing, file integration, data extraction, data base, data conversion, file, record

Comments
University of Pennsylvania
THE MOORE SCHOOL OF ELECTRICAL ENGINEERING
Philadelphia, Pennsylvania

TECHNICAL REPORT

A MANUAL WITH EXAMPLES FOR THE
DATA DESCRIPTION LANGUAGE (DDL)

by

Diane P. Smith

April 1971

Submitted to the
Office of Naval Research
Information Systems Branch
Arlington, Virginia

under
Contract N00014-67-A-0216-0007
Research Project NR 049-272

Reproduction in whole or in part is
permitted for any purpose of the
United States Government

Moore School Report No. 71-20
A Data Description Language (DDL) for describing the organizations of data in files and data bases, is specified. This language has been developed as part of a utility which will process data bases or data files, with existing formats and organizations, and which will produce these data in new desired forms.

The DDL is sufficiently rich and expressive to be readily used to describe the organization of existing data bases, the structure of desired data bases and the transformations between the existing ones to the desired ones. The DDL is specified in the form of an extensive manual containing specifications and a set of detailed examples of the use of the DDL.
<table>
<thead>
<tr>
<th>KEY WORDS</th>
<th>LINK A</th>
<th>LINK B</th>
<th>LINK C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Description</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data Structure</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Storage Structure</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Interfacing</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>File Integration</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data Extraction</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data Base</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data Conversion</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>File</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Record</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Part of Dissertation presented
to the University of Pennsylvania
to fulfill the requirement
for the Doctor of Philosophy Degree.
Abstract

A Data Description Language (DDL) for describing the organizations of data in files and data bases, is specified. This language has been developed as part of a utility which will process data bases or data files, with existing formats and organizations, and which will produce these data in new desired forms.

The DDL is sufficiently rich and expressive to be readily used to describe the organization of existing data bases, the structure of desired data bases and the transformations between the existing ones to the desired ones. The DDL is specified in the form of an extensive manual containing specifications and a set of detailed examples of the use of the DDL.
TABLE OF CONTENTS

Page

Index A

PART I INTRODUCTION

1. The Data Description Language 1
2. Basic Description Structure 3

PART II THE DATA DESCRIPTION LANGUAGE

1. Basic Elements 4
   1.1 The Character Set 4
   1.2 Numbers 6
   1.3 Names 6
2. Statements 8
   2.1 Record Specification Statements 9
      2.1.1 FIELD Statements 9
         2.1.1.1 CHAR Statement 15
      2.1.2 GROUP Statements 17
      2.1.3 RECORD Statements 27
      2.1.4 Parameter Statements 31
         2.1.4.1 LENGTH Statements 33
         2.1.4.2 CNT Statements 35
         2.1.4.3 EXIST Statements 36
         2.1.4.4 PARAMVAL Statements 37
      2.1.5 CONSTANT Statements 38
      2.1.6 CONCODE Statements 40
      2.1.7 SET Statements 59
      2.1.8 CRITEX Statements 61
      2.1.9 Alternate Action Statements 72
         2.1.9.1 RPLVAL Statements 73
         2.1.9.2 CRITMESSAGE Statements 75
      2.1.10 CRITERION Statements 78
      2.1.11 Device Statements 80
         2.1.11.1 BBLOCK Statement 81
         2.1.11.2 BLOCK Statement 89
         2.1.11.3 CARDIN Statement 94
         2.1.11.4 CARDOUT Statement 97
         2.1.11.5 TAPEIN Statement 98
         2.1.11.6 TAPEOUT Statement 102
TABLE OF CONTENTS (continued)

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1.11.7 DISKIN Statement</td>
<td>102</td>
</tr>
<tr>
<td>2.1.11.8 DISKOUT Statement</td>
<td>103</td>
</tr>
<tr>
<td>2.1.11.9 TTYIN Statement</td>
<td>111</td>
</tr>
<tr>
<td>2.1.11.10 TTYOUT Statement</td>
<td>114</td>
</tr>
<tr>
<td>2.1.12 Record State Statements</td>
<td></td>
</tr>
<tr>
<td>2.1.12.1 STORRCD Statement</td>
<td>116</td>
</tr>
<tr>
<td>2.1.12.2 LOGRCD Statement</td>
<td>117</td>
</tr>
<tr>
<td>2.2 File Specification Statements</td>
<td></td>
</tr>
<tr>
<td>2.2.1 Linkage Statements</td>
<td></td>
</tr>
<tr>
<td>2.2.1.1 OCC Statement</td>
<td>122</td>
</tr>
<tr>
<td>2.2.1.2 ALLOC Statement</td>
<td>123</td>
</tr>
<tr>
<td>2.2.1.3 SOMEALLOC Statement</td>
<td>127</td>
</tr>
<tr>
<td>2.2.1.4 LINK Statement</td>
<td>129</td>
</tr>
<tr>
<td>2.2.1.5 INVLINK Statement</td>
<td>131</td>
</tr>
<tr>
<td>2.2.2 Linkage Implementation Statements</td>
<td></td>
</tr>
<tr>
<td>2.2.2.1 SEQUEN Statement</td>
<td>136</td>
</tr>
<tr>
<td>2.2.2.2 EMBED Statement</td>
<td>137</td>
</tr>
<tr>
<td>2.2.2.3 DIREC Statement</td>
<td>139</td>
</tr>
<tr>
<td>2.2.3 STORFILE Statement</td>
<td>145</td>
</tr>
<tr>
<td>2.2.4 LOGFILE Statement</td>
<td></td>
</tr>
<tr>
<td>2.3 Task Specification Statements</td>
<td></td>
</tr>
<tr>
<td>2.3.1 Association Statements</td>
<td></td>
</tr>
<tr>
<td>2.3.1.1 ASSOCIATE Statement</td>
<td>155</td>
</tr>
<tr>
<td>2.3.1.2 COMBINE Statement</td>
<td>156</td>
</tr>
<tr>
<td>2.3.1.3 DISJOIN Statement</td>
<td>159</td>
</tr>
<tr>
<td>2.3.2 CREATE Statement</td>
<td>161</td>
</tr>
<tr>
<td>2.3.3 REVISE Statement</td>
<td>162</td>
</tr>
<tr>
<td>2.3.4 RETRIEVE Statement</td>
<td>165</td>
</tr>
<tr>
<td>2.3.5 STORE Statement</td>
<td>170</td>
</tr>
<tr>
<td>2.3.6 UPDATE Statement</td>
<td>171</td>
</tr>
<tr>
<td>2.3.7 Interfacing Statements</td>
<td></td>
</tr>
<tr>
<td>2.3.7.1 PROGDATA Statement</td>
<td>174</td>
</tr>
<tr>
<td>2.3.7.2 INTERFACE Statement</td>
<td>177</td>
</tr>
</tbody>
</table>
TABLE OF CONTENTS (continued)

2.3.8 Language Extension Statements 194
   2.3.8.1 PARAMPROG Statement 195
   2.3.8.2 EXTEND Statement 200

3. Paragraphs 205
   3.1 DESCRIBE Paragraph 206
   3.2 EXECUTE Paragraph 208

4. Descriptions 210
   4.1 DESCRIPTION's 211

PART III EXAMPLES 213

Example 1. Use of the DDL to Convert Files for Interfacing with New Programs and Programming Languages 214
   1.1 The EDMF Record Organization 215
   1.2 The COBOL Record Organization 220
   1.3 The Relationship Description 223
   1.4 The Complete Description of the Conversion 225

Example 2. Use of the DDL to Convert Files for Interfacing with New Operating Systems 227
   2.1 The TSOS SAM File Organization 228
   2.2 The TDOS ISAM File Organization 235
   2.3 The Relationship Description 241
   2.4 The Complete Description of the Conversion 241

Example 3. Use of the DDL to Reorganize Files 243
   3.1 The TITLE-FILE Organization 244
   3.2 The AUTHOR-FILE Organization 247
   3.3 The Relationship Description 250
   3.4 The Complete Description of the Conversion 251
## INDEX A

<table>
<thead>
<tr>
<th>Term</th>
<th>Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALLOC</td>
<td>2.2.1.2</td>
</tr>
<tr>
<td>AND</td>
<td>2.1.8</td>
</tr>
<tr>
<td>ASCLEX</td>
<td>2.1.2</td>
</tr>
<tr>
<td>ASSEMBLY</td>
<td>2.3.7.2, 2.3.8.1, 2.3.8.2</td>
</tr>
<tr>
<td>ASSOCIATE</td>
<td>2.3.1.1</td>
</tr>
<tr>
<td>B</td>
<td>2.1.1, 2.1.4.1, 2.1.5, 2.3.8.1</td>
</tr>
<tr>
<td>BBLOCK</td>
<td>2.1.11.1</td>
</tr>
<tr>
<td>BLOCK</td>
<td>2.1.11.2, 2.2.2.2, 2.2.2.3</td>
</tr>
<tr>
<td>C</td>
<td>2.1.1, 2.1.4.1, 2.1.5, 2.3.8.1</td>
</tr>
<tr>
<td>CARD</td>
<td>2.1.9.2</td>
</tr>
<tr>
<td>CARDIN</td>
<td>2.1.11.3</td>
</tr>
<tr>
<td>CARDOUT</td>
<td>2.1.11.4</td>
</tr>
<tr>
<td>CLGTH</td>
<td>2.2.2.2</td>
</tr>
<tr>
<td>CNT</td>
<td>2.1.4.2</td>
</tr>
<tr>
<td>COBOL</td>
<td>2.3.7.2, 2.3.8.1, 2.3.8.2</td>
</tr>
<tr>
<td>COLEND</td>
<td>2.1.6</td>
</tr>
<tr>
<td>COLST</td>
<td>2.1.6</td>
</tr>
<tr>
<td>COMBINE</td>
<td>2.3.1.2</td>
</tr>
<tr>
<td>COMPLEX</td>
<td>2.3.7.1</td>
</tr>
<tr>
<td>CONCODE</td>
<td>2.1.6</td>
</tr>
<tr>
<td>CONSTANT</td>
<td>2.1.5</td>
</tr>
<tr>
<td>CREATE</td>
<td>2.3.2</td>
</tr>
<tr>
<td>CRITERION</td>
<td>2.1.10</td>
</tr>
<tr>
<td>CRITEX</td>
<td>2.1.8</td>
</tr>
<tr>
<td>CRITMESSAGE</td>
<td>2.1.9.2</td>
</tr>
</tbody>
</table>
INDEX A (continued)

CSUTS 2.2.2.2
DELETE 2.3.3
DELIM 2.1.6
DESCRIBE 3.1
DESCRIPTION 4.1
DESCRIPTIONEND 4.1
DIREC 2.2.2.3
DISJOIN 2.3.1.3
DISKIN 2.1.11.7
DISKOUT 2.1.11.8
DSCLEX 2.1.2
EMBED 2.2.2.2
EMBPTRDIREC 2.2.2.3
END 3.1, 3.2
EQ 2.1.8
EXECUTE 3.2
EXIST 2.1.4.3
EXTEND 2.3.8.2
F 2.1.1, 2.1.2, 2.1.11.1, 2.2.1.4, 2.3.1.2, 2.3.1.3
FIELD 2.1.1
FILLER 2.1.6
FIXED 2.1.1, 2.1.2, 2.1.11.1, 2.2.1.4, 2.3.1.2, 2.3.1.3
FL 2.1.1
FORTRAN 2.3.7.2, 2.3.8.1, 2.3.8.2
INDEX A (continued)

FROM 2.3.4
FX 2.1.1
GE 2.1.8
GROUP 2.1.2
GT 2.1.8
HDR 2.1.11.1, 2.1.11.2
I 2.3.7.1
INTERFACE 2.3.7.2
INVLINK 2.2.1.5
INX 2.1.6
LE 2.1.8
LENGTH 2.1.4.1
LINK 2.2.1.4
LOGFILE 2.2.4
LOGRCDS 2.1.12.3
LT 2.1.8
M 2.1.2, 2.1.11.1, 2.1.11.2
MANDATORY 2.1.2, 2.1.11.1, 2.1.11.2
MEM 2.1.8
NEWLINE 2.1.6
NEWPAGE 2.1.6
NOLIMS 2.1.1, 2.1.2, 2.1.11.1, 2.1.11.2, 2.2.1.4,
2.3.1.2, 2.3.1.3
NOORD 2.1.2, 2.1.11.1, 2.1.11.2, 2.2.1.4
NOT 2.1.8
NQ 2.1.8
INDEX A (continued)

0 2.1.2, 2.1.11.1, 2.1.11.2, 2.3.7.1

OCC 2.2.1.1

OF 2.1.4

OPTIONAL 2.1.2, 2.1.11.1, 2.1.11.2

OR 2.1.8

PARAMPROG 2.3.8.1

PRINTER 2.1.9.2

PROGDATA 2.3.7.1

PRX 2.1.6

PTX 2.1.6

RECORD 2.1.3, 2.2.2.2, 2.2.2.3

RETRIEVE 2.3.4

REVISE 2.3.3

RPLVAL 2.1.9.1

S 2.2.1.1

SEQUEN 2.2.2.1

SET 2.1.7

SIMPLE 2.3.7.1

SOMEOCC 2.2.1.3

SPEC 2.1.2, 2.1.11.1, 2.1.11.2

SPLIT 2.1.11.1

STORE 2.3.5

STORFILE 2.2.3

STORRCD 2.1.12.1
INDEX A (continued)

<table>
<thead>
<tr>
<th>Term</th>
<th>Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>2.2.2.1</td>
</tr>
<tr>
<td>TAPEIN</td>
<td>2.1.11.5</td>
</tr>
<tr>
<td>TAPEOUT</td>
<td>2.1.11.6</td>
</tr>
<tr>
<td>TLR</td>
<td>2.1.11.1, 2.1.11.2</td>
</tr>
<tr>
<td>TTY</td>
<td>2.1.9.2</td>
</tr>
<tr>
<td>TTYIN</td>
<td>2.1.11.9</td>
</tr>
<tr>
<td>TTYOUT</td>
<td>2.1.11.10</td>
</tr>
<tr>
<td>UPDATE</td>
<td>2.3.6</td>
</tr>
<tr>
<td>V</td>
<td>2.1.1, 2.1.2, 2.1.11.1, 2.2.1.4, 2.3.1.2, 2.3.1.3</td>
</tr>
<tr>
<td>VARIABLE</td>
<td>2.1.1, 2.1.2, 2.1.11.1, 2.2.1.4, 2.3.1.2, 2.3.1.3</td>
</tr>
<tr>
<td>VIALINK</td>
<td>2.3.4</td>
</tr>
<tr>
<td>WHOLE</td>
<td>2.1.11.1</td>
</tr>
<tr>
<td>Xn</td>
<td>2.2.1.1, 2.2.1.2, 2.2.1.3</td>
</tr>
</tbody>
</table>
PART I INTRODUCTION

1. The Data Description Language

The Data Description Language (DDL) has been developed to serve as a new type of "utility" in computer systems. It is to be used to process data bases or data files, with existing formats and organizations, and produce these data in new desired forms.

This manual presents specifications of the language and examples of its use. In addition to examples which demonstrate the correct usage of each statement in the DDL, there is a set of comprehensive examples. These examples illustrate some of the broad applications of the DDL which are described below:

1) Interfacing of files with new programs and new programming languages. Frequently it is not possible for new programs written in one language to process files created by programs in another language. With the DDL descriptions, effective interfaces between files and programs can be achieved. The DDL processor would translate the old file so that it can be processed by the changed or new program. In this manner, files can be interfaced across programming language barriers.

2) Interfacing files with new operating systems and new data management systems. Frequently it is not possible for files created under one operating system or data management system to be processed under a different operating system or data management system. With the DDL descriptions, the translation of files for processing by new operating systems or data management system can be achieved.

3) Reorganizing of files. To achieve more effective use of data and to take advantage of new random access capabilities, it is
frequently advantageous to reorganize the data in a file by creating
single or multi-level directories of the data and by structuring data
for most effective processing.

4) Integration of files. Fragmentation of information in nu-
merous files frequently causes great difficulties in use and considerable
inefficiency in processing. The DDL processor will be able to integrate
many files into one.

5) Extraction of data. If only a small amount of data in a
file is used by a program, it is sometimes far more effective to create
a smaller file consisting only of the useful data. The DDL processor
provides this capability which allows the creation of many files from
one file.

6) Creation of new data bases. The combination of the above
two capabilities allows the translating of one set of files into
another set of files.

7) Interfacing files with new computers. Increased requirements
and new technology require the phasing out of old computers and their
replacement by new computers. The DDL processor would be able to
translate old files for transferring from the old computer to the new.

8) Interfacing files to use new devices. The advance of tech-
nology introduces new input/output devices which enhance the total cost
effectiveness of the entire computer system. The change of such input/
output devices can be facilitated by the DDL processing of the files
from the old devices to the new ones.
2. Basic Description Structure

A program written in the DDL is called a description. This terminology is used to emphasize the non-procedurality of the language, i.e., DDL users must describe what they want done, rather than prescribe how they want it done.

A DDL description is a series of one or more paragraphs. A paragraph is a series of one or more statements delimited by system names. A statement is a string consisting of the basic elements of the DDL. These are system names, user-defined names, numbers, and characters.
PART II THE DATA DESCRIPTION LANGUAGE (DDL)

1. Basic Elements

1.1 The Character Set

The Character Set is that of the host machine. In this case, the host machine is the RCA Spectra 70/46.

The Character Set is composed of digits, alphabetic characters, and special characters. There are ten digits:

0, 1, 2, 3, 4, 5, 6, 7, 8, 9 are decimal digits;
0, 1 are binary digits.

There are twenty-six alphabetic characters:

U, V, W, X, Y, Z

There are twenty-nine special characters:

<table>
<thead>
<tr>
<th>Character name</th>
<th>Character</th>
</tr>
</thead>
<tbody>
<tr>
<td>comma</td>
<td>,</td>
</tr>
<tr>
<td>semicolon</td>
<td>;</td>
</tr>
<tr>
<td>colon</td>
<td>:</td>
</tr>
<tr>
<td>period</td>
<td>.</td>
</tr>
<tr>
<td>question mark</td>
<td>?</td>
</tr>
<tr>
<td>exclamation</td>
<td>!</td>
</tr>
<tr>
<td>apostrophe</td>
<td>'</td>
</tr>
<tr>
<td>quote</td>
<td>&quot;</td>
</tr>
<tr>
<td>open parenthesis</td>
<td>(</td>
</tr>
<tr>
<td>close parenthesis</td>
<td>)</td>
</tr>
</tbody>
</table>

space
<table>
<thead>
<tr>
<th>Character name</th>
<th>Character</th>
</tr>
</thead>
<tbody>
<tr>
<td>number sign</td>
<td>#</td>
</tr>
<tr>
<td>ampersand</td>
<td>&amp;</td>
</tr>
<tr>
<td>asterisk</td>
<td>*</td>
</tr>
<tr>
<td>at the rate of</td>
<td>@</td>
</tr>
<tr>
<td>cents</td>
<td>¢</td>
</tr>
<tr>
<td>dollar sign</td>
<td>$</td>
</tr>
<tr>
<td>lozenge</td>
<td>⬤</td>
</tr>
<tr>
<td>underline</td>
<td>_</td>
</tr>
<tr>
<td>plus</td>
<td>+</td>
</tr>
<tr>
<td>minus</td>
<td>-</td>
</tr>
<tr>
<td>slash</td>
<td>/</td>
</tr>
<tr>
<td>equals</td>
<td>=</td>
</tr>
<tr>
<td>less than</td>
<td>&lt;</td>
</tr>
<tr>
<td>greater than</td>
<td>&gt;</td>
</tr>
<tr>
<td>percent</td>
<td>%</td>
</tr>
<tr>
<td>logical AND</td>
<td>∧</td>
</tr>
<tr>
<td>logical OR</td>
<td></td>
</tr>
<tr>
<td>logical NOT</td>
<td>¬</td>
</tr>
</tbody>
</table>

The characters in the character set are the most primitive elements of the DDL. The characters are combined to form strings called names and strings called numbers.

Note: In this manual when individual characters are discussed, they will be enclosed in quotes.
1.2 Numbers

An integer is a string of digits.

Examples:

0  
124  
03791

A signed integer is the character "+" or the character "-" followed by an integer.

Examples:

+1  
-124  
+03791

A floating point number is an integer or a signed integer, followed by the character ".", followed by an integer.

Examples:

9.012  
-102.0  
+1378.999

A number is an integer, a signed integer or a floating point number.

1.3 Names

A name is any string of characters from the character set.

An index is an integer enclosed by the characters "(" and ")".

Examples:

(10)  
(0)  
(131)

Note: From the above definition, signed integers and floating point numbers cannot appear in indices.

A user-defined name body is any string of characters from the character set excluding the characters "":, "(" and ")".

Examples:

GROUP ??::!A 9.012  
-102.0 NAME FALSE
An unindexed user-defined name is a user-defined name body enclosed by apostrophes.

Examples:

'GROUP'  '??:!A'  '9.012'
'1-02.0'  'NAME'   'FALSE'

An indexed user-defined is a user-defined name body, followed by an index, with the whole string enclosed by apostrophes.

Examples:

'NAME(10)'
'9.012(0)'
'FALSE(131)'

A user-defined name is either an indexed user-defined name or an unindexed user-defined name.

Examples:

'GROUP'
'NAME(10)'

A system name is any of the names listed in Index A.
2. Statements

Statements in the DDL have a fixed format consisting of a system name, followed by a sequence of parameters enclosed in parenthesis.

The statements are grouped together into three sections: Record Specification Statements, File Specification Statements and Task Specification Statements.

Each statement in the DDL is explained in a separate subsection. The format of each statement is presented, followed by a description of each parameter. A discussion of the usage of each statement, together with examples, is then given.

Note: In presenting statement formats, the following conventions are used:

(i) square brackets indicate optional parameters.

(ii) "parameter x, ..., parameter x" means that parameter x may occur in the statement one or more times.

(iii) "[ parameter x, ..., parameter x ]" means that parameter x may occur in the statement zero or more times.
### 2.1 Record Specification Statements

#### 2.1.1 FIELD Statement

| format | FIELD (field name, uniformity, length type, length, data type [; F, alignment factor, alignment ] [; V, alignment ] [; CONCODE statement, ...], CONCODE statement ) |
| parameters | (i) field name is an unindexed user-defined name. (ii) uniformity is either the system name: FIXED (or simply F), or VARIABLE (or simply V). (iii) length type is either the system name: B, or C; or a user-defined name. (iv) length is either the string: n, where n is an integer, or the system name NOLIM. (v) data type is the same as parameter (iii) (vi) alignment factor is the string: n, where n is an integer (vii) and (viii) alignment is the parameter string: orientation, pad factor where: orientation is either the system name L or R; and pad factor is a CONSTANT statement (see section 2.1.5) |

Note: The optional parameters will not be discussed in this section. Their format and usage will be discussed in Section 2.1.4.
The FIELD statement is used to specify a field.*

A field is the form of a set of values. A value is a string of characters or binary digits. Two values have the same form if: they are referred to by the same user-defined name (parameter i) and if both have the same length type, length and data type specification (parameters ii, iii, and iv). Thus, to specify a field it is necessary to include the following information in the FIELD statement:

(i) field name. This parameter is the name of the field being specified. It is used to refer to values of the field.

(ii) uniformity. This parameter is used to specify whether the number of characters or bits, the length, of each value of the field is the same. The parameter must be assigned the system name:

FIXED (or F) when the length of each value is to be the same, and

VARIABLE (or V) when the length of each value of the field may be different.

(iii) length type. This parameter is used to specify whether the length is to be given in bits or characters. The parameter must be assigned the character

B if the length is to be given in bits.

* Note that the term "field" is used in many different ways in computer literature. A field has been variously defined as: (1) the name (or attribute) of a value; (2) the value itself; (3) the name and the value. In this manual it will be used exclusively as defined above.
C if the length is given in 8 bit characters, or 
a user-defined name if the length is given in characters 
of length other than 8 bits. The name must appear 
as the first parameter in a CHAR statement. The use 
of this option will be discussed in Section 2.1.1.1.

(iv) length. This parameter is used to specify the number of 
characters or bits in a value. The parameter must be 
assigned the string:

\( n \) when the lengths of the values are fixed, and \( n \) 
characters or bits long, or

when the lengths of the values may vary and are, at 
most, \( n \) characters or bits long;

NOLIM when the lengths of the values may vary and are 
not limited.

(v) data type. This parameter is used to specify the data 
type of the values of a field. The parameter must be 
assigned the system name;

B when the values of the field are to be a string of 
binary digits,

C when the values are to be encoded in ASCII characters,

a user defined name when the values are encoded as non-
ASCII characters. The name must appear as the first 
parameter in a CHAR statement (see Section 2.1.1.1).

(vi) alignment factor. This parameter is used in specifying 
the alignment of fields with respect to arbitrary boundaries 
(e.g. word, and half-word boundaries, etc.). The integer 
specifies the number of bits between alignment boundaries.
(vii) alignment. This parameter specifies the actual alignment of the field. The parameter orientation is assigned the system name:

L when the field is to end on a boundary, and
R when the field is to begin on a boundary.

The parameter pad factor specifies the characters that are to fill unused storage between the preceding field and the field being aligned.

(viii) alignment. This parameter specifies the alignment of a value within a field when the value is longer or shorter than the number of storage spaces reserved for it. The parameter orientation is assigned the system name:

L when the value is to be left aligned (truncation or padding occurs at the right), and
R when the value is to be right aligned (truncation or padding occurs at the left).

The parameter pad factor specifies the characters that are to fill unused storage.

Example 1

Consider a set of values, each of which is an ASCII character string of variable length. Assume the maximum length of each value is 30 characters. The values are to be referred to by the name 'COLLEGE' and are to be interpreted as college names.

The following statement specifies a field for these values:

FIELD ( 'COLLEGE', V, C, 30, C )

where: 'COLLEGE' is the field name.

V specifies that the lengths of the values of 'COLLEGE' may vary.
C specifies that the length is to be given in characters.
30 specifies that no value of 'COLLEGE' may exceed
30 characters.
C specifies that the values of 'COLLEGE' are to be interpreted as strings of characters.
The following character strings can be values of the field specified in the above DDL statement:
UNIVERSITY OF PENNSYLVANIA
PURDUE
YALE

Example 2 Consider a set of values, each of which is an ASCII character string of fixed length. Assume the length of each value is 2 characters. The values are to be referred to by the name 'YRS' and are to be interpreted as the number of years spent at a college.

The following statement specifies a field for these values:

FIELD ( 'YRS', F, C, 2, C )

where: 'YRS' is the field name.
F specifies that the lengths of the values of 'YRS' must all be the same.
C specifies that the length is to be given in characters.
2 specifies that each value of 'YRS' must have a length of 2 characters.
C specifies that the values of 'YRS' are to be interpreted as strings of characters.
The following character strings can be values of the field specified in the above DDL statement:

04
02
10

Use of FIELD statements to determine parameters

If the string to be assigned a parameter is stored as the value of a field, the name of the field may be used as the parameter. For example, if the length of a field is to equal on the value of the field 'YRS' of Example 2, then the name 'YRS' appears in the position of the length parameter.
2.1.1.1 CHAR Statement

<table>
<thead>
<tr>
<th>format</th>
<th>CHAR ( character set name; character size, character set description )</th>
</tr>
</thead>
</table>
| parameters | (i) character set name is an unindexed user-defined name.  
(ii) character size is the string n, where n is an integer.  
(iii) character set description is either the system name:  
       EBCDIC, or  
       BCD, or the string  
       user-defined name, user-defined name. |

usage of the statement

The CHAR statement is used to specify the character set to be used to represent values. An existing character set may be specified or a new character set may be defined in terms of an existing one. Thus, to specify a character set, it is necessary to include the following information in the CHAR statement:

usage of the parameters

(i) character set name. This parameter is used to refer to the character set to be used.  
(ii) character size. This parameter is used to specify the length of a character in bits.
(iii) character set description. This parameter is used to specify the character set to be used. The parameter is assigned the system name: EBCDIC when the EBCDIC character code is to be used, and BCD when the BCD character code is to be used. user-defined name, user-defined name is used when a new code is to be defined. Each name must appear as the first parameter in a SET statement. The use of this option is described in Section 2.1.7.

Example

Consider a set of values, each of which is an EBCDIC character string of fixed length. Assume the length of each value is 2 characters. The values are to be referred to by the name 'AGE' and are to be interpreted as EBCDIC character strings.

The following statements specify the field for these values:

```
FIELD ( 'AGE', F, 'E', 2, 'E' )
CHAR ( 'E'; 8, EBCDIC )
```
2.1.2 GROUP Statements

format

GROUP ( group name, group order; ( list ),
... , ( list ) [; CONCODE statement, ... ,
CONCODE statement ] )

Note: the optional parameters, CONCODE statement,
will not be discussed in this section. Their
format and usage will be discussed in Section 2.1.4.

parameters

(i) group name is an unindexed user-defined name.

(ii) group order is either the system name:
    NOORD, or
    SPEC.

(iii) list is a string of parameters with the
     following format:
     name, optionality, repetition uniformity,
     repetition number [, 0, order ]
     [, V, criterion name ]

     where:
     a) name is an unindexed user-defined name;
     b) optionality is either the system name:
        MANDATORY (or simply M), or
        OPTIONAL (or simply 0);
     c) repetition uniformity is either the system
        name:
        FIXED (or simply F), or
        VARIABLE (or simply V).
d) repetition number is either the string:
n, where n is an integer, or the
system name
NOLIM.
e) the optional parameter, order, is
either the system name:
ASCLEX, or
DSCLEX, or
a user-defined name.
f) the optional parameter, criterion name,
is a user-defined name.

The GROUP statement is used to specify a group.
A **group** is an organization of values of different
fields and/or other groups. Values of different fields
and groups are in an **organization** if: they are referred
to by a user-defined name (parameter i) and if they
appear as specified in parameters ii and iii. Groups*
which are to be included in the group being specified
will be called **subordinate groups**. To specify a group,
it is necessary to include the following information in
the GROUP statement:

* The term "group" will also be used to refer to the set of values
  of fields and subordinate groups that are in the group being specified,
  when this does not lead to confusion.
usage of parameters

(i) group name. This parameter is the name of the group being specified. It is used to refer to the values and subordinate groups of the group.

(ii) order. This parameter is used to specify the sequence in which field values and/or subordinate groups are to occur. The parameter must be assigned the system name:

NOORD when the field values and/or subordinate groups may occur in any order in the group being specified; and

SPEC when the fields and/or subordinate groups must occur in the order in which they are listed in parameter iii.

(iii) list. Each list describes the occurrence of a field value (or values) or of a subordinate group (or groups) in the group being specified. The list parameters must be assigned strings as follows:

a) name. This parameter is the name of a field or subordinate group. The name must appear as the first parameter in a FIELD statement if it refers to a field. The name must appear as the first parameter in a GROUP statement if it refers to a group.
b) optionality. This parameter is used to specify whether the occurrence of the field or subordinate group named by the first parameter of the list is mandatory or optional. The parameter must be assigned the system name:

MANDATORY (or simply M) when the occurrence is mandatory, and

OPTIONAL (or simply O) when the occurrence is not mandatory.

c) repetition uniformity. This parameter is used to specify whether the repetition number of the field or subordinate group, named by the first parameter of the list, is the same for each occurrence of the group. The repetition number is the number of times the field values or subordinate groups occur in the group. The parameter must be assigned the system name:

FIXED (or F) when the repetition number of each field or subordinate group is to be the same in each occurrence of the group, and

VARIABLE (or V) when the repetition number of each field or subordinate group may be different in each occurrence of the group.
d) repetition number. This parameter is used to specify the number of times the field values or subordinate groups named by the first parameter of the list are to occur in the group. The parameter must be assigned the string:

- $n$ when the field or subordinate group is to occur $n$ times, or at most, $n$ times, and
- NOLIM when the field or subordinate group is to occur an unknown number of times.

e) order. This optional parameter is used to specify the sequence in which occurrences of a repeating field or repeating subordinate group are to occur in the group being specified. The parameter must be given either the system name:

- ASCLEX when the values of a field are to occur in ascending lexicographical order;
- DSCLEX when the values of a field are to occur in descending lexicographical order;
- or
- a user-defined name which appears as the first parameter in a LINK statement.

The use of this option is discussed in Section 2.2.1.4.
The parameter is omitted if the field or subordinate group occurrences are to occur in any order.

f) the optional parameter, criterion name.

This parameter refers to a criterion which is defined on the values of fields. If the criterion is not satisfied, the field or subordinate group named by parameter a, described above, is rejected. The criterion name must appear as the first parameter in a CRITERION statement. These statements and examples of GROUP statements containing criterion names as parameters will be discussed in Section 2.1.7.

Example 1

Consider two sets of values:

i) the set of values described in Example 1 of Section 2.1.1.

ii) the set of values described in Example 2 of Section 2.1.1.

Values from these sets are to be organized in a group called 'COLINF'. One value from each set is to appear; the value from the first set is to appear first and the value from the second set is to appear second.

The following statements specify this group:

FIELD ( 'COLLEGE', V, C, 30, C )
FIELD ( 'YRS', F, C, 2, C )
GROUP ( 'COLINF', SPEC;
    ( 'COLLEGE', M, F, 1 ),
    ( 'YRS', M, F, 1 ) )

where: 'COLINF' is the group name.

SPEC specifies that the field values and subordinate groups are to occur in the order in which they are specified in the list.

'COLLEGE' is the name of the first field value in 'COLINF'.

M specifies that the occurrence of a value of 'COLLEGE' is mandatory.

F specifies that the number of occurrences of values of 'COLLEGE' is fixed.

1 specifies that only 1 value of 'COLLEGE' will occur in 'COLINF'.

'YRS' is the name of the second field value in 'COLINF'.

M specifies that the occurrence of a value of 'YRS' is mandatory.

F specifies that the number of occurrences of values of 'YRS' is fixed.

1 specifies that only 1 value of 'YRS' will occur in 'COLINF'.

The structure specified by the above GROUP statement is illustrated in the diagram given below:
Consider three sets of values:

i) a set of values, each of which is a character string of variable length. The maximum length of each value is 15 characters. The values are to be referenced by the name 'SURNAME' and are to be interpreted as a person's last name.

ii) the set of values described in Example 1 of Section 2.1.1. These are values of the field 'COLLEGE'.

iii) the set of values described in Example 2 of Section 2.1.1. These are values of the field 'YRS'.

Values from these sets are to be organized in a group called 'PERSDATA'. One value from value set i must appear. Values from sets ii and iii may appear zero or more times, in the organization described in Example 1 above (i.e., in group 'COLINF'). The order of appearance must always be: a 'SURNAME' value followed by one or more sets of 'COLINF' values.
The following statements specify this group:

FIELD ('SURNAME', V, C, 15, C)
FIELD ('COLLEGE', V, C, 30, C)
FIELD ('YRS', F, C, 2, C)
GROUP ('COLINF', SPEC;
    ( 'COLLEGE', M, F, 1 ),
    ( 'YRS', M, F, 1 )
GROUP ('PERSDATA', SPEC;
    ( 'SURNAME', M, F, 1 ),
    ( 'COLINF', O, V, NOLIM )

where: 'PERSDATA' is the group name.

SPEC specifies that the field values and subordinate groups are to occur in the order in which they are specified in the lists.

'SURNAME' is the name of the value which is to occur first in 'PERSDATA'.

M specifies that the occurrence of a value for 'SURNAME' is mandatory.

F specifies that the number of occurrences of values of 'SURNAME' is fixed.

1 specifies that only 1 value of 'SURNAME' will occur in 'PERSDATA'.

'COLINF' is the name of the group of values that is to occur second in 'PERSDATA'.

O specifies that the occurrence of a group of values for 'COLINF' is optional.
V specifies that the number of occurrences of values of 'COLINF' may vary from one occurrence of 'PERSDATA' to another.

NOLIM specifies that there is no limit set on the number of groups of values of 'COLINF' that may occur in 'PERSDATA'.

The structure specified by the above GROUP statement is illustrated in the diagram given below:

```
| 'SURNAME' |
| 'COLLEGE' |
| 'YRS'     |
| ...       |
| 'COLLEGE' |
| 'YRS'     |
```

'COLINF(1)'

The following sets of values satisfy the specification of the group 'PERSDATA':

1) DANIELS

2) DANIELS PURDUE 03

3) DANIELS PURDUE 03 UNIVERSITY OF PENNSYLVANIA 01
2.1.3 RECORD Statement

**Format**

```
RECORD ( record name, group name [; criterion name ]
[; CONCODE statement, ... , CONCODE statement ] )
```

Note: the optional parameters, CONCODE statement, will not be discussed in this section. Their format and usage will be discussed in Section 2.1.6.

**Parameters**

(i) record name is a user-defined name.

(ii) group name is a user-defined name.

(iii) criterion name is a user-defined name.

**Usage of the Statement**

The RECORD statement is used to specify a record. A record is a group whose values are to be the basic unit of storage and retrieval. That is, all the values of the group are to be retrieved to satisfy a single retrieval request made on a set of data. Each set of values which occurs in a structure defined to be a record is called an occurrence of that record (or record occurrence). To specify a record, it is necessary to include the following information in the RECORD statement:

(i) record name. This parameter is used to refer to the record being specified.

(ii) group name. This parameter is the name of a group which is being declared a record. The name must appear as the first parameter of a GROUP statement.
(iii) criterion names. These optional parameters refer to criteria defined on the values of fields in the record. If the values do not satisfy the criteria, the complete set of values of the group named by parameter ii are rejected for storage. Each criterion name must appear as the first parameter of a CRITERION statement. This statement and examples of RECORD statements containing criterion names will be discussed in Section 2.1.10.

Consider three sets of values (those described in Example 2, Section 2.1.2):

i) a set of values, each of which is a character string of variable length. The maximum length of each value is 15 characters. The values are referenced by the name 'SURNAME' and are interpreted as a person's last name.

ii) a set of values, each of which is a character string of variable length. The maximum length of each value is 30 characters. The values are referenced by the name 'COLLEGE' and are interpreted as college names.

iii) a set of values, each of which is a character string of a fixed length of 2 characters. The values are referenced by the name 'YRS' and are interpreted as the number of years spent at a college.
Values from these sets are to be organized as records for storage. The record values are to have the organization described in Example 2, Section 2.1.2. That is, values from these sets are to be organized in a group called 'PERSDATA'. One value from value set i must appear. Values from sets ii and iii may appear zero or more times, in the organization described in Example 1, Section 2.1.2 (i.e., in group 'COLINF'). The order of appearance must always be: a 'SURNAME' value followed by one or more sets of 'COLINF' values.

The following statements specify this record:

```
FIELD ( 'SURNAME', V, C, 15, C )
FIELD ( 'COLLEGE', V, C, 30, C )
FIELD ( 'YRS', F, C, 2, C )
GROUP ( 'COLINF', SPEC;
     ( 'COLLEGE', M, F, 1 ),
     ( 'YRS', M, F, 1 ) )
GROUP ( 'PERSDATA', SPEC;
     ( 'SURNAME', M, F, 1 ),
     ( 'COLINF', O, V, NOLIM ) )
RECORD ( 'PERSRCD', 'PERSDATA' )
```

where: 'PERSRCD' is the record name.

'PERSDATA' is the name of the group whose values are to be stored and retrieved as a unit.
The following groups of values satisfy the above DDL statements:

i) DANIELS

ii) DANIELS PURDUE 03

iii) DANIELS PURDUE 03 YALE 01
2.1.4 Parameter Statements

Parameter statements are statements that are used as parameters in other DDL statements. They may be used for any arbitrary parameter. These statements are used when the user wants a parameter to be determined by a field or group. For example, a user may want to have the length of a field equal to the number of times another field repeats in a record. Parameter statements allow such a relationship to be described.

Since a field or group may appear more than once in a record (by repetition or by inclusion in different groups), specific values of the field or group must be referred to in a parameter statement. Such referencing is done by modifying the field or group name in the following ways:

(i) if the field, or group, named 'X' repeats, the n<sup>th</sup> value of the field, or n<sup>th</sup> values of the group, is referred to by the indexed name: 'X(n)' where n is an integer.

(ii) if the field or group named 'X' occurs in more than one group, say in groups named 'T', 'U', and 'V', then the values of the field or group named 'X' are referred to by the name: 'X' OF 'U'. As many of the phrases "OF group name" may be specified as are necessary to distinguish between different values of the field or group. For example, in the above case, if the group 'U' occurs in groups named 'S' and 'R', then the name:

'X' OF 'U' OF 'S'

refers to the values of 'X' which occur in 'U' which occurs in 'S'.
In defining different organizations of values, it will occasionally be necessary to indicate the record and the organization of records and their values in which values of a field or group occur. Such referencing is done by modifying the field or group name in the following ways:

(iii) if the field, or group, named 'X' occurs in a record 'RCD1', then the values of the field or group are referred to by the name:

'X' OF 'RCD1'

If the field or group occurs in one or more groups, the group in question, say 'U', is specified before the record name:

'X' OF 'U' OF 'RCD1'

(iv) if the record, 'RCD1' containing a field or group, 'X', is organized in some structure named 'Z', then the values of the field or group are referred to by the name:

'X' OF 'RCD1' OF 'Z'

(v) in general, whenever a structure, 'S1', occurs as part of another structure, 'S2', then the structure 'S1' can always be referred to by the name

'S1' OF 'S2'

Names, as described in (i), (ii), (iii), (iv), and (v) above, are called reference names.
2.1.4.1 LENGTH Statements

**Format**

LENGTH (data name, length type)

**Parameters**

(i) data name is a reference name

(ii) length type is either the string:
     B, or
     C.

**Usage of the Statement**

The LENGTH statement can be used as a parameter when the parameter can be an integer (e.g., the length parameter in the FIELD statement, or the repetition number parameter in the GROUP statement). The LENGTH statement is used when the parameter is to be assigned the length of field, or group, etc.

**Usage of the Parameters**

(i) reference name. This parameter refers to the value of a field or group, etc. whose length is to be used as a parameter.

(ii) length type. This parameter specifies whether the length of the field referred to by parameter (i) is to be given in characters or bits.

**Example**

Consider a field 'COLLEGE' whose values have variable lengths of maximum 30 characters and are interpreted as character strings. Assume the field occurs once in a record. The user may define another field 'X' whose values
are to be interpreted as character strings and are to have the same number of bits in length as 'COLLEGE' has in characters (i.e., if a value for 'COLLEGE' has a length of \( n \) characters, then the value for 'X' has \( n \) bits).

The following statement specifies this field:

\[
\text{FIELD ( 'X', V, B, LENGTH ( 'COLLEGE', C ), C )}
\]

Thus, in a record, if the value of the field 'COLLEGE' has a length of 3 characters, then the values of 'X' will have a length of 3 bits.
### 2.1.4.2 CNT Statements

<table>
<thead>
<tr>
<th>format</th>
<th>CNT( data name )</th>
</tr>
</thead>
<tbody>
<tr>
<td>parameter</td>
<td>data name is a reference name</td>
</tr>
<tr>
<td>usage of the statement and parameter</td>
<td>The CNT statement can be used as a parameter when the parameter can be an integer. The CNT statement is used when the parameter is to be assigned the number of times a field, or group, etc. referred to by the reference name, occurs.</td>
</tr>
<tr>
<td>Example</td>
<td>Consider a group 'COLINF' which repeats zero or more times in a record. The user may define a field 'Y' whose values are to be interpreted as character strings and are to have the same number of characters in length as the number of times 'COLINF' repeats. The following statement specifies this field:</td>
</tr>
<tr>
<td></td>
<td><code>FIELD ( 'Y', V, C, CNT ( 'COLINF' ), C )</code></td>
</tr>
<tr>
<td></td>
<td>Thus, in a record if 'COLINF' repeats twice, then values of field 'Y' will have a length of 2 characters.</td>
</tr>
</tbody>
</table>
2.1.4.3 EXIST Statements

**format**

EXIST ( data name )

data name is a reference name.

**usage of the statement and parameter**

The EXIST statement can be used as a parameter when the parameter can be an integer. The EXIST statement is used when the parameter is to be the integer 1 if the values of another field or group occur and 0 if the values of the other field or group do not occur.

Consider a group 'COLINF' which repeats zero or more times in a record. The user may define a group 'Z' in which a field 'Y' repeats once, if values for 'COLINF' occur at least once, and which does not repeat, if no values for 'COLINF' occur.

The following statement specifies this group:

```plaintext
GROUP ( 'Z', SPEC;

( 'Y', M, EXIST ( 'COLINF' ) ) )
```
2.1.4.4 PARAMVAL Statement

format

PARAMVAL ( name, parameter number )

parameters

(i) name is a reference name.
(ii) parameter number is the string n, where n is an integer.

usage of the statement and parameters

The PARAMVAL statement is used as a parameter when the parameter is to be determined by the value of another parameter which may vary. Name gives the name of the entity whose parameter is to be used. Parameter number is used to specify which parameter is to be used.

Example

Consider a field 'X' with the following structure:

FIELD ( 'X', V, C, NOLIM, C )

If the length of a second field 'Y' is to depend on the length of an occurrence of field 'X' this can be specified in the following way:

FIELD ( 'Y', V, C, PARAMVAL ( 'X', 4 ), C )

The LENGTH statement can also be used to specify this relationship. In general, this statement is meant to describe relationships that are not covered by the previous parameter statements.
2.1.5 CONSTANT Statements

format

CONSTANT ( character string, data type )

parameter

(i) character string is any string over the character set. If the characters "(" or ")" appear in the character string, they must be surrounded by apostrophes.

(ii) data type is either the system name:

   B, or

   C; or a user-defined name.

usage of the statement and parameters

The CONSTANT statement is used as a parameter when arbitrary character strings are required. The system name CONSTANT and the parentheses serve to delimit the character string for the DDL processor. For this reason, if parentheses must occur in the character string, then they must be surrounded by apostrophes.

The data type must be assigned the system name:

   B, when the string contains only the characters "0" and "1" and the arbitrary string is to be a bit string,

   C, when the arbitrary string is to be a string of ASCII characters,

   a user-defined name, when the values of the field are to be strings of characters other than ASCII.

The name must appear as the first parameter in a CHAR statement.
Example

Consider the string: A(1). If this string is to be used for a parameter, it must be entered as:

\[
\text{CONSTANT ( A'(1'), C )}
\]

Note: Any blanks other than the first blank following the open parenthesis will be considered part of the character string.
2.1.6 CONCODE Statements

<table>
<thead>
<tr>
<th>format</th>
<th>CONCODE ([ CONSTANT statement ] control code interpretation expression)</th>
</tr>
</thead>
<tbody>
<tr>
<td>parameters</td>
<td>(i) CONSTANT statement is a statement with the format described in Section 2.1.5.</td>
</tr>
<tr>
<td></td>
<td>(ii) control code interpretation expression has one of the following forms:</td>
</tr>
<tr>
<td></td>
<td>a) DELIM, position type</td>
</tr>
<tr>
<td></td>
<td>where position type is either the system name:</td>
</tr>
<tr>
<td></td>
<td>PRX,</td>
</tr>
<tr>
<td></td>
<td>PFX, or</td>
</tr>
<tr>
<td></td>
<td>INX.</td>
</tr>
<tr>
<td></td>
<td>b) FILLER, integer, integer [ , integer ] [ , INX ]</td>
</tr>
<tr>
<td></td>
<td>c) COLST, integer [ , integer ]</td>
</tr>
<tr>
<td></td>
<td>d) COLEND, integer [ , integer ]</td>
</tr>
<tr>
<td></td>
<td>e) NEWLINE [ , integer ] [ , INX ]</td>
</tr>
<tr>
<td></td>
<td>f) NEWPAGE [ , integer ] [ , INX ]</td>
</tr>
</tbody>
</table>

usage of the statement

The CONCODE statement is used as a parameter in other DDL statements to define control codes in the form of character strings and/or positions on media. These codes are used to locate or position values and groups of values. Control codes are needed in the
following cases:

1) in inputting values which are of variable length, a control code is needed to determine the end of the value. The control code may be a character string delimiting the value, or the position on a media of the start or end of the value.

2) in inputting values of fields or groups which repeat zero or more times, a control code is needed to determine the end of the group of values. The control code may be a character string delimiting the group of values, or the position on a media of the end of the group of values.

3) in inputting and outputting values and sets of values, control codes are needed to denote the location of the values and sets of values on the I/O media.

To specify a control code, it is necessary to include the following information:

(i) CONSTANT statement. This parameter is used to specify that the character string, which is entered as the statement's parameter, is to be a control code. The character string is to be interpreted in accordance with parameter ii of the CONCODE statement.

(ii) control code interpretation expression. This parameter describes the way control codes are to be interpreted by the DDL processor. Each option for this parameter will be discussed separately and illustrated by example before the next option is discussed.
a) DELIM, position type.

This option is used to specify that the character string given in parameter i is to be used to delimit value or a set of values. The position type must be assigned the system name:

FRX, when the control code is to precede (prefix) the value or set of values,

FRX, when the control code is to follow (postfix) the value or set of values,

INX, when the control code is to be inserted between (infix) values of repeating fields and between sets of values of repeating groups.

Example 1

Consider the set of values described in Example 1 of Section 2.1.1:

Each value in the set is a character string of variable length. The maximum length of each value is 30 characters. The values are referred to by the name 'COLLEGE' and are interpreted as college names.
If the occurrence of any of these values in a record is to be followed by a comma and a blank, then the following statement specifies a field for these values:

```
FIELD ( 'COLLEGE', V, C, 30, C; CONCODE
   ( CONSTANT ( , , C ) DELIM, PX ) )
```

where, in the CONCODE statement:

- DELIM specifies that a comma followed by a blank will be used to delimit values of 'COLLEGE'
- PX specifies that the comma followed by a blank will denote the end of a value

The following character strings can be values of the field specified in the above DDL statements:

1. UNIVERSITY OF PENNSYLVANIA,
2. PURDUE,
3. YALE,

**Example 2**

Consider two sets of values:

1. The set of values described in Example 1, above. These are values of the field 'COLLEGE'.
2. The set of values described in Example 2 of Section 2.1.1:

   Each value in the set is a character string of a fixed length of 2 characters. The values are referred to by the name 'YRS' and are to be interpreted as the number of years spent at a college.
Values from these sets are to be organized in a group called 'CCOLINF'. One value from each set is to appear; a value from the first set is to appear first and a value from the second set is to appear second. If more than one group of these values occurs, the groups are to be separated from each other by the character semicolon followed by a blank.

The following statements specify this group:

FIELD ( 'COLLEGE', V, C, 30, C;  
CONCODE ( CONSTANT ( , , C ) DELIM, PX ) )
FIELD ( 'YRS', F, C, 2, C )

GROUP ( 'CCOLINF', SPEC;  
( 'COLLEGE', M, F, 3 ),  
( 'YRS', M, F, 1 );  
CONCODE ( CONSTANT ( ; , C ) DELIM, INX ) )

where, in the CONCODE statement:

DELIM specifies that a semicolon followed by a blank will be used to delimit the set of values of 'CCOLINF'

INX specifies that the semicolon followed by a blank will separate each set of values from other sets of values of 'CCOLINF'.
In this example, the control code occurs only when 'CCOLINF' is a repeating group. Consider a group 'REPCCOLINF' in which only 'CCOLINF' occurs and repeats one or more times. This group is specified by the following statements:

FIELD ( 'COLLEGE', V, C, 30, C;

   CONCODE ( CONSTANT ( , , C ) DELIM, PTX )
)
FIELD ( 'YRS', F, C, 2, C )
GROUP ( 'CCOLINF', SPEC;
   ( 'COLLEGE', M, F, 1 ),
   ( 'YRS', M, F, 1 );
   CONCODE ( CONSTANT ( ; , C ) DELIM, INX )
) 
GROUP ( 'REPCCOLINF', SPEC;
   ( 'CCOLINF', M, V, NOLIM ) )

The following character strings satisfy these DDL statements:

   i) PURDUE, 03
   ii) PURDUE, 03; YALE, 01
   iii) UNIVERSITY OF PENNSYLVANIA, 04; PURDUE, 03; YALE, 01

   b) FILLER, integer, integer [, integer ] [, INX ]

   This option is used for outputting character strings that are not part of the retrieved records. For example, the string "x = " could be printed out before a retrieved value as follows:

   \[ X = 2.1 \]

   Such character strings are called auxiliary data.
These character strings are given as the parameter of the CONSTANT statement of parameter i.

The first integer gives the number of times the character string is to repeat consecutively.

The second integer gives the column in which the character string is to begin.

If the string consists of 1's and 0's and is to be output on cards in binary form, the third (optional) integer gives the row in which the character string is to begin. The rows on a card are considered to be numbered consecutively with the topmost row as 1.

If the auxiliary data is to be inserted between (infix) values of repeating fields or between sets of values of repeating group, the (optional) system name, INX, is used.

Example 3

Consider the case when a user, in outputting values of a field or group to a line printer, wants to fill the first 9 columns of the line, on which the values are to be printed, with the character "X".

The following statement specifies this control code:

```
CONCODE ( CONSTANT ( X , C ) FILLER, 8 , 1 )
```

where:  FILLER specifies that the string "X" will be output,
8 specifies that "X" will be repeated 8 times,
1 specifies that the string will be output beginning in column 1.
usage of parameters (continued)

c) COLST, integer [ , integer ]

This option is used for inputting and outputting values and sets of values from and to I/O media such as cards, the line printer, teletype, etc. It is used to specify that a value, or a set of values, is to begin in the column given by the integer. If the data is to be input or output in binary form, on cards, the second (optional) integer gives the row number in which the value or set of values begins.

Example 4

Consider the set of values described in Example 2 of Section 2.1.1:

Each value in the set is a character string of fixed length of 2 characters. The values are referred to by the name 'YRS' and are to be interpreted as the number of years spent at a college. Assuming a user wants to input such values on cards where the values are to begin in column 10, the following statement specifies the field and control code for these values:

```
FIELD ( 'YRS', F, C, 2, C; CONCODE ( COLST, 10 ) )
```

where, in the CONCODE statement:

COLST specifies that values of the field 'YRS' will always begin in the same column.

10 specifies that values of 'YRS' will always begin at column 10.
The card images shown below illustrate input data which satisfy this DDL statement:

```
I
2
4
6
```

usage of parameters (continued)

d) COLEND, integer [, integer ]

This option is used for inputting and outputting values and sets of values from and to I/O media such as cards, the line printer, etc.

It is used to specify that a value, or a set of values, is to end in the column given by the integer. If the data is to be input or output in binary, on cards, the second (optional) integer gives the row number in which the value or set of values ends.
Consider the set of values described in Example 1 of Section 2.1.1:

Each value in the set is a character string of variable length. The maximum length of each variable is 30 characters. The values are referred to by the name 'COLLEGE' and are interpreted as college names. Assuming a user wants to output such values on a line printer such that a value ends in column 35, the following statement specifies the field and control code for these values:

```
FIELD ( 'COLLEGE', F, C, 2, C; CONCODE ( COLEND, 35 ) )
```

where, in the CONCODE statement:

- `COLEND` specifies that values of the field 'COLLEGE' will always end in the same column.
- `35` specifies that values of 'COLLEGE' will always end in column 35.

The printer output shown below illustrates output data which would be produced from this DDL statement. Each line represents one occurrence of the field 'COLLEGE' in the output record.

```
  column 35
    
      PURDUE
      ...
      UNIV. OF MARYLAND
      UNIV. OF PENNSYLVANIA
      UNIV. OF RHODE ISLAND
      ...
```
usage of the parameters (continued)

e) NEWLINE [, integer ] [, INX ]

This option is used for inputting and outputting the values and sets of values. It is used to specify that the value, or set of values, is to occur on the next new line (or card). If several lines (or cards) are to be skipped, the optional integer is used to specify the number of lines (or cards) to be skipped. If the lines are to be inserted between values of repeating fields or between sets of values of repeating groups, the (optional) system name, INX, is used. An example of this option is given at the end of this section.

f) NEWPAGE [, integer ] [, INX ]

This option is used for outputting values and sets of values on the TTY and line-printer. It is used to specify that values, or sets of values are to be printed on a new page. If several pages are to be skipped, the optional integer is used to specify the number of pages to be skipped. If the pages are to be inserted between values of repeating fields or between sets of values of repeating group, the (optional) system name, INX, is used. An example of this option is given at the end of this section.
When a sequence of two or more CONCODE statements are used as a parameter in other DDL statements, the following rules must be followed:

1. The following system names or strings of system names may occur in CONCODE statements only once in the sequence:
   - DELIM
   - PRX
   - DELIM, PTX
   - DELIM, INX
   - COLST
   - COLEND.

2. The order in which the CONCODE statements occur in the sequence is important. CONCODE statements are processed in the order in which they appear.
   a) CONCODE statements containing the system names: DELIM, COLST and COLEND can be used to control the positioning, or relative positioning, of values or sets of values on a line (or card).
      i) If the values, or sets of values, are to appear on a new line (or card) or page, or are to be preceded by several blank lines (or cards) or pages, then these CONCODE statements must be preceded in the sequence by CONCODE statements containing the system names: NEWLINE and/or NEWPAGE.
      ii) If the values, or sets of values, are to be preceded by auxiliary data, these CONCODE statements must be preceded in the sequence by CONCODE statements containing the system name: FILLER.
iii) A CONCODE statement containing string of system names DELIM, PRX, C but without the optional first parameter may be used to mark the place of the values or set values in the sequence of CONCODE statements.

b) If no CONCODE statement containing the system names: DELIM, COLST or COEND appears in the sequence, the values or sets of values are assumed to follow the control code defined by the last CONCODE statement in the sequence.

c) The positioning of auxiliary data is controlled by CONCODE statements containing the system names: FILLER, NEWLINE and NEWPAGE.

i) If auxiliary data is to appear on a new line (or card) or page, or is to be preceded by several blank lines (or cards) or pages, then the CONCODE statements containing the system name FILLER must be preceded by CONCODE statements containing the system names: NEWLINE and NEWPAGE.

ii) If auxiliary data is to be preceded by additional auxiliary data, the CONCODE statements containing the system name FILLER for each piece of auxiliary data must occur in the sequence in the order in which the data is to appear.

Example 6

Consider 4 sets of values, from which the contents of personnel reports on employees' college educations are to be formed:
i) a set of values, each of which is a character string of a fixed length of 2 characters. The values are to be referred to by the name 'DEPT NO' and are to be interpreted as a department number.

ii) a set of values, each of which is a character string of variable length. The maximum length of each value is 15 characters. The values are to be referred to by the name 'SURNAME' and are to be interpreted as an employee's last name.

iii) a set of values, each of which is a character string of variable length. The maximum length of each value is 30 characters. The values are referred to by the name 'COLLEGE' and are interpreted as college names.

iv) a set of values, each of which is a character string of a fixed length of 2 characters. The values are to be referred to by the name 'YRS' and are to be interpreted as the number of years in attendance at a college.

A report is to be generated for each department listing every employee in the department and giving the name of the colleges and the number of years in attendance at each college for each employee.
The report is to have the following format:

a) the department number is to be given on the first page in the format illustrated below:

```
EMPLOYEE REPORT:
COLLEGE EDUCATION
DEPARTMENT 03
```

The auxiliary data "EMPLOYEE REPORT:" is to appear on line 25, starting in column 43. Following this, 2 lines are to be skipped, and the auxiliary data "COLLEGE EDUCATION" is to appear on line 28, starting in column 43. Following this, 2 more lines are to be skipped, and the auxiliary data "DEPARTMENT ", followed by the value of the department number is to appear on line 30, starting in column 43.
b) the employees' names, the names of the colleges they attended, and the number of years they were in attendance at each college are to be given on the second page in the format illustrated below:

<table>
<thead>
<tr>
<th>NAME</th>
<th>COLLEGES ATTENDED</th>
<th>YEARS</th>
</tr>
</thead>
<tbody>
<tr>
<td>JANE</td>
<td>UNIVERSITY OF PENNSYLVANIA</td>
<td>04</td>
</tr>
<tr>
<td>BILL</td>
<td>UNIVERSITY OF HOLLYWOOD</td>
<td>02</td>
</tr>
<tr>
<td>JOHN</td>
<td>UNIVERSITY OF SHOES</td>
<td>03</td>
</tr>
<tr>
<td>JANE</td>
<td>UNIVERSITY OF HOUSE ISLAND</td>
<td>04</td>
</tr>
<tr>
<td>KELLY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SHANE</td>
<td>UNIVERSITY OF CEREAL</td>
<td>04</td>
</tr>
<tr>
<td>RICHARD</td>
<td>UNIVERSITY OF COKE</td>
<td>02</td>
</tr>
<tr>
<td>VALLE</td>
<td>UNIVERSITY OF TEMPLE</td>
<td>02</td>
</tr>
<tr>
<td>WENDE</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The employees' names, the names of the colleges they attended, and the number of years they were in attendance at each college are organized as a group called 'TABLE'.

Three pieces of auxiliary data precede the values:

1) the string "NAME" is to appear on line 12, starting in column 18.

2) the string "COLLEGES ATTENDED" is to appear on line 12, starting in column 51.
iii) the string "YEARS" is to appear on line 12, starting in column 79.

A single employee's name, and the college and years he attended each college form a group 'ENTRY' within 'TABLE'. Each set of values for 'ENTRY' are separated from other sets of values by 2 lines. The employee's name starts in column 15.

Each college name and the years attended there form a group 'COLINF' within 'ENTRY'. Each set of values for 'COLINF' are separated from other sets of values by 1 line. College names end in column 70 and years start in column 80.

Each complete report will be considered to be an output record.

The following DDL statements specify the layout of these records:

```
RECORD ( 'EMPLOYEE COLLEGE REPORT', 'REPORT LAYOUT' )

GROUP ( 'REPORT LAYOUT', SPEC;
    ( 'DEPT NO', M, F, 1 ),
    ( 'TABLE', M, F, 1 )

FIELD ( 'DEPT NO', F, C, 2, C;
    CONCODE ( NEWPAGE ),
    CONCODE ( NEWLINE, 25 ),
    CONCODE ( CONSTANT ( EMPLOYEE REPORT: , C )
    FILLER, 0, 51 ),
```
CONCODE ( NEWLINE, 2,),
CONCODE ( CONSTANT ( COLLEGE EDUCATION, C )
  FILLER, 0, 51 ),
CONCODE ( NEWLINE, 2,),
CONCODE ( ( CONSTANT ( DEPARTMENT, C )
  FILLER, 0, 51 ),
CONCODE ( COLST, 62 ) )

GROUP ( 'TABLE', SPEC;
  ( 'ENTRY', M, V, NOLIM );
CONCODE ( NEWPAGE ),
CONCODE ( NEWLINE 12 ),
CONCODE ( CONSTANT ( NAME )
  FILLER, 0, 18 ),
CONCODE ( CONSTANT ( COLLEGES ATTENDED, C )
  FILLER, 0, 51 ),
CONCODE ( CONSTANT ( YEARS, C )
  FILLER, 0, 79 ) )

GROUP ( 'ENTRY', SPEC;
  ( 'SURNAME', M, F, 1 ),
  ( 'COLINF', O, V, NOLIM );
CONCODE ( NEWLINE, 2 ) )

GROUP ( 'COLINF', SPEC;
  ( 'COLLEGE', M, F, 1 ),
  ( 'YEARS', M, F, 1 );
CONCODE ( NEWLINE, 1, INX ) )
FIELD ('SURNAME', V, C, 15, C;
   CONCODE (COLST, 15))

FIELD ('COLLEGE', V, C, 30, C;
   CONCODE (COLEND, 70))

FIELD ('YEARS', F, C, 2, C;
   CONCODE (COLST, 80))
## 2.1.7 SET Statements

<table>
<thead>
<tr>
<th>Format</th>
<th>SET ( set name; member, ..., member )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameters</td>
<td>(i) set name is a user-defined name.</td>
</tr>
<tr>
<td></td>
<td>(ii) member is a CONSTANT statement.</td>
</tr>
<tr>
<td>Usage of the</td>
<td>The SET statement is used to specify a set of strings over the character set. The set is specified by listing the members extensively and assigning a name to the collection of members listed. SET statements are used together with CRITERION statements (see Section 2.1.10). To specify a set, it is necessary to include the following information in the SET statement:</td>
</tr>
<tr>
<td>Statement</td>
<td>(i) set name. This parameter is the name of the set being specified. It is used to refer to the collection of strings listed in parameter ii.</td>
</tr>
<tr>
<td></td>
<td>(ii) member. The parameter of each CONSTANT statement is to be a string in the set being specified.</td>
</tr>
<tr>
<td>Example</td>
<td>Consider a set of character strings: BURNS, JACOBS, MILLER, and SANDERSON. These are to be interpreted as authors' names. The following statement assigns the name 'AUTHORS' to this set.</td>
</tr>
<tr>
<td></td>
<td>SET ( 'AUTHOR'; CONSTANT ( BURNS, C ), CONSTANT ( JACOBS, C ), CONSTANT ( MILLER, C ), CONSTANT ( SANDERSON, C ) )</td>
</tr>
</tbody>
</table>
SET statements may also be used to define new character sets (see Section 2.1.1.1). Two SET statements are required. The first SET statement is used to list the character codes of the new character set as bit strings. The character codes are listed in their sort order. The second SET statement is used to relate the new character codes to their equivalents in another code. The characters of the other code are listed in the order that the new character codes were listed. If there are more new characters than ASCII characters, then the character string *** is used in place of the lacking character. If a character of the new code is to be equivalent eliminated in translation to the other code, then the character string ELIM is used in the second SET statement in the place of the character to be eliminated.
### 2.1.8 CRITEX Statement

<table>
<thead>
<tr>
<th>Format</th>
<th>CRITEX ( criterion expression name, criterion expression, )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameters</td>
<td></td>
</tr>
<tr>
<td>(i)</td>
<td>criterion expression name is an unindexed user-defined name.</td>
</tr>
<tr>
<td>(ii)</td>
<td>criterion expression is a string of the form:</td>
</tr>
<tr>
<td>a)</td>
<td>( data name ) arithmetic relation</td>
</tr>
<tr>
<td></td>
<td>( arithmetic operation expression )</td>
</tr>
<tr>
<td>where:</td>
<td></td>
</tr>
<tr>
<td>1)</td>
<td>data name is a reference name</td>
</tr>
<tr>
<td>2)</td>
<td>arithmetic relation is either the string:</td>
</tr>
<tr>
<td></td>
<td>EQ, or =</td>
</tr>
<tr>
<td></td>
<td>NQ,</td>
</tr>
<tr>
<td></td>
<td>LT, or &lt;</td>
</tr>
<tr>
<td></td>
<td>LE,</td>
</tr>
<tr>
<td></td>
<td>GT, or &gt;, or</td>
</tr>
<tr>
<td></td>
<td>GE.</td>
</tr>
<tr>
<td>3)</td>
<td>arithmetic operation expression is a string of the form:</td>
</tr>
<tr>
<td>3i)</td>
<td>data name</td>
</tr>
<tr>
<td></td>
<td>where data name is defined as above for parameter (ii), a), 1).</td>
</tr>
<tr>
<td>3ii)</td>
<td>CONSTANT statement</td>
</tr>
<tr>
<td></td>
<td>where the CONSTANT statement has the format described in Section 2.1.5.</td>
</tr>
</tbody>
</table>
3ii) ( arithmetic operation expression )
    arithmetic operation ( data name )
    where arithmetic operation is either
    the string:
    +, 
    -, 
    x, or 
    /,
    and data name is defined as
    above for parameter (ii), a), 1)

3iv) ( arithmetic operation expression )
    arithmetic operation ( CONSTANT
    statement )
    where arithmetic operation is
    defined as above in c, and the CONSTANT
    statement has the format described
    in Section 2.1.5.

b) ( data name ) MEM ( set name )

where:

1) data name is a reference name

2) set name is an unindexed user-defined name.
c) NOT ( criterion expression form )
where criterion expression form is either an unindexed user-defined name, or a criterion expression of the form described in a) - e) of parameter ii.

d) ( criterion expression form ) AND ... AND ( criterion expression form )
where criterion expression form is defined as in c) above.

e) ( criterion expression form ) OR ... OR ( criterion expression form )
where criterion expression form is defined as in c) above.

Values and sets of values are rejected during input, if they differ from the format described for them. These format descriptions are conditions on the values and sets of values which must be satisfied for the values and sets of values to be accepted for input. The user may describe additional conditions (called criteria) on values and sets of values. These criteria may take the form of arithmetic relations, set relations and logical combinations of these. Criteria are given by expressions called criterion expressions. To specify a criterion expression it is necessary to include the following information in the CRITEX statement;
usage of the parameters

(1) criterion expression name. This parameter is the name of the criterion expression being specified and is used to refer to it.

(ii) criterion expression. This parameter describes a criterion for values. Each of the forms this parameter may take will be discussed separately and illustrated by examples before the next form is discussed.

a) (data name) arithmetic relation

\[ (\text{arithmetic operation expression}) \]

This form is used to specify that the value of a field referred to by the reference name is:

- \( \text{EQ}, \text{or } = \), equal to,
- \( \text{NQ}, \) not equal to,
- \( \text{LT}, \text{or } < \), less than,
- \( \text{LE}, \) less than or equal to,
- \( \text{GT}, \text{or } > \), greater than, or
- \( \text{GE}, \) greater than or equal to,

the value of the arithmetic operation expression.

If the arithmetic operation expression has the form:

i) data name

where data name refers to a field, then the value of the arithmetic operation expression is the value of the field referred to.
ii) CONSTANT statement
then the value of the arithmetic operation expression is the string declared in the CONSTANT statement.

iii) ( arithmetic operation expression )
arithmetic operation ( data name )
where data name refers to a field,
then the value of the arithmetic operation expression, where the arithmetic operation is:
+ , is the sum of the value of the internal arithmetic operation expression and the value of the field referred to,
- , is the difference between the value of the internal arithmetic operation expression and the value of the field referred to,
\times , is the value of the internal arithmetic operation expression multiplied by the value of the field referred to,
\div , is the value of the internal arithmetic operation expression divided by the value of the field referred to.
iv) \( (\text{arithmetic operation expression}) \)

\[
\text{arithmetic operation (CONSTANT statement)}
\]

where reference name refers to a field, and the string declared by the \text{CONSTANT} statement is a number, then the value of the arithmetic operation, where the arithmetic operation is:

\[\text{+} , \text{ is the sum of the value of the internal arithmetic operation expression and the numeric value of the string declared in the \text{CONSTANT} statement,}\]

\[-, \text{ is the difference between the value of the internal arithmetic operation expression and the numeric value of the string declared in the \text{CONSTANT} statement,}\]

\[x, \text{ is the value of the internal arithmetic operation expression multiplied by the value of the string declared in the \text{CONSTANT} statement,}\]

\[/, \text{ is the value of the internal arithmetic operation expression divided by the value of the string declared in the \text{CONSTANT} statement.}\]
Example 1

Consider a field 'YEARS' whose values are interpreted as the number of years a person attended a particular college. A user may test to see if a particular value of 'YEARS' is equal to the value of another field, say 'X'. The following statements specifies this criterion expression and assigns it the name 'TEST1':

\[
\text{CRITEX} \left( \text{'TEST1'}, \left( \text{'YEARS'} \right) \text{EQ} \left( \text{'X'} \right) \right)
\]

Example 2

Consider the field 'YEARS' described in Example 1, above. A user may test to see if a particular value of 'YEARS' is less than the constant 4. The following statement specifies this criterion expression and assigns it the name 'TEST2':

\[
\text{CRITEX} \left( \text{'TEST2'}, \left( \text{'YEARS'} \right) \text{LT} \left( \text{CONSTANT} \left( 4, \text{C} \right) \right) \right)
\]

Example 3

Consider the field 'YEARS' described in Example 1, above; and two fields 'DATESTART' and 'DATEEND' whose values are interpreted, respectively, as the year in which a person started a college and the year in which the person left the college. A user may test to see if a particular value of 'YEARS' equals the difference between the values of 'DATEEND' and 'DATESTART'. The following statement specifies this criterion expression and assigns it the name 'TEST3':

\[
\text{CRITEX} \left( \text{'TEST3'}, \left( \text{'YEARS'} \right) \text{EQ} \left( \text{DATEEND} \text{MINUS} \text{DATESTART} \right) \right)
\]
usage of the parameters (continued)

Example 4

Consider a field 'AUTH' whose values are interpreted as the last names of authors; and a set of character strings: BURNS, JACOBS, MILLER, and SANDERSON, to which the name 'AUTHORS' has been assigned (see the example in Section 2.1.7). A user may test to see if a particular value of 'AUTH' is a member of the set 'AUTHORS'. The following statement specifies this criterion expression and assigns it the name 'TEST4':

```
CRITEX ( 'TEST4', ( 'AUTH') MEM ( 'AUTHORS'))
```

usage of the parameters (continued)

b) \((\text{data name}) \text{MEM} (\text{set name})\)

This form is used to specify that the value of a field referred to by the data name is a member of the set named by set name. Set name must appear as the first parameter in a SET statement.

```
CRITEX ( 'TEST3', ( 'YEARS')
EQ (( 'DATEEND') - ( 'DATESTART') ) )
```

c) \(\text{NOT} (\text{criterion expression form})\)

This form is used to specify that the negation of a criterion expression is to be tested, where the criterion expression is given directly or is only named. If the criterion expression is named, the name must appear as the first parameter in another CRITEX statement.
Consider the criterion expression described in Example 4, above: Is a particular value of the field 'AUTH' a member of the set 'AUTHORS'. The criterion expression which is the negation of this criterion expression is: Is a particular value of the field 'AUTH' not a member of the set 'AUTHORS'. It may be specified in two ways. These are given below. The first way is assigned the name 'TEST5.1' and the second way is assigned the name 'TEST5.2':

i) \( \text{CRITEX ( 'TEST5.1', NOT ( 'TEST4' ) )} \)

where 'TEST4' is specified in Example 4, above.

ii) \( \text{CRITEX ( 'TEST5.2', NOT ( ( 'AUTH' ) MEM ( 'AUTHORS' ) ) )} \)

d) ( criterion expression form ) AND ... AND ( criterion expression form )

This form is used to specify that the logical conjunction of two or more criterion expressions is to be tested, where the criterion expressions are given directly or are only named. If the criterion expressions are named, the names must appear as the first parameters in other CRITEX statements.

Consider the criterion expressions described in Examples 2 and 3, above:
i) Is a particular value of a field 'YEARS'
    less than the constant 4, and

ii) Is a particular value of the field 'YEARS'
    equal to the difference between the values of
    the fields 'DATEEND' and 'DATASTART'.

The following statement specifies the criterion
expression which is the conjunction of these criterion
expressions and assigns it the name 'TEST6':

\[
\text{CRITEX ('TEST6', ('TEST2') AND ('TEST3'))}
\]

where 'TEST2' and 'TEST3' are specified in Examples 2 and
3, above.

The same test may be specified by giving directly
the criterion expressions for 'TEST2' and/or 'TEST3'.

e) ( criterion expression form ) OR ... OR
   ( criterion expression form )

This form is used to specify that the
logical disjunction of two or more criterion
expressions is to be tested, where the criterion
expressions are given directly or are only named.
If the criterion expressions are named, the
names must appear as the first parameters in
other CRITEX statements.

Consider the criterion expressions described in
Examples 1 and 2, above:

1) Is a particular value of the field 'YEARS'
   equal to the value of the field 'X', and
ii) Is a particular value of the field 'YEARS' less than the constant 4.

The following statement specifies the criterion expression which is the disjunction of these criterion expressions and assigns it the name 'TEST7':

\[
\text{CRITEX ('TEST7', ( 'TEST1' ) OR ( 'TEST2' ) )}
\]

where 'TEST1' and 'TEST2' are specified in Examples 1 and 2, above.

The same test may be specified by giving directly the criterion expressions for 'TEST1' and/or 'TEST2'.

The parameter statements described in Section 2.1.4 can be used in place of data names in parameter ii. Thus, for example, conditions such as the length of values of fields, or the existence of values for optional fields or groups can be tested.

Consider an optional field 'DRAFT STATUS' whose values are interpreted as a person's draft status. A user may test to see if a value for this field occurs. The following statement specifies this criterion expression and assigns it the name 'TEST6':

\[
\text{CRITEX ('TEST6', ( EXIST ('DRAFT STATUS') )})
\]

\[
\text{EQ ( CONSTANT ( 1, C ) )}
\]
2.1.9 Alternate Action Statements

Alternate action statements are statements that are used as parameters in CRITERION statements (see Section 2.1.10) to specify which actions should be taken when values or sets of values fail to satisfy a criterion expression. For example, if during input a value fails to satisfy a criterion expression, the user may have the value replaced with another value, or he may have values and an error message output.
### 2.1.9.1 RPLVAL Statements

<table>
<thead>
<tr>
<th>format</th>
<th>RPLVAL ( data name, replacement value )</th>
</tr>
</thead>
<tbody>
<tr>
<td>parameters</td>
<td>(i) data name is a reference name</td>
</tr>
<tr>
<td></td>
<td>(ii) replacement value is either a reference name, which has the form described for parameter i, above, or it is a CONSTANT statement.</td>
</tr>
</tbody>
</table>

The RPLVAL statement is used as a parameter in a CRITERION statement whenever the following action is to be taken upon the failure of a value or set of values to satisfy a criterion. This action is that the values of a field or group are to be replaced by other values.

| usage of the statement | (i) reference name. This parameter is used to name the field or group whose values are to be replaced. |
|                       | (ii) replacement value. This parameter is used to name the field or group whose values are to be used as replacement values, or to give the replacement value directly in a CONSTANT statement. The parameter is assigned: |
|                       | a) a replacement name which refers to a field, when the first parameter names a field whose value is to be replaced by the value of the field referred to. |
b) a replacement name which refers to a group, when the first parameter names a group whose values are to be replaced by the values of the group referred to.

c) a CONSTANT statement when the first parameter names a field whose value is to be replaced by the character string specified.

Consider a field 'AUTH' whose values are interpreted as the last names of authors. The following statements specify that the value of 'AUTH' is to be replaced:
a) by the value of a field 'X', and b) by the string "ANON":

\[
\begin{align*}
\text{a) } & \text{RPLVAL ('AUTH', 'X')} \\
\text{b) } & \text{RPLVAL ('AUTH', CONSTANT ('ANON', C))}
\end{align*}
\]

The parameter statements described in Section 2.1.4 can be used for parameter ii. Thus, for example, the value of a field can be replaced with the length of another field.
2.1.9.2 CRITMESSAGE Statements

**format**

CRITMESSAGE ( device name, criterion message )

**parameters**

- (i) device name is either the system name:
  - TTY,
  - PRINTER, or
  - CARD.

- (ii) criterion message is a string of the form:
  - [ data name, ] CONCODE statement, ... ,
  - CONCODE statement

  where:
  - a) data name is a reference name.
  - b) CONCODE statement is a statement with the format described in Section 2.1.5.

**usage of the statement**

The CRITMESSAGE statement is used as a parameter in a CRITERION statement whenever the following action is to be taken when a value or set of values fails to satisfy a criterion. The action is that a message, determined by the user, be output.

**usage of the parameters**

- (i) device name. This parameter is used to signify the device on which the message is to be output. The parameter must be assigned the system name:
  - TTY, when the teletypewriter is to be used to output the message,
  - PRINTER, when the line printer is to be used, or
  - CARD, when the message is to be punched on cards.
(ii) criterion message. This parameter is used
to specify the message to be output. It may
include values or sets of values from the record
being validated, or it may simply be a string
determined by the user.

a) data name. This optional parameter is used
to indicate that a value or set of values
is to be output as part of the message. The
name must refer to the field or group whose
values are to be output.

b) CONCODE statements. These parameters are
used to specify the layout of the message.
If a value or set of values are to be output,
the CONCODE statements may be used to position
them on the output media (this is discussed
in detail in Section 2.1.6).

If no values are to be output, the CONCODE
statement with the option FILLER may be used
to output character strings.

Example 1

Consider a field 'AUTH' whose values are interpreted
as the last names of authors. The following statement
specifies that the value of 'AUTH' is to be output with
the string "FAILS TO SATISFY THE CRITERION" on the tele-
typewriter:
Example 2

The following statement specifies that the string "CRITERION X" is to be output on the printer:

```
CRITMESSAGE ( PRINTER,
    CONCODE ( CONSTANT ( CRITERION X, C )
               FILLER, 0, 40 ) )
```
### 2.1.10 CRITERION Statements

| format                  | CRITERION ( criterion name, criterion expression name
|                        |   ; alternate action statement; ... ; alternate
<table>
<thead>
<tr>
<th></th>
<th>action statement )</th>
</tr>
</thead>
<tbody>
<tr>
<td>parameters</td>
<td>(i) criterion name is an unindexed user-defined name.</td>
</tr>
<tr>
<td></td>
<td>(ii) criterion expression name is an unindexed user-defined name.</td>
</tr>
<tr>
<td></td>
<td>(iii) alternate action statement is a statement with a format described in Section 2.1.9.</td>
</tr>
</tbody>
</table>

Criteria on values and sets of values are used to determine whether or not those values and sets of values are to be accepted for input, output or storage, and what actions are to be taken if they are rejected. Thus, to specify a criterion it is necessary to include the following information in the CRITERION statement.

| usage of the statement | (i) criterion name. This parameter is the name of the criterion being specified and is used to refer to it. |
|                       | (ii) criterion expression name. This parameter is the name of the criterion expression which describes the conditions on the values or sets of values to be tested. The name must appear as the first parameter in a CRITEEX statement. |
(iii) alternate action statements. These optional parameters are used to specify what actions are to be taken when values fail to satisfy the criterion expression named by parameter ii. The options which may be chosen are discussed in Section 2.1.9. If no action is specified the values are rejected.

Consider a field 'YEARS' whose values are interpreted as the number of years a person attended a particular college. A user may test to see if a particular value of 'YEARS' is less than the constant 4. If the value does not satisfy the criterion the user may have the string "FAILS CRITERION" printed out on the teletype starting in column 15. The following statements specify this criterion and assign it the name "CRITERION":

\[
\text{CRITEX ('TEST', ('YEARS') LT (CONSTANT (4, C)))}
\]

\[
\text{CRITERION ('CRITERION', 'TEST', CRITMESSAGE (TTY, CONCODE (CONSTANT (FAILS CRITERION, C) FILLER, 0, 15)))}
\]
2.1.11 Device Statements

Device statements are statements that are used to specify the device on which record occurrences are to be input, output, and stored and to specify the location of the record occurrences on the device media.

The same two statements are used to describe the location of record occurrences on device media whatever the device. These are the BBLOCK and BLOCK statements.

In addition there is one statement for each supported device, which relates the location of record occurrences as specified in the BBLOCK and BLOCK statements to the specific devices.
2.1.11.1 BBLOCK Statement

| format | BBLOCK ( basic block name; uniformity, length; |
|        | record distribution, record count uniformity, |
|        | record count, basic block count, record order; |
|        | ( list ), ... , ( list ) [ ; HDR, header ] ... |
|        | [ ; HDR, header ] [ ; TLR, trailer ] ... |
|        | [ ; TLR, trailer ] ) |

| parameters | (i) basic block name is an unindexed user-defined name. |
|            | (ii) uniformity is either the system name: |
|            |   FIXED (or simply F), or |
|            |   VARIABLE (or simply V). |
|            | (iii) length is either the string: |
|            |   n, where n is an integer, or the system name NOLIM. |
|            | (iv) record distribution name is either the system name: |
|            |   WHOLE, or |
|            |   SPLIT. |
|            | (v) record count uniformity is either the system name: |
|            |   FIXED (or simply F), or |
|            |   VARIABLE (or simply V). |
|            | (vi) record count is either the string: |
|            |   n, where n is an integer, or the system name NOLIM.
(vii) basic block count is either the string:

m, where m is an integer, or the system name NOLIM.

(viii) record order is either the system name:

SPEC, or

NOORD.

(ix) list is a string of parameters with the following format:

record name, optionality, repetition number

where:

a) record name is an unindexed user-defined name;

b) optionality is either the system name:

MANDATORY (or simply M), or

OPTIONAL (or simply O);

c) repetition number is either the string:

n, where n is an integer, or the system name NOLIM.

(x) the optional parameter, header, is either an unindexed user-defined name, or a CONSTANT statement, with the format described in Section 2.1.5.

(xi) the optional parameter, trailer, is either an unindexed user-defined name, or a CONSTANT statement, with the format described in Section 2.1.5.
The BBLOCK statement is used to define the basic physical block of storage on a device media, and to describe the location of record occurrences in the basic block. The statements describing the individual devices specify whether the basic block is to be interpreted as a physical block on a tape or disk drive, or as a deck of cards, etc. To specify a basic physical block of storage it is necessary to include the following information in the BBLOCK statement:

(i) basic block name. This parameter is the name of the basic block being specified.

(ii) uniformity. This parameter is used to specify whether the length of the basic block is the same each time it occurs. If the basic block is to occur only once then the parameter is used to specify whether the length of the block is known. The parameter is assigned the system name:

FIXED (or F) when the length of each occurrence of the basic block is the same, or when the length of the basic block is given if it is to only occur once, and

VARIABLE (or V) when the length of each occurrence of the basic block is to vary, or when the length of the basic block is to occur only once and is not known in advance.
Note: The meaning of the word length for a basic block depends on the media on which it occurs. Thus for cards, length is the number of cards in a deck, or for tape it is the number of bytes in a physical block. This will be discussed further in the sections dealing with the individual devices (Sections 2.1.11.3 - 2.1.11.10).

(iii) length. This parameter is used to specify the length of the basic block. The parameter must be assigned the string:

n when the lengths of occurrences of a basic block are fixed at n, or when the lengths of occurrences vary but do not exceed a maximum of n.

NOLIM when the lengths of occurrences of basic blocks vary and are not limited.

(iv) record distribution name. This parameter is used to specify whether the values of a record occurrence are to be split between two occurrences of a basic block if the length of the record occurrence exceed the length of a basic block occurrence. The parameter is assigned the system name:

WHOLE, when a record occurrence is not to be split, and

SPLIT, when a record occurrence may be split.
(v) record count uniformity. This parameter is to specify whether the number of record occurrences in the basic block is to be fixed or variable. The parameter must be assigned the string:

FIXED (or F) when the number of record occurrences is to be fixed, and

VARIABLE (or V) when the number of record occurrences is to vary.

(vi) record count, and

(vii) basic block count. These parameters are used to specify the number of record occurrences, specified by the record count, that are to be placed in the number of occurrences of the basic block, specified by the basic block count. The two parameters are to be assigned the strings:

n, m when n record occurrences are to be placed in m occurrences of the basic block, or a maximum of n record occurrences are to be placed in m occurrences of the basic block.

1, NOLIM when a single record occurrence of variable length exceeds the length of a basic block, and each record occurrence is to begin in a new occurrence of the basic block.

NOLIM, 1 when there is to be no limit on the number of record occurrences to be placed in a single occurrence of a basic block.
(viii) record order. This parameter is used to specify the order in which occurrences of different records are to occur in a basic block. The parameter is assigned the system name:

SPEC, when the record occurrences are to be located in the basic block occurrence in the order in which they are named in parameter (ix), and NOORD, when record occurrences may be located in any order in a basic block occurrence.

(ix) list. Each list describes the location of record occurrences for a particular record in the basic block being specified. The list parameters must be assigned strings as follows:

a) record name. This parameter is the name of a record whose occurrences may be located in the basic block being specified. The name must appear as the first parameter in a RECORD statement.

b) optionality. This parameter is used to specify whether occurrences of the record named by list parameter a) are mandatory or optional. The parameter must be assigned the system name:

MANDATORY (or M) when the record occurrence is mandatory, and

OPTIONAL (or O) when the record occurrence is optional.
c) repetition number. This parameter is used to specify the number of times occurrences of the record named by list parameter a) are to occur in the basic block. The parameter must be assigned the string:

- n when the record is to be repeated n times,
- 0 when the record is not to be repeated, and
- NOLIM when the record is to be repeated an unknown number of times.

d) the optional parameter, criterion name.

This parameter refers to a criterion which is defined on values in record occurrences. If the criterion is not satisfied, occurrences of the record named by parameter a) do not appear in the basic block. The criterion name must appear as the first parameter in a CRITERION statement.

(x) header. This optional parameter is used to specify control information which occurs at the head of a basic block. The statements describing the individual devices specify whether the header is to be interpreted as a start card for a card deck, or a label for a tape, etc. The parameter is assigned
a) a record name, when the contents of the header depend on values from record occurrences or other basic blocks. The record name must appear as the first parameter in a RECORD statement.

b) a CONSTANT statement, when the contents of the header can be expressed as a character or bit string.

(xi) trailer. This optional parameter is used to specify control information which occurs at the tail of a basic block. It is specified in the same way as the header parameter, described as parameter (x), above.

Examples of the use of the BBLOCK statement will be given in the section discussing the individual devices.
2.1.11.2 BLOCK Statement

format

```
BLOCK ( block name; order; ( list ), ... , ( list )
[ ; HDR, header ] ... [ ; HDR, header ]
[ ; TLR, trailer ] ... [ ; TLR, trailer ] )
```

parameters

(i) block name is an unindexed user-defined name.
(ii) order is either the system name:

   NOORD, or

   SPEC.

(iii) list is a string of parameters of the form:

   b/block name, optionality, repetition number

   [ , criterion name ]

where:

a) b/block name is an unindexed user-defined name;

b) optionality is either the system name:

   MANDATORY (or simply M), or

   OPTIONAL (or simply O);

c) repetition number is either the string:

   n, where n is an integer, or the system name

   NOLIM.

d) the optional parameter, criterion name,

   is an unindexed user-defined name.

(iv) the optional parameter, header, is either an

    unindexed user-defined name, or a CONSTANT

    statement with the format described in Section

    2.1.5.
The BLOCK statement is used to specify a block. A block is an organization of occurrences of different basic blocks or blocks and basic storage. The statements describing the individual devices specify whether the block is to be interpreted as a track or a cylinder of a disk, or as a page on a printer, etc.

To specify a block, it is necessary to include the following information:

1. **Block name.** This parameter is the name of the block being specified.
2. **Order.** This parameter specifies the sequence in which blocks and basic blocks are to occur in any order in the block being specified.
3. **List.** Each list describes the occurrence of a basic block or block in the block being specified.

The **BLOCK statement** has the following format:

```
BLOCK (block-name, order, list)
```

The order parameter must be specified for the system name:

- **NOORD,** when the blocks or basic blocks may occur in any order in the block being specified.
- **SPECIAL,** when the blocks or basic blocks must occur in any order in the block being specified.
- **ORDER,** when the blocks or basic blocks must occur in the order in which they are named.

The list parameter must be assigned the system name:

- **NOORD,** when the blocks or basic blocks may occur in any order in the block being specified.
- **SPECIAL,** when the blocks or basic blocks must occur in the order in which they are named.
- **ORDER,** when the blocks or basic blocks must occur in the order in which they are named.

(FCC)
follows:

a) block name. This parameter is the name of a basic block or block. The name must appear as the first parameter in a BBLOCK statement if it refers to a basic block. The name must appear as the first parameter in a BLOCK statement if it refers to a block.

b) optionality. This parameter is used to specify whether the occurrence of the basic block or block named by list parameter a) is mandatory or optional. The parameter must be assigned the system name:

MANDATORY (or M) when the occurrence is mandatory, and

OPTIONAL (or 0) when the occurrence is not mandatory.

c) repetition number. This parameter is used to specify the number of times basic block or block named by list parameter a) occurs in the block being specified. The parameter must be assigned the string:

n when the block or basic block is to be repeated n times,

0 when the block or basic block is not to be repeated, and

NOLIM when the block or basic block is to be repeated an unknown number of times.
d) criterion name. This parameter refers to a criterion which is defined on the behavior of other blocks or basic blocks in terms of their existence, length or repetition number. If the criterion is not satisfied, the block or basic block named by list parameter a) does not occur in the block being specified. The criterion name must appear as the first parameter in a CRITERION statement.

(iv) header, and

(v) trailer. These optional parameters are used to specify control information which is to occur at the head and tail of the block being specified. They specified as described for parameters (x) and (xi) of the BBLOCK statement in Section 2.1.11.1.

Examples of the use of the BLOCK statement will be given in the section discussing the individual devices that follow.
Since a basic block or block 'B' may be contained in more than one block, say in block 'B1', 'B2', 'B3', then the occurrences of 'B' in 'B3' is referred to by the name: 'B' OF 'B3'. As many of the phrases: "OF group name " may be specified as necessary to distinguish between occurrences of a block or basic block. Such names are also considered to be reference names.

Similarly, occurrences of a record 'R' may occur in more than one basic block, say in basic blocks 'BB1', 'BB2' and 'BB3'. The occurrence of 'R' in 'BB2' is referred to by the name: 'R' OF 'BB2'. Such names are also considered to be reference names.

The parameter statements REP and EXIST may be applied to blocks and basic blocks.
### 2.1.11.3 CARDIN Statement

**format**

\[
\text{CARDIN ( card specification name, bblock name} \\
\quad [ ; \text{association list name } ] )
\]

**parameters**

(i) card specification name is an unindexed user-defined name.

(ii) bblock name is an unindexed user-defined name.

(iii) the optional parameter, association list name, is an unindexed user-defined name.

**usage of the statement**

The CARDIN statement is used to specify that the card reader is to be used for inputting data. To specify this, it is necessary to include the following information in the CARDIN statement:

**usage of the parameters**

(i) card specification name. This parameter is the name of the device specification being described.

(ii) bblock name. This parameter refers to the specification of the location of record occurrences in a card deck. The card deck is considered to be the basic physical block of storage for the card reader. The bblock name must appear as the first parameter in a BBLOCK statement with the following parameters:

a) parameter (ii) of the BBLOCK statement is assigned the value:

\[
\text{FIXED (or F) when the number of cards}
\]

in the deck is known, and
VARIABLE (or V) when the number of cards in the deck is not known.

b) parameter (iii) of the BBLOCK statement is assigned the string:
   n when there is to be either a maximum of n cards in the deck or exactly n cards.
   NOLIM, otherwise.

c) parameter (iv) of the BBLOCX statement is assigned the string WHOLE.

d) parameters x and xi are interpreted as header and trailer cards. Each string of the form:
   [ ; HDR, header ] or
   [ ; TLR, trailer ] is interpreted as a single card.

(iii) association list name. This optional parameter is used when the header and/or trailer parameters in the BBLOCK statement referred to by parameter (ii) refer to records. The name must appear as the first parameter in an ASSOCIATE statement (see Section 2.3.1.1).

Example

The following statements specify that a card deck is to be input with one start card and one end card.
The deck is to contain an unknown number of occurrences of the record 'X'. The start card is to contain the string: START OF DATA starting in column 1. The end card is to contain the string: END OF DATA starting in column 1.
CARDIN ( 'INPUT CARDS', 'CARD DECK' )

BBLOCK ( 'CARD DECK'; V, NOLIM; WHOLE, V,
        NOLIM, 1; SPEC; ( 'X', M, NOLIM );
        HDR, CONSTANT ( START OF DATA, C );
        TLR, CONSTANT ( END OF DATA, C ) )
2.1.11.4 CARDOUT Statement

| format | CARDOUT ( card specification name, bblock name
|        | [ ; association list name ] ) |
| parameters | (i) card specification name is an unindexed user-defined name. |
|           | (ii) bblock name is an unindexed user-defined name. |
|           | (iii) the optional parameter, association list name, is an unindexed user-defined name. |

The CARDOUT statement is used to specify that the card punch is to be used for outputting data. To specify this, it is necessary to include the following information in the CARDOUT statement:

<table>
<thead>
<tr>
<th>usage of the statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>(i) - (iii) The parameters are the same as those specified for the CARDIN Statement of Section 2.1.11.3.</td>
</tr>
</tbody>
</table>
2.11.5 TAPEIN Statement

format

TAPEIN ( tape specification name, file block name;
        physical block name, ..., physical block name
        [ ; association list name ] )

parameters

(i) tape specification name is an unindexed user-defined name.
(ii) file block name is an unindexed user-defined name.
(iii) physical block is a reference name.
(iv) the optional parameter, association list name, is an unindexed user-defined name.

usage of the statement

The TAPEIN statement is used to specify that the tape device is to be used for inputting data. To specify this, it is necessary to include the following information in the TAPEIN statement:

usage of the parameters

(i) tape specification name. This parameter is the name of the device specification being described.
(ii) file block name. This parameter refers to the specification of the organization of physical blocks on a tape. The name must appear as the first parameter in a BLOCK statement with the following parameters:
a) parameter (ii) of the BLOCK statement is assigned the system name:

NOORD when there is more than one kind of physical block on the tape and the various kinds may occur in any order, and SPEC, when there is only one kind of physical block on the tape, or when there is more than one kind and the various kinds occur in the order specified by parameter (iii) of the BLOCK statement.

b) the organization of physical blocks on a tape may be specified by as many levels of BLOCK statements as are needed. The first list parameter of parameter (iii) refers either to other BLOCK statements or to BBLOCK statements.

c) parameters (iv) and (v) of the BLOCK statement are interpreted as header and trailer labels or tape marks. Each string of the form:

[; HDR, header ] or

[; TLR, trailer ] is interpreted as a single label or tape mark.

(iii) physical block name. This parameter refers to the specification of each physical block on the tape. There must be one physical block name for each kind of physical block that may occur in the tape. The physical block names must appear as the first parameters in BBLOCK statements which specify
the location of record occurrences in the physical blocks. The BBLOCK statements must have the following parameters:

a) parameter (ii) of the BBLOCK statement is assigned the system name:
   FIXED (or F) when the number of bytes in the physical block is to be fixed, and VARIABLE (or V) when the number of bytes in the physical block is to vary from one occurrence of the physical block to another.

b) parameters $x$ and $x_1$ are interpreted as header and trailer bytes in the physical block.

(iv) association list name. This optional parameter is used when the header and/or trailer parameter in the BBLOCK statement referred to by parameter (ii) refer to records. The name must appear as the first parameter in an ASSOCIATE statement (see Section 2.3.1.1).

Example

The following statements specify that data is to be input from a tape containing a file which is stored in fixed length physical blocks of 2064 bytes. Each physical block is to contain 25 records which are never split between physical blocks. The file is to be preceded by two header labels: 'STANDARD VOLUME LABEL' and 'STANDARD FILE LABEL', and a tape-
mark 'TM'. The file is to be trailed by a single label: 'STANDARD FILE TLR LABEL' and two tapemarks 'TM'. Each physical block is to be headed by a 16 byte header called 'KEY'. There are an unknown number of record occurrences 'BILL-RECORD' in the file.

TAPEIN ( 'BILL-TAPEIN'; 'BILL-TAPE', 'BILL-BLK' )

BLOCK ( 'BILL-TAPE', SPEC;

( 'BILL-BLK', M, NOLIM );
HDR, 'STANDARD VOLUME LABEL';
HDR, 'STANDARD FILE LABEL';
HDR, 'TM';
TLR, 'STANDARD FILE TLR LABEL';
TLR, 'TM';
TLR, 'TM' )

BBLOCK ( 'BILL-BLK'; F, 2064; WHOLE, F, 25,
SPEC; ( 'BILL-RECORD', M, NOLIM );
HDR, 'KEY' )
2.1.11.6 TAPEOUT Statement

<table>
<thead>
<tr>
<th>format</th>
<th>TAPEOUT (tape specification name, file block name; physical block name, ..., physical block name [ ; association list name ] )</th>
</tr>
</thead>
<tbody>
<tr>
<td>parameters</td>
<td>(i) tape specification name is an unindexed user-defined name.</td>
</tr>
<tr>
<td></td>
<td>(ii) file block name is an unindexed user-defined name.</td>
</tr>
<tr>
<td></td>
<td>(iii) physical block is a reference name.</td>
</tr>
<tr>
<td></td>
<td>(iv) the optional parameter, association list name, is an unindexed user-defined name.</td>
</tr>
</tbody>
</table>

The TAPEOUT statement is used to specify that the tape device is to be used for outputting data. To specify this, it is necessary to include the following information in the TAPEOUT statement:

(i) - (iv) The parameters are the same as those specified for the TAPEIN statement of Section 2.1.11.5.
2.1.11.7 DISKIN Statement

format

DISKIN ( disk specification name, file block name;
cylinder block name, ... , cylinder block name;
track block name, ... , track block name;
sentence block name, ... , sentence block name
[ ; association list name ])

parameters

(i) disk specification name is an unindexed user-defined name.
(ii) file block name is an unindexed user-defined name.
(iii) cylinder block name is a reference name.
(iv) track block name is a reference name.
(v) sentence block name is a reference name.
(vi) the optional parameter, association list name, is an unindexed user-defined name.

usage of the statement

The DISKIN statement is used to specify that input data is to be found on disk. To specify this, it is necessary to include the following information in the DISKIN statement:

usage of the parameters

(i) disk specification name. This parameter is the name of the device specification being described.
(ii) file block name. This parameter refers to the specification of the organization of the cylinders on which a file is to be stored. The name must appear as the first parameter in a BLOCK statement with the following parameters:

a) parameter (ii) of the BLOCK statement is assigned the system name:

   NOORD, when there is more than one kind of cylinder format in the file, and the various kinds of cylinder formats may occur in any order, and

   SPEC, when there is only one kind of cylinder format in the file, or when there is more than one and the various kinds occur in the order in which they are named in parameter (iii) of the BLOCK statement.

b) the organization of cylinders in a disk may be specified by as many levels of BLOCK statements as are needed. The first list parameter of parameter (iii) refers either to a BLOCK statement describing groupings of cylinder formats or directly to a BLOCK statement describing a cylinder format.
c) parameters (iv) and (v) of the BLOCK statement are not used.

(iii) cylinder block name. This parameter refers to the specification of the organization of tracks on cylinders. There must be one cylinder block name for each kind of cylinder format that may occur in a file. The cylinder block names must appear as the first parameter in a BLOCK statement with the following parameters:

a) parameter (ii) of the BLOCK statement is assigned the system name:

   NOORD, when there is more than one kind of track format in the cylinder, and the various kinds of track format may occur in any order, and

   SPEC, when there is only one kind of track format in the cylinder, or when there is more than one and the various kinds occur in the order in which they are named in parameter (iii) of the BLOCK statement.

b) the organization of tracks in a cylinder may be specified by as many levels of BLOCK statements as are needed. The first list parameter of parameter (iii) refers either to a BLOCK statement describing groupings of tracks, or
directly to a BLOCK or BBLOCK statement describing a track format.

c) parameters (iv) and (v) of the BLOCK statement are interpreted as header and trailer tracks for a cylinder.

(iv) track block name. This parameter refers to the specification of the organization of sentences on a track, or to the specification of the organization of unblocked records on the track. There must be one track block name for each kind of track format that may occur in a cylinder.

The track block name must appear as the first parameter in a BLOCK statement when the name refers to the specification of the organization of sentences on a track. The BLOCK statement must have the following parameters:

a) parameter (ii) of the BLOCK statement is assigned the system name:

NOORD, when there is more than one kind of sentence format in the track, and the various kinds of track format may occur in any order, and

SPEC, when there is only one kind of sentence format in the track, or when there is more than one and the various kinds occur in the order in which they are named in parameter (iii) of the BLOCK statement.
b) the organization of sentences in a track may be specified by as many levels of BLOCK statements as are needed. The first list parameter of parameter (iii) refers either to a BLOCK statement describing groupings of sentences, or directly to a BBLOCK statement describing a sentence format.

c) parameters (iv) and (v) of the BLOCK statement are interpreted as track header and trailer labels. Each header and trailer will be set off by gaps on the track.

The track block name must appear as the first parameter in a BBLOCK statement when the name refers to the specification of the organization of unblocked records on the track. The BBLOCK statement must have the following parameters:

a) parameter (ii) of the BBLOCK statement is assigned the system name FIXED (or F).

b) Parameter (ii) of the BBLOCK statement is assigned the number of bytes per track for the given disk.

c) Parameters (x) and (xi) of the BBLOCK statement are interpreted as header and trailer labels that precede and follow each record on the track. Each header and trailer will be set off by gaps.
(v) sentence block name. The sentence block name refers to the specification of the organization of records in a sentence. The name must appear as the first parameter in a BBLOCK statement with the following parameters:

Parameters (x) and (xi) of the BBLOCK statement are interpreted as header and trailer labels that precede and follow each sentence. Each header and trailer will be set off by gaps.

(vi) association list name. This optional parameter is used when the header and/or trailer parameter in the BLOCK or BBLOCK statements referred to by parameters (ii) - (v) refer to records. The name must appear as the first parameter in an ASSOCIATE statement (see Section 2.3.1.1).

Example

The following statements specify that data is to be input from disk. The data is in the form of a file which is stored in fixed length sentences of 7241 bytes. There are to be 90 record occurrences of the record 'BILL-RECORD' per sentence, 1 sentence per track, and 20 tracks per cylinder. Only sentences are to have header labels.

```
DISKIN ( 'DISKINP', 'FILE-BLOCK'; 'DATA-CYL';
    'DATA-TRACK'; 'DATA-SENTENCE' )
```
BLOCK ( 'FILE-BLOCK', SPEC;
 ( 'DATA-CYL', M, NOLIM ) )

BLOCK ( 'DATA-CYL', SPEC;
 ( 'DATA-TRACK', M, 20 ) )

BLOCK ( 'DATA-TRACK', SPEC;
 ( 'DATA-SENTENCE', M, l ) )

BBLOCK ( 'DATA-SENTENCE', F, 7241;
 SPLIT, F, 90, l, SPEC;
 ( 'BILL-RECORD', M, NOLIM );
 HDR, 'HDR' )
2.1.11.8 DISKOUT Statement

format

DISKOUT ( disk specification name, file block name;
cylinder block name, ... , cylinder block name;
track block name, ... , track block name;
sentence block name, ... , sentence block name
[ ; association list name ] )

parameters

(i) disk specification name is an unindexed user-defined name.
(ii) file block name is an unindexed user-defined name.
(iii) cylinder block name is a reference name.
(iv) track block name is a reference name.
(v) sentence block name is a reference name.
(vi) the optional parameter, association list name,
is an unindexed user-defined name.

usage of the statement

The DISKOUT statement is used to specify that output is to be placed on disk. To specify this, it is necessary to include the following information in the DISKOUT statement:

(i) - (vi) The parameters are the same as those specified for the DISKIN statement of Section 2.1.11.7
9.1.11.9  TTYIN Statement

format       TTYIN ( teletype specification name, file block name;
             page block name, ... , page block name
             [ ; association list name ] )

parameters   (i) teletype specification name is an unindexed
             user-defined name.
(iii) page block name is an unindexed user-defined name.

usage        (ii) file block name is an unindexed user-defined name.
             (iv) the optional parameter, association name, is
             an unindexed user-defined name.

The TTYIN statement is used to specify that the

   teletype is to be used for inputting data. To specify

this it is necessary to include the following information

in the TTYIN statement:

   (i) teletype specification name. This parameter
   is the name of the device specification being
   described.

   (ii) file block name. This parameter refers to a
   block which describes the layout of the data
   being output. The block must describe the
   organization of pages to be printed. The file
   block name must appear as the first parameter
   of a BLOCK statement with the following parameters:
a) parameter (ii) of the BLOCK statement is assigned the system name:

NOORD, when there is more than one kind of page format and the various kinds may occur in any order, and

SPEC, when there is only one kind of page format, or when there is more than one and the various kinds must occur in the order specified by parameter (iii) of the BLOCK statement.

b) the organization of pages in a file may be specified by the many levels of BLOCK statement as are needed. The first list parameter (iii) refers either to other BLOCK statements or to BBLOCK statements.

c) parameters (iv) and (v) of the BLOCK statement are interpreted as lines. Each string of the form:

[ ; HDR, header ] or
[ ; TLR, trailer ] is interpreted as a line.

(iii) page block name. This parameter refers to the specification of each page format in the file. There must be one page block name for each kind of page format that may occur. The page block names must appear as the first parameters in BBLOCK statements which specify the location of record occurrences on a page. The BBLOCK statements must have the following parameters:
a) parameter (ii) of the BBLOCK statement is assigned the system name:

FIXED (or F) when the number of lines in a page is to be fixed, and

VARIABLE (or V) when the number of lines in a page is to vary from one page to another.

b) parameters (x) and (xi) are interpreted as header and trailer lines.

(iv) association list name. This optional parameter is used when the header and/or trailer parameters in the BLOCK and BBLOCK statements referred to by parameters (ii) and (iii) refer to records. The name must appear as the first parameter in an ASSOCIATE statement (see Section 2.3.1.1).

Example

See Section 2.1.11.10 for an example of the use of the teletype for input and output.
2.1.11.10 TTYCUT Statement

format

TTYCUT ( teletype specification name, file block name;
page block name, ... , page block name
[ ; association list name ]

parameters

(i) teletype specification name is an unindexed user-defined name.
(ii) file block name is an unindexed user-defined name.
(iii) page block name is an unindexed user-defined name.
(iv) the optional parameter, association name, is an unindexed user-defined name.

usage of the statement

The TTYCUT statement is used to specify that the teletype is to be used for outputting data. To specify this it is necessary to include the following information in the TTYCUT statement:

usage of the parameters

(i) - (iv) The parameters are the same as those specified for the TAPEIN statement of Section 2.1.11.9.

Example

The following statements specify that the teletype is to be used to output occurrences of the record 'EMPLOYEE COLLEGE REPORT' given in Example 6 of Section 2.1.6. Each page output is to have 64 lines. Record occurrences are to be split across pages. There are to be
no headers or trailers.

TTYOUT ( 'TTYOUTPUT', 'FILE-BLOCK';

'PAGE BLOCK' )

BLOCK ( 'FILE-BLOCK', SPEC;

( 'PAGE BLOCK', M, NOLIM ) )

BBLOCK ( 'PAGE-BLOCK', F, 64;

SPLIT, V, 1, NOLIM, SPEC;

( 'EMPLOYEE COLLEGE REPORT', M, NOLIM ) )
2.1.12 Record State Statements

The following terminology will be used in discussing Record State statements:

(i) When the values in a record are stored, or exist externally on an input or output device, the structure and device location of the values are called the Storage State of those values.

(ii) When the values in a record exist in a record structure only during an intermediate step in the translation of those values between Storage States, the structure and temporary location of the values in the DDL processor work space are called the Logical State of those values.
2.12.1 STORRCD Statements

format

STORRCD ( storage record name, record structure name
[ , device specification name ] )

parameters

(i) storage record name is an unindexed user-defined name.

(ii) record structure name is an unindexed user-defined name.

(iii) the optional parameter, device specification name, is an unindexed user-defined name.

usage of the statement

The STORRCD statement is used to specify the Storage State of the values of a record. That is, it is used to declare that the occurrences of a particular record are to be input, or output on a particular device, or that they are to be stored physically, and to specify the storage device when this is to differ from the conventions of the DDL implementation. Thus, to specify the Storage State of the values of a record, it is necessary to include the following information in the STORRCD statement:

usage of the parameters

(i) storage record name. This parameter is the name of the Storage State being specified. It is used to refer to values in the Storage State.

(ii) record structure name. This parameter is the name of the structure of the values. The name must appear as the first parameter in a RECORD statement.
(iii) device specification name. This parameter names the specification of the input or output device, or of the storage device to be used if it is to differ from the system convention. If this parameter is not used, the system conventions will be followed for storage. The name must appear as the first parameter of a device statement.

Example

To specify that the values of the record 'PERSRCD' described in the Example of Section 2.1.3 are to be stored physically, and to assign the name 'STORPERSRCD' to the Storage State, the following statement is used:

```
STORRCD ( 'STORPERSRCD', 'PERSRCD' )
```

The following statements completely specify the Storage State of the values of the record 'PERSRCD':

```plaintext
FIELD ( 'SURNAME', V, C, 15, C )
FIELD ( 'COLLEGE', V, C, 30, C )
FIELD ( 'YRS', F, C, 2, C )
GROUP ( 'COLINF', SPEC;
  ( 'COLLEGE', M, F, 1 ),
  ( 'YRS', M, F, 1 ) )
GROUP ( 'PERSDATA', SPEC;
  ( 'SURNAME', M, F, 1 ),
  ( 'COLINF', O, V, NOLIM ) )
RECORD ( 'PERSRCD', 'PERSDATA' )
STORRCD ( 'STORPERSRCD', 'PERSRCD' )
```
2.1.12.2 LOGRCD Statements

LOGRCD ( logical record name, record structure name )

(i) logical record name is an unindexed user-defined name.

(ii) record structure name is an unindexed user-defined name.

The LOGRCD statement is used to specify the Logical State of the values of a record. It is used when values which already exist in a Storage State are to be given a new structure without creating a new copy of the values. This is done by specifying the new structure and translating values from their current structure in their Storage State to the new structure only when they are to be used. The values exist in the new structure only in the DDL processor's working space prior to translation to an output or storage state. Thus, to specify the Logical State of the values of a record, it is necessary to include the following information in the LOGRCD statement:

(i) logical record name. This parameter is the name of the Logical State being specified. It is used to refer to values in the Logical State.

(ii) record structure name. This parameter is the name of the structure of the values. The name must appear as the first parameter in a RECORD
Consider the Storage State described in the Example of Section 2.1.12.1. If a user wants to use all the values in that Storage State except those of the field 'YRS', and he does not want to have a new copy of the values created, he may specify a Logical State for those values. The structure of the Logical State, to be called 'LPERSRCD', would exclude the field 'YRS'.

To specify that values are to have the structure 'LPERSRCD', and to assign the name 'LOGPERSRCD' to the Logical State, the following statement is used:

```
LOGRCR ('LOGPERSRCD', 'LPERSRCD')
```

The following statements completely specify the Logical State of the values of the record 'LPERSRCD':

- `FIELD ('SURNAME', V, C, 15, C)`
- `FIELD ('COLLEGE', V, C, 30, C)`
- `GROUP ('LPERSDATA', SPEC;
  ( 'SURNAME', M, F, 1 ),
  ( 'COLLEGE', O, V, NOLIM ) )`
- `RECORD ('LPERSRCD', 'LPERSDATA')`
- `LOGRCR ('LOGPERSRCD', 'LPERSRCD')`
2.2 File Specification Statements

The following terminology will be used in discussing File Specification Statements:

(i) A record is the structure of values which are to be treated as the basic unit of storage and retrieval. Each set of values which occurs in the structure is called an occurrence of that record (or record occurrence). For example, the 'PERSRCD' record of the example in Section 2.1.3 is the structure of values giving a person's name, and college education. An occurrence of this record might be the set of values: DANIELS PURDUE 03.

(ii) File Specification Statements are statements that are used to describe how record occurrences are organized into structures called files.

(iii) Files are described as sets of links between record occurrences. A link connects one record occurrence called the source occurrence to another record occurrence called the target occurrence.

Links may be implemented by storing the record occurrences sequentially, or by connecting them with a pointer. The pointer is either stored in the record occurrence or it is stored separately in a table.
2.2.1 Linkage Statements

Linkage statements are statements that are used to define sets of links in the following way:

A source occurrence of a record is linked to a target occurrence of a record if and only if a criterion defined over values of fields in the records is satisfied.

In describing such criteria it may be necessary to indicate that fields and groups are being compared from different occurrences of the same record. To refer to such fields or groups unambiguously, field or group reference names are modified in the following way: The record name following the system name "IN" in a reference name is replaced by the OCC statement defined in Section 2.1.1.1 below. Reference names modified in this way are also reference names.

When a criterion for linking two record occurrences depends on values in record occurrences other than these two, it is necessary to describe this dependence in one of the following ways:

i) the criterion is satisfied if it is true for all other occurrences of the record,

ii) the criterion is satisfied if it is true for at least one other occurrence of the record. To specify these cases, the criterion expression in a CRITEX statement is replaced either by an ALLOC statement (see Section 2.1.1.2, below), or by a SOMEOCC statement (see Section 2.1.1.3, below).
2.2.1.1 OCC Statements

format

OCC ( record name, occurrence name )

parameters

(i) record name is an unindexed user-defined name.
(ii) occurrence name is either the system name:

T, or

S, or the string

Xn, where n is an integer.

usage of the statement

The OCC statement is used in a reference name to refer to a particular occurrence of a record. To specify the particular occurrence, the following information must be included in the OCC statement:

usage of the parameters

(i) record name. This parameter names the record whose occurrence is being referred to.

(ii) occurrence name. This parameter specifies the particular occurrence, that is being referred to. The parameter is assigned the system name:

T, if the record occurrence being referred to is the target occurrence of the linkage.

S, if the record occurrence being referred to is the source occurrence of the linkage, and

Xn, if the record occurrence is an occurrence other than the source or target occurrence.

For each such distinct occurrence the integer must be different.
Consider the case where a set of linkages is being defined between occurrences of a particular record, say 'PERSRCD', which is described in the Example of Section 2.1.3. If the criterion determining the linkages is defined on the values of the field 'SURNAME' in 'PERSRCD', such that the value of 'SURNAME' in the source occurrence of 'PERSRCD' is less than or equal to the value of 'SURNAME' in the target occurrence of 'PERSRCD', then the following statements are used to specify each of these occurrences:

\[
\text{OCC ( 'PERSRCD', T )}
\]

\[
\text{OCC ( 'PERSRCD', S )}
\]

These statements are used in reference names to refer to the value of the field 'SURNAME' in each occurrence of 'PERSRCD' as follows:

\[
\text{'SURNAME' IN OCC ( 'PERSRCD', T )}
\]

\[
\text{'SURNAME' IN OCC ( 'PERSRCD', S )}
\]

These reference names would be used in a criterion expression, to specify the condition on linkage described above, as follows:

\[
\text{CRITEX ( 'CRITEX1',}
\]

\[
( \text{'SURNAME' OF OCC ( 'PERSRCD' S )})
\]

\[
\text{LE ( 'SURNAME' OF OCC ( 'PERSRCD', T ) )}
\]

Consider the case where a set of linkages is being defined between occurrences of the same record as in Example 1, above. In this case the criterion determining the linkages is defined on the values of 'SURNAME' such that two record occurrences are linked if and only if the values of 'SURNAME' in the source and target occurrences are related as in Example 1, above, and there is no other record occurrence in which the value of 'SURNAME' is less than the value of 'SURNAME' in the target occurrence and greater than the value of 'SURNAME' in the source occurrence. To describe this by a criterion expression, reference must be made to occurrences of the record 'PERSRCD' other than the source and target occurrence. The following statement specifies this:

\[ \text{OCC ( 'PERSRCD', X1 )} \]

This statement would be used in a reference name to refer to the value of the field 'SURNAME' in the other occurrences of 'PERSRCD' as follows:

\[ \text{'SURNAME' OF OCC ( 'PERSRCD', X1 )} \]

This reference name would be used in a criterion expression to specify the part of the condition on linkage relating to the other occurrences of 'PERSRCD', as follows:

\[ \text{NOT ( (( 'SURNAME' OF OCC ( 'PERSRCD', X1 ) ) LT ( 'SURNAME' OF OCC ( 'PERSRCD', T ) ) AND ( ( 'SURNAME' OF OCC ( 'PERSRCD', S ) LT ( 'SURNAME' OF OCC ( 'PERSRCD', X1 ) ) ) ) )} \]

Note: To completely specify this part of the criterion the criterion expression above would appear as a
parameter in an ALLOC statement. This is demonstrated in the example given in Section 2.2.1.2 below.
2.2.1.2 ALLOCC Statements

format

ALLOCC ( occurrence name, ..., occurrence name;
        criterion expression form )

parameters

(i) occurrence name is a string of the form:
    Xn, where n is an integer.
(ii) criterion expression form is
    a) a string of the form defined in Section
       2.1.8 for parameters (ii)a) - e) of the
       CRITEX statement,
    b) an ALLOCC statement with the format described
       above, or
    c) a SOMEOCC statement with the format described
       in Section 2.2.1.3.

usage of the statement

The ALLOCC statement is used as a criterion expression form in a CRITEX statement. An ALLOCC statement indicates that the criterion expression form in the statement is satisfied when it is true for all record occurrences for all the occurrences named in the first parameter of the statement. To specify this, the following information must be included in the ALLOCC statement:
usage of the parameters

(i) occurrence name. These parameters are used to indicate the reference names for which the values of all the other occurrences of a record must be tested.

(ii) criterion expression form. This parameter describes the criterion for linkage. It must contain references to at least one occurrence of a record other than the source or target occurrences.

Example

To completely specify the criterion described in Example 2 of Section 2.2.1.1, it is necessary to state that the criterion expression must be satisfied for all occurrences of 'PERSRCD'. This is specified by the following statement:

CRITEX ( 'CRITEX2', ALLOCC ( X1;

    NOT ( ( ( 'SURNAME' OF OCC ( 'PERSRCD', X1 ) )

    LT ( 'SURNAME' OF OCC ( 'PERSRCD', T ) )

    AND ( ( 'SURNAME' OF OCC ( 'PERSRCD', S ) )

    LT ( 'SURNAME' OF OCC ( 'PERSRCD', X1 ) ) ) ) ) )
2.2.1.3 SOMEOCC Statements

format

SOMEOCC ( occurrence name, ... , occurrence name;
  criterion expression form )

parameters

(i) occurrence name is a string of the form:
  Xn, where n is an integer.

(ii) criterion expression form is
  a) a string of the form defined in Section 2.1.8 for parameters (ii)a) - e) of the
     CRITEX statement,
  b) an ALLOCC statement with the format described in Section 2.2.1.2, or
  c) a SOMEOCC statement with the format described above.

usage of the statement

The SOMEOCC statement is used as a criterion expression form in a CRITEX statement. A SOMEOCC statement indicates that the criterion expression form in the statement is satisfied when it is true for at least one record occurrence for all the occurrences named in the first parameter of the statement. To specify this, the following information must be included in the SOMEOCC statement;
usage of the parameters

(i) occurrence name. These parameters are used to indicate the reference names for which values must be tested from other occurrences of the record until the criterion is satisfied once, or until no further values remain to be tested.

(ii) criterion expression form. This parameter describes the criterion for linkage. It must contain references to at least one occurrence of a record other than the source or target occurrences.

Example

The criterion described in the Example of Section 2.2.1.2 can also be described by stating that: it must not be the case that there is a single occurrence of the record which contains a value of 'SURNAME' that is greater than the value of 'SURNAME' in the source occurrence and less than the value of 'SURNAME' in the target occurrence:

CRITEX ( 'CRITEX2-ALT',

NOT ( SOMEOCC ( X1;

    ( ( 'SURNAME' OF OCC ( 'PERSRCD', X1 ) )
    LT ( 'SURNAME' OF OCC ( 'PERSRCD', T ) )
    AND ( ( 'SURNAME' OF OCC ( 'PERSRCD', S )
    LT ( 'SURNAME' OF OCC ( 'PERSRCD', X1 ) ) )
))
) )
2.2.1.4 LINK Statement

format

| LINK ( link name; record state name, record state name; criterion name, link uniformity, link number ) |

parameters

| (i) link name is an unindexed user-defined name. |
| (ii) and (iii) record state name is an unindexed user-defined name. |
| (iv) criterion name is either the system name: NOORD, or an unindexed user-defined name. |
| (v) link uniformity is either the system name: FIXED (or simply F), or VARIABLE (or simply V). |
| (vi) link number is either the string: n, where n is an integer, or the system name NOLIM. |

usage of the statement

The LINK statement is used to specify the means for determining when one occurrence of a record is to be linked to another occurrence of the same or of a different record, and the maximum number of target occurrences that may be linked to a single source occurrence. To specify this, it is necessary to include the following information in the LINK statement:

usage of the parameters

| (i) link name. This parameter is the name of the set of links being specified. |
(ii) record state name. This parameter is used to name the state of the source record occurrence. The name must appear as the first parameter in a Record State statement.

(iii) record state name. This parameter is used to name the state of the target record occurrence. The name must appear as the first parameter in a Record State statement.

(iv) criterion name. This parameter names the criterion over values in record occurrences which determines the existence of a link. The name must appear as the first parameter in a CRITERION statement. If records are to be linked in the order in which they are input, then the parameter is assigned the system name: NOORD.

(v) link uniformity. This parameter specifies whether the number of target occurrences that may be linked to a single source occurrence of a record is to be the same for each source occurrence. The parameter must be assigned the system name:

- FIXED (or F) when there is to be a fixed number of target occurrences linked to each source occurrence of a record, and
- VARIABLE (or V) when the number of target occurrences linked to each source occurrence of a record may vary.
When NOORD is specified for parameter (iv),
link uniformity must be FIXED.

(vi) link number. This parameter specifies the
number of target occurrences that may be
linked to each source occurrence of a record.
The parameter must be assigned the string:

n, when exactly n target occurrences are to
be linked to each source occurrence of a
record, or when at most n target occurrences
are to be linked to each source occurrence
of a record, and

NOLIM, when there is no limit on the number of
target occurrences that may be linked to
each source occurrence of a record.

When NOORD is specified for parameter (iv),
link number must be 1.

The following statements specify how the record
occurrences of the Storage State 'STORPERSRCD' are to be
linked to form a sequential list in ascending lexicographical
order by the field 'SURNAME':

```
LINK ( 'LINK1'; 'STORPERSRCD', 'STORPERSRCD';
'CRIT', F, 1 )
```

CRITERION ( 'CRIT1', 'CRITEX' )
CRITEX ( 'CRITEX', ( 'CRITEX1' ) AND
( 'CRITEX2' ) )

where 'CRITEX1' and 'CRITEX2' are defined in the Examples
of Sections 2.2.1.1 and 2.2.1.2 respectively.
Example 2

The following statement specifies that the record occurrences of the Storage State 'STORPERSRCD' are to be linked in

```
LINK ( 'LINK2'; 'STORPERSRCD', 'STORPERSRCD';
   NOORD, F, 1 )
```

The LINK statement may be used to specify the means for determining when one occurrence of a repeating field or group is to occur before another occurrence of the same repeating field or group. In the LINK statement, parameter (v) must be assigned the system name: FIXED, and parameter (vi) must be assigned the string: 1. Parameter (iv) may refer to a CRITERION statement containing OCC, ALLOCC, and SOMEOCC statements in criterion expressions. These statements would be used to refer to occurrences of the repeating group of field.
2.2.1.5 INVLINK Statement

format

INVLINK ( inverse link name, link name )

parameters

(i) inverse link name is an unindexed user-defined name.

(ii) link name is an unindexed user-defined name.

usage of the statement

The INVLINK statement is used to specify that record occurrences are to be linked in the reverse direction to their linkage as defined by a LINK statement. To specify this, it is necessary to include the following information in the LINK statement:

(i) inverse link name. This parameter is the name of the set of links being specified.

(ii) link name. This parameter refers to the specification of the set of links for which reverse links are being defined. The name must appear as the first parameter in a LINK statement.

Example

The following statements specify that the record occurrences of the Storage State 'STORPERSRCD' are to be linked in the opposite order to the one in which they were input:

```
LINK ( 'LINK2'; 'STORPERSRCD', 'STORPERSRCD';
    NOORD, F, 1 )
```

```
INVLINK ( 'INVLINK1', 'LINK2' )
```
2.2.2 Linkage Implementation Statements

Links between record occurrences may be implemented by:

1) storing the target record occurrences sequentially after the source record occurrences;

2) embedding a pointer to the target record occurrences in the source record occurrence; and

3) storing a pointer to the target record occurrence in a directory.

The Linkage Implementation statements are statements that are used to specify that a set of links is to be implemented in one of these three ways.
2.2.2.1 SEQUEN Statement

format

SEQUEN ( sequence name; link name, ... , link name )

parameters

(i) sequence name is an unindexed user defined name.
(ii) link name is an unindexed user-defined name.

usage of the statement

The SEQUEN statement is used to specify that the occurrences of a record are to be organized sequentially in storage according to some criterion. To specify this, it is necessary to include the following information in the SEQUEN statement:

usage of the parameters

(i) sequence name. This parameter is the name of the sequential organization being specified.
(ii) link name. This parameter refers to the specification of the linkage structure which describes the sequence in which the record occurrences are to be stored. The name must appear as the first parameter in a LINK statement. If record occurrences are to be stored in the order in which they are input, then NOORD must be specified for parameter (iv) of the LINK statement.

Example

The following statements specify that the record occurrences in the Storage State described in the Example of Section 2.1.12.1 are to be stored sequentially in the
order in which they are input:

    LINK ( 'LINK2'; 'STORPERSRCD', 'STORPERSRCD';
            NOORD, F, 1 )

    SEQUEN ( 'SEQL'; 'LINK2' )
2.2.2.2 EMBED Statement

format

EMBED ( embedded pointer list name; link name;
    pointer name, ... , pointer name; pointer type
    [ , b/block name ] [ connection length expression ]
    [ connection set expression ] )

parameters

(i) embedded pointer list name is an unindexed user-defined name.
(ii) link name is an unindexed user-defined name.
(iii) pointer name is a reference name.
(iv) pointer type is either the system name: RECORD, or BLOCK.
(v) the optional parameter, b/block name, is a reference name.
(vi) the optional parameter, connection length expression is a string of parameters of the form:
    CLGTH, connection length [ , directory structure name ]

where:
(a) connection length is an integer,
(b) the optional parameter, directory structure name is an unindexed user-defined name.

(vii) the optional parameter, connection set expression, is a string of parameters of the form:
    CSETS, connection set count [ , directory structure name ]
where:

a) connection set count is an integer,
b) the optional parameter, directory structure
   name, is an unindexed user-defined name.

The EMBED statement is used to specify that record occurrences are to be linked by pointers stored in the record occurrences. To specify this, it is necessary to include the following information in the EMBED statement:

(i) embedded pointer list name. This parameter is the name of the embedded pointer linkage to be implemented.

(ii) link name. This parameter refers to the linkage structure which is to be implemented by pointers stored in the source record occurrences.

(iii) pointer name. This parameter refers to a field in the source record to be reserved for storing a pointer. The name must appear as the first parameter in a FIELD statement, in which the length parameter is set to the length of the addresses for the appropriate storage device. If the linkage defined allows more than one target occurrence of a record to be linked to the source occurrence, then there must be that many pointer fields defined in the source record.
Note: A pointer field may be defined as a field which repeats an unknown number of times to provide enough pointer fields for linkages requiring varying numbers of pointers.

(iv) pointer type. This parameter specifies whether a pointer points directly to a target occurrence of a record or to the block or basic block of the storage device medium in which the record occurrence is stored. The parameter must be assigned the system name:

   RECORD, when the pointer points directly to a target occurrence of a record, and
   BLOCK, when the pointer points to the block or basic block of the storage device medium in which the record occurrence is stored.

(v) b/block name. This optional parameter is used whenever parameter (iv) is assigned the system name BLOCK. It is used to specify the name of the block or basic block to which the pointer is to point. The name must appear as the first parameter in a BLOCK or BBLOCK statement.

(vi) connection length expression. This optional parameter is only used when the user wishes to partition a linkage structure by restricting the number of consecutively linked record occurrences. For example, if the structure specified is a list,
then the user may restrict the length of the list. Then, when the length limit is exceeded, a new list would be started. When a structure is partitioned in this way, the user may specify a directory that contains pointers to each partition of the structure. The connection length expression parameters must be assigned strings as follows:

a) connection length. This parameter gives the maximum number of record occurrences which may be linked consecutively.

b) directory name. This optional parameter refers to the specification of the directory. The directory must be specified as a file. The file may be complex, or as simple as a file containing a single record occurrence. The name must appear as the first parameter in a STORFILE statement (see Section 2.2.3).

(vii) connection set expression. This optional parameter is only used when the user wishes to partition a linkage structure by specifying that the structure must be split into a number of linked record occurrences. For example, if the structure specified is a list, then the user may specify that the list is to be broken into a number of lists. The connection set expression parameters must be assigned strings as follows:
a) connection set count. This parameter gives the number of partitions into which the structure being specified is to be broken.

b) directory name. This optional parameter refers to the specification of the directory. The directory must be specified as a file. The file may be complex, or as simple a file as one containing a single record occurrence. The name must appear as the first parameter in a STORFILE statement (see Section 2.2.3).

Note: Optional parameters (vi) and (vii) may not both appear in a single EMBED statement.

Example

Consider the following record state:

STORRC ( "BOOK-STATE", "BOOK" )
RECORD ( "BOOK", "BOOK-GROUP" )
GROUP ( "BOOK-GROUP", SPEC;
    ( "TITLE", M, F, 1 ),
    ( "AUTHOR", M, V, NOLIM ),
    ( "POINTER", M, F, 1 )
) FIELD ( "TITLE", V, C, NOLIM, C )
FIELD ( "AUTHOR", V, C, NOLIM, C )
FIELD ( "POINTER", F, C, 3, B )
The occurrences of the record 'BOOK' give
the title and authors of books in a library, and a
pointer to other record occurrences.

To organize these record occurrences such that
they are linked in a list alphabetically by title, the
following statements are used:

\[
\text{LINK ( 'BOOKS-BY-TITLE'; 'BOOK-STATE', }
\text{ 'BOOK-STATE'; 'TITLE-CRIT'; 1 )}
\]

\[
\text{CRITERION ( 'TITLE-CRIT', 'TITLE-CRITEX' )}
\]

\[
\text{CRITEX ( 'TITLE-CRITEX', ( 'T-CR1' ) AND }
\text{ ( 'T-CR2' ) )}
\]

\[
\text{CRITEX ( 'T-CR1', ( 'TITLE' OF OCC }
\text{ ( 'BOOK', S ) ) LE ( 'TITLE' OF OCC }
\text{ ( 'BOOK', T ) ) )}
\]

\[
\text{CRITEX ( 'T-CR2', NOT ( ALLOC ( X1, ( 'T-CR2.1' )}
\text{ AND ( 'T-CR2.2' ) ) ) )}
\]

\[
\text{CRITEX ( 'T-CR2.1', ( 'TITLE' OF OCC ( 'BOOK', X1 ) )}
\text{ LT ( 'TITLE' OF OCC ( 'BOOK', T ) ) )}
\]

\[
\text{CRITEX ( 'T-CR2.2', ( 'TITLE' OF OCC }
\text{ ( 'BOOK', S ) ) LT ( 'TITLE' OF OCC }
\text{ ( 'BOOK', X1 ) ) )}
\]

\[
\text{EMBED ( 'EMBED-BOOKS-BY-TITLE';}
\text{ 'POINTER', 'BOOKS-BY-TITLE' )}
\]
### 2.2.3 DIREC Statement

**format**

DIREC (directory structure name; linkage expression, directory name [, association list name]; pointer name, pointer type [, b/block name] [, ; criterion name])

**parameters**

(i) directory structure name is an unindexed user-defined name.

(ii) linkage expression is either the string:

a) link name
   where link name is an unindexed user-defined name, or the system name

b) EMBPTRDIREC.

(iii) directory name is an unindexed user-defined name.

(iv) the optional parameter, association list name, is an unindexed user-defined name.

(v) pointer name is a reference name.

(vi) pointer type is either the system name:

   RECORD, or

   BLOCK.

(vii) the optional parameter, b/block name, is a reference name.

(viii) the optional parameter, criterion name, is an unindexed user-defined name.
The DIREC statement is used to specify that record occurrences are to be linked by storing pointers in a directory. The statement is also used to create a directory for structure, implemented by embedded pointers, which are partitioned into several sets of linked record occurrences by length or by connection set limits. To specify this, it is necessary to include the following information in the DIREC statement:

(i) directory structure name. This parameter is the name of the directory linkage to be implemented.

(ii) linkage expression. This parameter is used to indicate whether a directory is to be created for a linkage or for partitions of a linkage implemented by embedded pointers. The parameter must be assigned the string:

a) link name, when the directory is to be created for the linkage structure referred to by the link name. The name must appear as the first parameter in a LINK statement.

b) EMBPTADIREC, when the directory is to be created for the partitions of a linkage implemented by embedded pointers.
(iii) directory name. This parameter refers to the specification of the directory. The directory must be specified as a file. The file may be as complex as necessary or as simple as a file containing a single record occurrence. The name must be the first parameter in a STORFILE statement (see Section 2.2.3).

(iv) association list name. This optional parameter is used when the directory is to contain fields and groups whose values are obtained from the record occurrences pointed to. The parameter refers to the specification of the relationship between fields and groups grouped with a pointer and their source of values in the record occurrence pointed to. The name must appear as the first parameter in an ASSOCIATE statement (see Section 2.3.1).

(v) pointer name. This parameter specifies a field in a record of the directory to be reserved for storing a pointer. The name must appear as the first parameter in a FIELD statement, in which the length parameter is set to the length of the addresses for the appropriate storage device.

(vi) pointer type. This parameter specifies whether a pointer points directly to a record occurrence or to a block or basic block of the storage device medium in which the record occurrence is stored. The parameter must be assigned the
system name:

RECORD, when the pointer points directly to a record occurrence, and

BLOCK, when the pointer points to the block or basic block of the storage device medium in which the record occurrence is stored.

(vii) b/block name. This optional parameter is used whenever parameter (v) is assigned the system name BLOCK. It is used to specify the name of the block or basic block in which the record occurrence to be pointed to is stored. The name must appear as the first parameter in a BLOCK or BBLOCK statement.

(viii) criterion name. This optional parameter refers to a criterion which determines whether or not the directory is to be created. This is necessary when hierarchies of directories are to be specified. The name must appear as the first parameter in a CRITERION statement.

Example

The following statement specifies that record occurrences in the Storage State 'STORPERSRCD' described in the Example of Section 2.1.12.1 are to be linked sequentially by value of the field 'SURNAME'. The structure is to be implemented by a directory 'DIREC1' in which
each entry consists of a pointer 'PTR' and a field 'PSURNAME' which has the value of the field 'SURNAME' in the record occurrence pointed to.

```
DIREC ( 'PERSFILE', 'LINK1', 'DIREC1', 'ASSOC',
    'PTR', RECORD )
```

where: 'PERSFILE' is the name of the structure implemented by a directory.

'LINK1' refers to the linkage structure being implemented. 'LINK' is specified in the Example of Section 2.2.1.4.

'DIREC1' refers to the specification of the structure of the directory. It is assumed that it is specified to be a sequentially stored set of record occurrences, each record occurrence consisting of the two fields 'PSURNAME' and 'PTR'.

'ASSOC' refers to the ASSOCIATE statement which relates the value of 'PSURNAME' in the directory record to 'SURNAME' in 'STORPERSRCD'.

'PTR' is the name of the field in the directory record reserved for the pointer.

RECORD indicates that the pointer points directly to the record occurrence.

The structure of the file is illustrated by the diagram that follows:
Directory
'DIREC1'

Record occurrences of
'STORPERSRCD'

'PSURNAME'
'ADAMS'

'PTR'

'PSURNAME'
'ZERBY'

'PTR'

'PSURNAME'
'NEWMAN'

'SURNAME'
'JONES'

'PSURNAME'
'ZERBY'

'SURNAME'
'ADAMS'
### 2.2.3 STORFILE Statements

**Format**

```plaintext
STORFILE ( file name; structure name, ...;
structure name )
```

**Parameters**

(i) file name is an unindexed user-defined name.

(ii) structure name is an unindexed user-defined name.

**Usage of the Statement**

The STORFILE statement is used to specify the organization of record occurrences in storage. This organization is called a storage file and is specified by describing the sequential storage order of the records and the linkages implemented by embedded pointers.

To specify this, the following information must be included in the STORFILE statement.

**Usage of the Parameters**

(i) file name. This parameter is the name of the storage file being specified.

(ii) structure name. These parameters refer to the descriptions of structures implemented by sequential storage or embedded pointers. One structure name must appear as the first parameter in a SEQUEN statement. The other structure names must appear as the first parameters in EMBED statements and/or DIREC statements.
The following statement specifies the organization of a storage file named 'STORFILE1' in which record occurrences of the Storage State 'STORPERSRCD' are to be stored sequentially in the order in which they are input:

```
STORFILE ( 'STORFILE1'; 'SEQ1' )
```

where 'SEQ1' is defined in the Example of Section 2.2.2.1.
2.2.4 LOGFILE

format
LOGFILE ( file name; direc structure name, \ldots, direc structure name )

parameters
(i) file is an unindexed user-defined name.
(ii) direc structure name is an unindexed user-defined name.

usage of the statement
The LOGFILE statement is used to specify new structures for record occurrences already organized as files. The new structures are implemented by directories such that no changes are made either to the record occurrences themselves or to their organizations. Such new structures are called logical files and are specified by describing the linkages connecting record occurrences. To specify a logical file, the following information must be included in the LOGFILE statement:

usage of the parameters
(i) file name. This parameter is the name of the logical file being specified.
(ii) direc structure name. This refers to the description of the linkages and directories implementing them. Each name must appear as the first parameter in a DIREC statement.
Example

The following statement specifies that the logical file 'LOGFILE1' is to have the structure and directory described in the Example of Section 2.2.2.3:

```
LOGFILE ('LOGFILE1', 'PERSFILE')
```
2.3 Task Specification Statements

Task specification statements are statements that are used to request the translation of values from one described structure to another described structure for the purposes of:

1) creating new files,
2) revising current files,
3) retrieving data from files,
4) storing data into files,
5) updating data in files,
6) interfacing user-written programs with data in described files, and
7) extending the DDL.

These tasks are discussed in detail in the sections which follow.

For some of these tasks, it is necessary for the user to describe the relationships that exist between the data in its current structure and the same data in its new structure after translation. These relationships are specified by the Association statements. These statements are discussed in the first of the following sections. They are followed by the discussions of the individual task specification statements.
2.3.1 Association Statements

2.3.1.1 ASSOCIATE Statement

| format | ASSOCIATE ( association name; ( associate list ),
|        | ... , ( associate list ) [ ; criterion name ] ) |

| parameters | (i) association name is an unindexed user-defined name. |
|            | (ii) associate list is a string of parameters with the following format: |
|            | target name, source name |
|            | where: |
|            | a) target name is a reference name. |
|            | b) source name is a reference name. |
|            | (iii) the optional parameter, criterion name, is an unindexed user-defined name. |

The ASSOCIATE statement is used to specify the relationships between values occurring in one state and the same values occurring in a different state. Such relationships must be specified when values are to be translated from one state to another. The current state of a value is called its source state. Fields and groups in the structure of the source state are called source fields and source groups. The state into which a value is to be translated is called its target state. Fields and groups in the structure of the target state are called target fields and target groups. For each field or group
occurring in the target state of a translation, the source of the values of that field or group must be specified in term of source fields or groups. Thus, to specify the relationships between the states of a set of values, it is necessary to include the following information in the ASSOCIATE statement.

(i) association name. This parameter is the name used to refer to the relationships, between the states of sets of values, being specified.

(ii) associate list. This string of parameters is used to specify the relationship between the target and source states of a value or set of values. There must be a list for each field or group in the target structure, such that the sources of all values in an occurrence of the target structure are specified. Thus, if the sources of values of a target group are specified in an associate list, the sources of values of fields in that group need not be specified. The associate list parameters must be assigned strings as follows:

a) target name. This parameter refers to the target field of a value or the target group of a set of values.
b) source name. This parameter refers to the source of the values for the field or group referred to by parameter (ii) a), above.

Note: The source of values for repeating target fields or groups may be specified as follows:

1. If the target field or group repeats a fixed number of times, then an associate list may be specified for each repetition. Indexed user-defined names are used in the reference names to indicate particular repetitions of the repeating group or field.

2. If the target field or group repeats an unlimited number of times, then a single associate list must be specified, which relates the repeating target field or group to repeating source fields or groups.

(iii) criterion name. This optional parameter is used when more than one record is to be the source of values for the target record. The parameter refers to the criterion which determines which record occurrence for each record is to be retrieved. The name must appear as the first parameter in a CRITERION statement.
The following statement specifies the relationships between occurrences of values in the Storage State 'STORPERSRCD' described in the example of Section 2.1.12.1 and the occurrences of those same values in the Logical State 'LOGPERSRCD' described in the Example of Section 2.1.12.13, and assigns the name 'PERSASSOC1' to the specification:

ASSOCIATE ( 'PERSASSOC1',

( 'SURNAME', 'SURNAME' OF 'PERSRCD' ),

( 'COLLEGE', 'COLLEGE' OF 'COLINF' OF 'PERSRCD' ) )

where 'PERSASSOC1' is the name used to refer to the relationships being specified.

The value of 'SURNAME' in the target record is the value of 'SURNAME' in the source record 'PERSRCD'.

The value of the repeating field 'COLLEGE' in the target record is the value of the field 'COLLEGE' in the repeating group 'COLINF' in the source record 'PERSRCD'.

These relationships are illustrated in the diagram given below:
<table>
<thead>
<tr>
<th>Source Record 'PERSRCD'</th>
<th>Target Record 'LPERSRCD'</th>
</tr>
</thead>
<tbody>
<tr>
<td>'COLINF(1)'</td>
<td></td>
</tr>
<tr>
<td>'SURNAME'</td>
<td>'SURNAME'</td>
</tr>
<tr>
<td>'COLLEGE'</td>
<td>'COLLEGE(1)'</td>
</tr>
<tr>
<td>'YRS'</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>'COLINF(n)'</td>
<td></td>
</tr>
<tr>
<td>'SURNAME'</td>
<td></td>
</tr>
<tr>
<td>'COLLEGE'</td>
<td>'COLLEGE(n)'</td>
</tr>
<tr>
<td>'YRS'</td>
<td></td>
</tr>
</tbody>
</table>
### 2.3.1.2 COMBINE Statement

**Format**

```
COMBINE ( source name, record occurrence count type,  
          record occurrence count [ , criterion name ] )
```

**Parameters**

1. **Source Name** is a reference name.
2. **Record Occurrence Count Type** is either the system name:
   - FIXED (or simply F), or
   - VARIABLE (or simply V).
3. **Record Occurrence Count** is either the string:
   - n, where n is an integer, or the system name NOlim.
4. **Optional Parameter**, criterion name, is an unindexed user-defined name.

**Usage of the Statement**

The COMBINE statement may be used for parameter (ii) b) of the ASSOCIATE statement. The COMBINE statement is used when two or more occurrences of a record are to be the sources of values for a repeating field or group. To specify this, it is necessary to include the following information in the COMBINE statement:

1. **Source Name**. This parameter refers to the field or group which is the source of values, in each occurrence of a source record, for the target field or group named in the ASSOCIATE statement.
(ii) record occurrence count type. This parameter is used to specify whether the number of occurrences of the source record is to be the same for each occurrence of the target record being created. The parameter must be assigned the system name:

FIXED (or F) when the same number of occurrences of the source record are to provide values for each occurrence of the target record being created, and

VARIABLE (or V) when a different number of occurrences of the source record may provide values for each occurrence of the target record being created.

(iii) record occurrence count. This parameter is used to specify the number of occurrences of the source record which are to provide values for a single occurrence of the target record. The parameter must be assigned the string:

n, when the number of occurrences of the source record required is exactly n, or

when the number of occurrences of the source record required is, at most, n;

NOLIM, when no limit is to be set on the number of occurrences of the source records required.
(iv) criterion name. This optional parameter
refers to the criterion which determines
which source occurrences are to be retrieved.
The name must appear as the first parameter
in a CRITERION statement.

Example

Consider a record 'BOOK-RCD1' with fields:
'AUTH' and 'TITLE' organized as illustrated in the
diagram given below:

```
+------------------+
| 'AUTH'           |
|                  |
+------------------+
| 'TITLE'          |
+------------------+
```

The following statements indicate that the values
of the field 'TITLE' from all occurrences of the record
'BOOK-RCD1', with the same value of the field 'AUTH',
are to be combined in a single occurrence of a target
record:

```
COMBINE ( 'TITLE' OF 'BOOK-RCD1', V, NOLIM,
          'BOOK-CRIT' )
```

CRITERION ( 'BOOK-CRIT', 'BOOK-CRITEX' )
CRITEX ( 'BOOK-CRITEX',
          ( 'AUTH' OF OCC ( 'BOOK-RCD1', S ) )
          EQ ( 'AUTH' OF OCC ( 'BOOK-RCD1', T ) ) )

To specify that one or more occurrences of the source
record 'BOOK-RCD1' are to provide the values for a single
occurrence of the target 'BOOK-RCD2', which contains a
field 'AUTHOR' and the repeating field 'TITLE', the
following statement is used:
ASSOCIATE ('C-ASSOC');

( 'AUTHOR', 'AUTH' OF 'BOOK-RCD1' ),

( 'TITLE', COMBINE ( 'TITLE' OF 'BOOK-RCD1', V, NOLIM, 'BOOK-CRT' ) )

These relationships are illustrated in the diagram given below:

Source Record Occurrences

'TITLE'

'AUTH'

'BOOK-RCD1'

Target Record Occurrence

'TITLE(n)'

'AUTH(n)'

'BOOK-RCD2'

'TITLE(1)'

::

'AUTH'
2.3.1.3 DISJOIN

format

DISJOIN ( source name, record occurrence count type, record occurrence count )

parameters

(i) source name is a reference name.
(ii) record occurrence count type is either the system name:
    FIXED (or simply F), or
    VARIABLE (or simply V).
(iii) record occurrence count is either the string:
      n, where n is an integer, or the system name NOLIM.

usage of the parameters

The DISJOIN statement may be used for parameter (ii) b) of the ASSOCIATE statement. The DISJOIN statement is used when values of a repeating field or group in a single record occurrence are to be used to form two or more target occurrences of a record. To specify this, it is necessary to include the following information in the DISJOIN statement:

(i) source name. This parameter refers to the field or group which is the source of values, in each occurrence of a source record, for the target field or group named in the ASSOCIATE statement.
(ii) record occurrence count type. This parameter is used to specify whether the number of occurrences of the target record being created is to be the same for each occurrence of the source record. The parameter must be assigned the system name:

FIXED (or F) when the same number of occurrences of the target record are to be created from each occurrence of the source record, and

VARIABLE (or V) when a different number of occurrences of the target record are to be created from each occurrence of the source record.

(iii) record occurrence count. This parameter is used to specify the number of occurrences of the target record which are to be created from a single occurrence of a source record. The parameter must be assigned the string:

n, when the number of the target occurrences to be created is exactly n, or

when the number of the target occurrences to be created is at most n;

NOLIM, when no limit is to be set on the number of target occurrences to be created.
Example

Consider a record 'BOOK-RCD3' with the field 'TITLE' and the repeating field 'AUTHOR' organized as illustrated in the diagram given below:

<table>
<thead>
<tr>
<th>'TITLE'</th>
</tr>
</thead>
<tbody>
<tr>
<td>'AUTHOR(1)'</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>'AUTHOR(n)'</td>
</tr>
</tbody>
</table>

The following statement indicates that occurrences of this record are to be split into n target record occurrences, each containing a single value of the field 'AUTHOR':

\[
\text{DISJOIN ( 'AUTHOR', V, NOLIM )}
\]

To specify that n occurrences of the target record 'BOOK-RCD4' are to be formed from a single occurrence of the source record 'BOOK-RCD3', where 'BOOK-RCD4' contains two fields 'AUTH' and 'TITLE', the following statement is used:

\[
\text{ASSOCIATE ( 'D-ASSOC',}
\]
\[
( 'AUTH', \text{DISJOIN ( 'AUTHOR' OF 'BOOK-RCD3', V, NOLIM )},
\]
\[
( 'TITLE', 'TITLE' OF 'BOOK-RCD3' ) )
\]
appear as the first parameters in STORFILE or LOGFILE statements.

(iii) associate list name. This parameter names the association list which relates fields and groups in a record, in the file being created, to fields and groups in records in source files. The name must appear as the first parameter in an ASSOCIATE statement. There must be an associate list name for each record in the file being created.

Example

The following statement requests the creation of the storage file 'STORFILE1' which was described in the Example of Section 2.2.4. The values for the file will come from a file named 'STORFILEINP'. The association list relating fields and groups in the record occurrences of the Storage State 'STORPERSRCD' to the fields and groups in the records of the source file is named 'ASSOCPERS'.

```
CREATE ('STORFILE1'; 'STORFILEINP'; 'ASSOCPERS')
```
2.3.3 REVISE Statements

| format            | REVISE ( specification name, [ , DELETE ];
|                  | ... ; specification name [ , DELETE ] ) |
| parameters        | Specification name is a reference name. |
| usage of the      | The REVISE statement is used to request the |
| statement         | revision of a file description. To specify this |
|                   | task the following information must be included in |
|                   | the REVISE statement: |
| usage of the      | (i) specification name. This parameter is used |
| parameters        | to name the part of a file description to be |
|                   | revised. The name must appear as the first |
|                   | parameter of a statement in the description |
|                   | being revised. If the revision being made |
|                   | requires the replacement of part of the |
|                   | description with a new statement, the same |
|                   | name must be used for the new statement. The |
|                   | specification name must appear as the first |
|                   | parameter in the new statement. |
|                   | (ii) DELETE. This optional parameter is used to |
|                   | specify that the part of a description named |
|                   | by parameter (i) is to be deleted completely |
|                   | from the description being revised. |
Consider the following set of statements specified in the Example of Section 2.1.12.1:

FIELD ( 'SURNAME', V, C, 15, C )
FIELD ( 'COLLEGE', V, C, 30, C )
FIELD ( 'YRS', F, C, 2, C )
GROUP ( 'COLINF', SPEC;
              ( 'COLLEGE', M, F, 1 ),
              ( 'YRS', M, F, 1 ) )
GROUP ( 'PERSDATA', SPEC;
              ( 'SURNAME', M, F, 1 ),
              ( 'COLINF', 0, V, NOLIM ) )
RECORD ( 'PERSRCID', 'PERSDATA' )
STORRCID ( 'STORPERSRCID', 'PERSRCID' )

These statements specify the Storage State of record occurrences organized in the file 'STORFILE1' described in the Example of Section 2.2.4.

The following statements request that the specification of the group 'PERSDATA' be changed such that the group 'COLINF' occurs in it no more than one time:

GROUP ( 'PERSDATA', SPEC;
              ( 'SURNAME', M, F, 1 ),
              ( 'COLINF', 0, V, 1 ) )

REVISE ( 'PERSDATA' OF 'PERSRCID'
FOR 'STORFILE1' )
3.4 REtrieve Statements

**format**

`RETRIEVE ( file source; output file name,
association list name [ ; retrieved criterion expression ]
)`

**parameters**

(i) file source is a string of the form:

```
file name, ... , file name
```

where file name is an unindexed user-defined name.

(ii) output file name is an unindexed user-defined name.

(iii) association list name is an unindexed user-defined name.

(iv) the optional parameter, retrieval criterion expression, is a string of the form:

```
( source structure name [ , criterion name ] )

[ VIALINK link name FROM ( source structure name [ , criterion name ] ) VIALINK ...
 VIALINK link name FROM ( source structure name [ , criterion name ] ) ]
```

where:

a) source structure name is an unindexed user-defined name.

b) the optional parameter, criterion name, is an unindexed user-defined name.

c) the optional parameter, link name, is an unindexed user-defined name.
The RETRIEVE statement is used to request the retrieval of values from files created under the DDL processor. The statement specifies from which files the values are to be retrieved, the output structure of the values, the relationships between values in the output structure and those values in the structures in which they are stored, and the criterion for determining which record occurrences are to be retrieved. To specify this request, the following information must be included in the RETRIEVE statement:

(i) file source. This parameter specifies from which files values are to be retrieved. There may be one or more such source files. They may be storage or logical files. They must have been described in the DDL.

(ii) output file name. This parameter is used to specify the output organization of the data. The name must appear as the first parameter in a STORFILE statement.

(iii) association list name. This parameter names the association list which relates fields and groups in the output file structure to fields and groups in the Storage and/or Logical State structures of the source files. The name must appear as the first parameter in an ASSOCIATE statement.
(iv) retrieval criterion expression. This optional parameter is used to specify the criterion for determining which record occurrences are to be retrieved from the source files. If it does not appear in the statement it is assumed that all records in the source files are to be retrieved and output. Otherwise, the retrieval criterion is expressed as follows:

a) source structure name. This parameter in the retrieval criterion expression gives the structure of the record occurrences to be retrieved.

b) criterion name. This optional parameter in the retrieval criterion expression gives the criterion for selecting the particular record occurrences in the structure named by parameter a) above. If this parameter is omitted, all occurrences of the record are retrieved.

c) link name. This optional parameter in the retrieval criterion expression is used when record occurrences are to be retrieved by tracing through the different sets of links of a file. The link name is the name of a set of links defined as part of the file.

Note: When more than one file is listed for parameter (i), the retrieval criterion expression specifies the criterion for
retrieving record occurrences from only the first named file. The criteria for retrieving record occurrences from the remaining files is given in the ASSOCIATE statement referred to by parameter (iii), above.

The following statements request the retrieval of record occurrences from the file 'STORFILE1' whose creation was described in the Example of Section 2.3.2. Only those records are to be retrieved in which there are no values for the group 'COLINF'.

```
RETRIEVE ( 'STORFILE1'; 'OUTPUTFILE',
           'ASSOC-STORF1'; 'CRIT-NOCOLINF' )

CRITERION ( 'CRIT-NOCOLINF', 'CRITEX-NOCOLINF' )

CRITEX ( 'CRITEX-NOCOLINF', ( EXIST
           ( 'COLINF' OF 'PERSRCD' OF 'STORFILE' ) )
           EQ ( CONSTANT ( 0 , C ) ) )
```
2.3.5 STORE Statement

<table>
<thead>
<tr>
<th>format</th>
<th>STORE ( target file name, data source; association list name )</th>
</tr>
</thead>
</table>
| parameters              | (i) target file is an unindexed user-defined name.  
                        | (ii) data source is a string of parameters of the form:    
                        | source file name, ..., source file name [ ; retrieval criterion expression ] 
                        | where source file name is an unindexed user-defined name, and the optional parameter, retrieval criterion expression, is a string of the form described for parameter (iv) of the RETRIEVE statement in Section 2.3.4.  
                        | (iii) association list name is an unindexed user-defined name. |

The STORE statement is used to request the addition of record occurrences to a file which has already been created. To specify this task, the following information must be included in the STORE statement:

| usage of the statement | (i) target file name. This parameter names the file into which the record occurrences are stored.  
                        | (ii) data source. This parameter describes the source of the record occurrences to be stored. Each |
source file name must appear as the first parameter in a STORFILE or LOGFILE statement. When more than one source file name is given, it is assumed that record occurrences retrieved from each of the files named will be combined to form the record occurrence to be stored. The optional parameter, retrieval criterion expression, is used to specify the criterion for determining which record occurrences are to be retrieved from the source files named. When more than one source file is listed, the retrieval criterion expression specifies the criterion for retrieving record occurrences from only the first named source file. The criteria for retrieving record occurrences from the remaining source files is given in the ASSOCIATE statement referred to by parameter (iii), below. If the retrieval criterion expression does not appear in the statement, it is assumed that all records in the source file are to be retrieved.

(iii) association list name. This parameter names the association list which relates the values of fields and groups in the record to be stored to the values of the fields and groups in the source files. The name must appear as the first parameter in an ASSOCIATE statement.
The following statement requests that new record occurrences be stored in the file whose creation was described in the Example of Section 2.3.2. The values are to be input from a file name 'INPFIL'.

\[
\text{STORE ( 'STORFILE1', 'INPFIL'; 'ASSOCINP' )}
\]
2.3.6 UPDATE Statements

format

UPDATE ( file name; RPLVAL statement, ... ,

       RPLVAL statement [ ; retrieval criterion expression ] )

parameters

(i) file name is an unindexed user-defined name.

(ii) RPLVAL statement is a statement with the format
described in Section 2.1.9.1.

(iii) the optional parameter, retrieval criterion
expression, is a string of the form described
for parameter (iv) of the RETRIEVE statement in
Section 2.3.4.

usage of the statement

The UPDATE statement is used to request the modification of values in record occurrences. It specifies that new values are to replace current values or that values or sets of values are to be completely deleted, and it specifies criteria for determining in which record occurrences the modifications are to be made. To specify this, the following information must be included in the UPDATE statement:

usage of the parameters

(i) file name. This parameter specifies the file
in which the modifications are to be made. The file must have been described in the DDL.

(ii) RPLVAL statements. These parameters are used to specify the modifications to be made to record occurrences. As described in Section 2.1.9.1, the first parameter of the RPLVAL statement is
used to name the field or group whose values are to be replaced or deleted.

If the values are to be deleted, rather than replaced, the second parameter of the RPLVAL statement is assigned the system name: DELETE. Complete record occurrences can be deleted by assigning the first parameter of the RPLVAL statement the record name.

If the values are to be replaced, the second parameter of the RPLVAL statement is assigned a CONSTANT statement which specifies the string which is to replace the values named.

(iii) retrieval criterion expression. This optional parameter is used to specify the criterion for determining which record occurrences are to be modified. If the parameter does not appear in the UPDATE statement, it is assumed that all records in the file are to be modified. Otherwise, the retrieval criterion is expressed as described for parameter (iv) of the RETRIEVE statement in Section 2.3.4.

The following statements request the addition of values for the fields 'COLLEGE' and 'YRS' to the record occurrence, of the file created in the Example of Section 2.3.2, in which the value of 'SURNAME' equals "KANE".
CRITERION ( 'CRIT-UPDATE', 'CRITEX-UPDATE' )
CRITEX ( 'CRITEX-UPDATE', ( 'SURNAME' OF 'PERSRCD' )
EQ ( CONSTANT ( KANE, C ) )

UPDATE ( 'STORFILE1';
RPLVAL ( 'COLLEGE' OF 'COLINF',
CONSTANT ( PENN STATE, C ) ),
RPLVAL ( 'YRS' OF 'COLINF',
CONSTANT ( 02, C ) );
'CRIT-UPDATE' )
2.3.7 Interfacing Statements

The following terminology will be used in discussing the interfacing statements:

(i) A user-written program and files of data described in the DDL are said to be interfaced by the DDL processor when:
   a) values stored in the files are converted by the DDL processor into new structures and input to the executing user-written program expecting them in the new structure, and
   b) values output from the user-written executing program are converted from their output structure into the storage structures of the files by the DDL processor and stored into the files.

(ii) The I/O statements of user-written programs are called data calls.

(iii) A user-written program requires simple interfacing when:
   a) each input call in the program is made to a single file, and the input call expects the next record occurrences not yet retrieved, which satisfy the single retrieval criterion, and
   b) each output call in the user-written program is made to, at most, a single file described in the DDL.

   Such user-written programs are called simple programs.

(iv) A user-written program which does not require simple interfacing is said to require complex interfacing.

   Such user-written programs are called complex programs.
Interfacing statements are statements which are used to request the execution of user-written programs and their interfacing with DDL described files.

For a user-written program to be interfaced with a set of DDL described files:

a) the structure, in which input data is expected and in which output data is generated by the user-written program, must be described in the DDL for each data call in the user-written program.

b) each data call to DDL described files must, by convention, select the card I/O device.

c) the relationships, between values in the structures required by the data calls of the user-written programs and the same values in the structures of the DDL described files, must be described.

d) for complex programs, the user-written program must be modified such that each of its data calls is preceded by an output call, which is to be called a signal call. A signal call is to output a single string of characters, where this string is the name of the DDL description of the data call which follows.

The interfacing statements, which are used to specify this information are discussed in the sections that follow:
2.3.7.1 PROGDATA Statement

<table>
<thead>
<tr>
<th>format</th>
<th>PROGDATA ( program data name, type of interfacing, ( data call list ), ... , ( data call list ) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>parameters</td>
<td>(i) program data name is an unindexed user-defined name.</td>
</tr>
<tr>
<td></td>
<td>(ii) type of interfacing is either the system name: SIMPLE, or COMPLEX.</td>
</tr>
<tr>
<td></td>
<td>(iii) data call list is a string of parameters with the following format: data call name, call type where:</td>
</tr>
<tr>
<td></td>
<td>a) data call name is an unindexed user-defined name.</td>
</tr>
<tr>
<td></td>
<td>b) call type is either the system name: I, or 0.</td>
</tr>
</tbody>
</table>

The PROGDATA statement is used to describe the data calls of a user-written program, which is to be interfaced to DDL described files. The PROGDATA statement specifies whether the program requires simple or complex interfacing and gives the structure for each data call in the program. To specify this, it is necessary to include the following information in the PROGDATA statement:
usage of the parameters

(1) program data name. This parameter is the name of the program data description being specified.

(ii) type of interfacing. This parameter is used to specify whether the program whose data calls are being described requires simple or complex interfacing. The parameter is assigned the system name:

SIMPLE, when the program requires simple interfacing, and

COMPLEX, when the program requires complex interfacing.

(iii) data call list. Each data call list specifies the description of a data call in the user-written program to be interfaced. There must be one data call list for each data call in the program. The data call list parameters must be assigned strings as follows:

a) data call name. This parameter refers to the data call being specified. It is the name which must be output by the signal call, in the program, preceding the data call named. The name must appear as the first parameter in a RECORD statement. If the data call is not to the DDL described files the second parameter of the RECORD statement must be assigned the system name "EXTERNAL".

b) call type. This parameter specifies whether the data call is for input or output. It is assigned the system name; 
I, when the data call is for input, and 
0, when the data call is for output.

Note: The output structure for each signal call must be the same. To describe this structure, the first parameter of the data call list must be assigned the system name "SIGNAL". And this system name must appear as the first parameter of the RECORD statement which specifies the structure of the signal calls.

Example

Consider the following FORTRAN program called "FORPRG":

```
PROGRAM FORPRG

READ ( 5, 10 ) INP, FIELD1, FIELD 2
10 FORMAT ( I1, 2 ( 5X, F8, 5 ) )

IF ( INP ) 2,2,1
1 OUTPUT = ( FIELD1 + FIELD2 ) / 2.

GO TO 3
2 OUTPUT = 0.0

3 WRITE ( 7, 20 ) OUTPUT
20 FORMAT ( F8, 5 )

STOP

END
```
This program expects three values as input and generates a single output value. The program is written for simple interfacing (i.e., it contains no signal calls). The following statements specify the data requirements of the program:

```
RECORD ( 'INPUT', INP-STRUC' )
GROUP ( 'INP-STRUC', SPEC;
    ( 'INP', M, F, 1 ),
    ( 'FIELD', M, F, 2 ) )
FIELD ( 'INP', F, C, 1, C )
FIELD ( 'FIELD', F, C, 8, FL )
RECORD ( 'OUTPUT', 'OUT-STRUC' )
GROUP ( 'OUT-STRUC', SPEC;
    ( 'OUT', M, F, 1 ) )
FIELD ( 'OUT', F, C, 8, FL )
```

where: 'FORPRG-PD' is the name of the program data description for the program "FORPRG". SIMPLE indicates that the user-written program is a simple program. 'INPUT' is the name of the description of the input call in the program. 'OUT' is the name of the description of the output call in the program.
### 2.3.7.2 INTERFACE Statement

<table>
<thead>
<tr>
<th>format</th>
<th>INTERFACE ( program source; program language, program data name; ( list ), ... , ( list ) )</th>
</tr>
</thead>
</table>
| parameters | (i) program source is the string of parameters: file name, field name, retrieval criterion expression where:
  a) file name is an unindexed user-defined name,
  b) field name is a reference name, and
  c) retrieval criterion expression has the format described in Section 2.4.3, parameter (iv).

(ii) program language is either the system name:
  a) COBOL,
  b) FORTRAN,
  c) ASSEMBLY.

(iii) program data name is an unindexed user-defined name.

(iv) list is a string of parameters with the following format:
data call name, association list name, ... ,
association list name; file name, ... , file name
where:
  a) data call name is an unindexed user-defined name,
b) association list name is an unindexed user-defined name,
c) file name is an unindexed user-defined name.

The INTERFACE statement is used to request the execution of a user-written program and the interfacing of it with DDL described files. The statement specifies where the user-defined program is to be located, the files it is to be interfaced with, and the interfacing relationships between the program and the files. To specify this, it is necessary to include the following information in the INTERFACE statement:

(i) program source. This parameter describes where the user-written program is to be located.

Strings are assigned as follows:

a) file name is the name of the file in which the program is stored. This name must appear as the first parameter in a STORFILE or LOGFILE statement.

b) field name is the name of the field which has the program as its value. This name must appear as the first parameter in a FIELD statement.

c) The retrieval criterion expression specifies the criterion for determining which
record occurrence in the file contains the program. The retrieval criterion is expressed in accordance with the description of parameter (iv) of the RETRIEVAL statement in Section 2.3.5.

(ii) program language. This parameter names the language in which the user-written program is coded. The parameter is assigned the system name:
   a) COBOL, if the program is coded in COBOL,
   b) FORTRAN, if the program is coded in FORTRAN,
   c) ASSEMBLY, if the program is coded in Assembly language.

(iii) program data name. This parameter refers to the description of the data calls in the user-written program. The name must appear as the first parameter of a PROGDATA statement.

(iv) list. Each list specifies the relationship between values in the structure of a data call and the DDL described files from which the values are retrieved, or into which they are stored. There must be one list for each data call in the user-written program. The list parameters must be assigned strings as follows:
   a) data call name. This parameter refers to the data call being specified.
b) association list name. If the data call referred to by the list parameter (a) is an input call, then there is only one association list name. The name refers to the description of the relationships between values in the data call structure and values in the source files' structures. If the data call referred to by the list parameter (a) is an output call, then there are as many association list names as there are files in which the output values are to be stored. The name refers to the description of the relationships between values in the data call structure, and values in the target files' structures. In each case, the names must appear as the first parameters in ASSOCIATE statements.

c) file name. These parameters specify the files which are the source of input values for input data calls. They specify the files into which values are stored for output data calls. Each name must appear as the first parameter in a STORFILE or a LOGFILE statement.

Example

Assuming the FORTRAN program described in the Example of Section 2.3.7.1 is to be input from the file 'FORFILE' as the value of the field 'EXFOR', and is to be interfaced with the input file 'INFILE' and the out-
put file 'OUTFILE', the interfacing request is specified by the following statement:

```
INTERFACE ( 'FORFILE', 'EXTFOR', 'FORFILE', 'CRIT';
           FORTRAN, 'FORPRG-PD';
           ( 'INPUT', 'ASSOCINFILE', 'INFILE' ),
           ( 'OUTPUT', 'ASSOC-OUTFILE', 'OUTFILE' ) )
```
2.3.8 Language Extension Statements

The descriptive power of the DDL can be extended in two ways:

(i) by parameter programs. The user writes a program which
outputs an actual parameter of a DDL statement. The
name of the program and the input on which it runs are
entered in the place of the actual parameter in the DDL
statement.

(ii) by new statements. New device statements may be added
by the user along with user-written programs to process
data input or output on new devices.

These methods of extending the DDL are discussed in detail in
the two sections which follow:
### 2.3.8.1 PARAMPROG Statement

<table>
<thead>
<tr>
<th>format</th>
<th>PARAMPROG ( param name, program source; program language, program data name, input structure name [ ; output constant ] )</th>
</tr>
</thead>
</table>
| parameters | (i) param name is an unindexed user-defined name. (ii) program source is a string of parameters with the format described for parameter (i) of the INTERFACE statement in Section 2.3.7.2. (iii) program language is either the system name: 
  a) COBOL, 
  b) FORTRAN, 
  c) ASSEMBLY, 
(iv) program data name is an unindexed user-defined name. (v) input structure name is an unindexed user-defined name. (vi) the optional parameter, output constant, is either the system name: 
  B, or 
  C, or an unindexed user-defined name. |

---

The PARAMPROG statement is used to specify that a user-written program will generate actual parameters for use in DDL statements. Such user-written programs may only have two data calls: one input call and one output call. That is, the user-written program must
require only simple interfacing. The PARAMPROG statement specifies where the program is to be located, the structure of the input and output for the program, the structure of the input as it is to occur in a DDL statement. To specify this, it is necessary to include the following information in the PARAMPROG statement:

(i) param name. This parameter is the name which is to appear as the first part of a parameter in a DDL statement. When appearing as a part of a parameter, the name is not enclosed in apostrophes.

(ii) program source. The parameter describes where the user-written program is to be located. The parameter is assigned the strings described for parameter (i) of the INTERFACE statement in Section 2.3.7.2.

(iii) program language. This parameter names the language in which the user-written program is coded. The parameter is assigned the system name:

a) COBOL, if the program is coded in COBOL,

b) FORTRAN, if the program is coded in FORTRAN,

c) ASSEMBLY, if the program is coded in Assembly Language.
(iv) program data name. This parameter refers to the description of the data calls in the user-written program. The name must appear as the first parameter of a PROGDATA statement in which there are only two list parameters.

(v) input structure name. This parameter refers to the specification of the structure of the input values for the user-written program, as they appear as part of a parameter in a DDL statement. The input values must appear in the specified input structure. They are enclosed in parentheses and placed after the param name, given by parameter (i), above. Together name and the input values form a complete parameter for a DDL statement. The input structure name must appear as the first parameter in a RECORD statement.

Note: The field names of values in the input structure specification must be the same as the field names for those same values in the program data description.

(vi) output constant. This optional parameter is used when the user-written parameter program is to output a constant string. The parameter is assigned the system name:

$B$, when the string to be output is a binary string,
C, when the string to be output is an ASCII character string, and an unindexed user-defined when the string to be output is a string of characters other than ASCII. The name must appear as the first parameter in a CHAR statement.

If the user-written FORTRAN program, 'FORPRG', given in the Example of Section 2.3.7.1, is to be used as a parameter program called "AVERAGE" in DDL statement; and if it is to be input on cards in the External State, 'EXTFOR', this extension to the DDL is specified by the following statement:

```
PARAMPROG ( 'AVERAGE', 'FORFILE', 'EXTFOR', 'CRIT';
                FORTRAN, 'FORPRG-PD', 'INP-STRUC'; C )
```

where 'FORPRG-PD' refers to the PROGDATA statement given in the Example of Section 2.3.7.1, and

'INP-STRUC' refers to the RECORD statement given below:

```
RECORD ( 'INP-STRUC', 'INP-GROUP' )
```

where:

```
GROUP ( 'INP-GROUP', SPEC;
         ( 'INP', M, F, l ),
         ( 'FIELD', M, F, l ) )
FIELD ( 'INP', F, C, l, C;
       CONCODE ( CONSTANT ( ; , C )
                 DELIM, PIX, C ) )
FIELD ( 'FIELD', F, C, 8, FL;
       CONCODE ( CONSTANT ( , , C )
                 DELIM, INX, C ) )
```
An example of a DDL statement in which such a parameter might appear is the RPLVAL statement of Section 2.1.9.1. The following statement specifies that the value of a field 'X' is to be replaced with the average of the values of the fields 'Y' and 'Z' if field 'W' exists and with zero otherwise:

```
RPLVAL ( 'X', AVERAGE ( EXIST
         ( 'W' ); 'Y', 'Z' ) )
```
### 2.3.8.2 EXTEND Statements

**format**

EXTEND ( device name, program source;
    program language, program data name,
    input structure name )

**parameters**

(i) device name is an unindexed user-defined name.

(ii) program source is a string of parameters with
    the format described for parameter (i) of
    the INTERFACE statement in Section 2.3.7.2.

(iii) program language is either the system name:
    a) FORTRAN,
    b) COBOL,
    c) ASSEMBLY.

(iv) program data name is an unindexed user-defined
    name.

(v) input parameter name is an unindexed user-defined
    name.

**usage**

The EXTEND statement is used to specify that a
user-written program will accept values from a new input
device, or output or store values on a new output or
storage device. Such a user-written program may require
only simple interfacing. There may be only one input
call in the program to accept values from the DDL pro-
cessor and only one output call to generate values for
the DDL processor. The EXTEND statement specifies where
the user-written program is to be located, the structure of the I/O values for the program, and the structure of the input to the user-program from the new DDL statement. To specify this, it is necessary to include the following information in the EXTEND statement:

(i) device name. This parameter is used to specify the system name which forms the first part of the new device statement.

(ii) program source. This parameter describes where the user-written program is to be located. The parameter is assigned the strings described for parameter (i) of the INTERFACE statement in Section 2.3.7.2.

(iii) program language. This parameter names the language in which the user-written program is coded. The parameter is assigned the system name:
   a) COBOL, if the program is coded in COBOL,
   b) FORTRAN, if the program is coded in FORTRAN,
   c) ASSEMBLY, if the program is coded in Assembly Language.

(iv) program data name. This parameter refers to the description of the data calls in the user-written program. The name must appear as the first parameter of a PROGDATA statement.
(v) input parameter name. This parameter refers to the specification of the structure of the parameters which are to be input, via the new device statement being defined. In the new device statement, the parameters must appear in the structure named by the input parameter name. The parameters must be enclosed in parentheses and placed after the device name, given by parameter (i) above. Together, device name and input parameters form a complete device statement. The input parameter name must appear as the first parameter in a RECORD statement.

Note: The first input parameter for the new device statement must be a name which may be used to reference the statement from other DDL statements. The next parameters should be names which reference BLOCK or BBLOCK statements. Remaining parameters are at the discretion of the user. The DDL processor will replace BLOCK and BBLOCK names referred in the user created device statement by the actual parameters of those referenced statements. Similarly, if the user requires references to other statements, the parameters of these statements will references to them in the parameters of the user created device statement.
If a user writes an Assembly Language program to support input from tapes which have been generated by the system under which the DDL is running, then this extension to the DDL is specified by the following statement:

```
EXTEND ( 'NEWTAPEIN', 'ASSEMBLE', 'EXTASSEM', 'ASSEMCRIT'; ASSEMBLY, 'ASSEM-PD', 'NEWTAPEIN-STRUC' )
```

where:

'NEWTAPEIN' is the beginning string of characters in the new device statement being defined.

'ASSEMBLE' is the file in which the program written by the user to support input from system tapes is stored.

'EXTASSEM' is the name of the field in 'ASSEMBLE' containing the program.

'ASSEMCRIT' is the name of the criterion which determines which record occurrence in 'ASSEMBLE' contains the program.

ASSEMBLY indicates that the user-written program is coded in Assembly Language.

'ASSEM-PD' is the name of the program data description for the user-written program.

'NEWTAPEIN-STRUC' refers to the RECORD statement given below:

```
RECORD ( 'NEWTAPEIN-STRUC', 'TAPEIN-GR' )
```

where:
GROUP ('TAPEIN-GR', SPEC;

('NAME', M, F, 1 ),

('B/BLOCK- NAME', M, F, 1 ) )

FIELD ( 'NAME', V, C, NOLIM, C;

CONCODE ( CONSTANT ( , , C ) DELIM, PIIx ) )

FIELD ( 'B/BLOCK-NAME', V, C, NOLIM, C )

An example of the use of the new DDL device statement follows:

NEWTAPEIN ( 'X', 'B' )
3. Paragraphs

A paragraph is a system name, followed by a sequence of DDL statements, followed by another system name.

Paragraphs are provided as a convenience for the DDL user to break up sequences of DDL statements into meaningful parts. A DDL statement is used either to describe some part of a file structure, or to describe a task. Paragraphs are used to separate statements describing file structures from statements describing tasks.
3.1 DESCRIBE Paragraph

format

DESCRIBE [ ( paragraph name ) ] :

   File Structure Statement
   [ File Structure Statement ]
   :
   [ File Structure Statement ]

END

parameters

   (i) the optional parameter, paragraph name, is a user-defined name.
   
(ii) File Structure Statement is a statement of one of the following types:
      a) Record Specification Statement,
      b) File Specification Statement,
      c) ASSOCIATE Statement,
      d) PROCDATA Statement.

usage of the paragraph

   The DESCRIBE paragraph is used to list together all the statements completely describing a structure. For example, if a file structure is being specified, all the statements relating to the description of the file must be included as parameters in the DESCRIBE paragraph. Thus, to write a DESCRIBE paragraph, it is necessary to include the following information:

usage of the parameters

   (i) paragraph name. This optional parameter allows the user to assign a name to the paragraph. The
paragraph name may also be used for comments.

(ii) File Structure Statement. This parameter is used to specify a structure. Every statement referred to by a statement parameter must appear as a parameter in the paragraph.

Example

The statements in the Examples of Sections 2.2.3, 2.2.2.1, and 2.1.12.1 which describe the File 'STORFILE1' constitute a DESCRIBE paragraph when enclosed by the system names: "DESCRIBE:" and "END".
EXECUTE Paragraph

format

EXECUTE [ ( paragraph name ) ]:

Task Statement
[ Task Statement ]

;

[ Task Statement ]

END

parameters

(i) the optional parameter, paragraph name, is a user-defined name.

(ii) Task Statement is one of the following statements:

a) CREATE Statement,
b) REVISE Statement,
c) RETRIEVE Statement,
d) STORE Statement,
e) UPDATE Statement,
f) INTERFACE Statement,
g) PARAMPROG Statement, and
h) EXTEND Statement.

usage of the paragraph

The EXECUTE paragraph is used to list together statements describing tasks. Thus, to write an EXECUTE paragraph, it is necessary to include the following information:

usage of the parameters

(i) paragraph name. This optional parameter allows the user to assign a name to the paragraph. The
paragraph name may also be used for comments.

Note: The user may assign the same paragraph name to an EXECUTE paragraph as he assigned to a DESCRIBE paragraph for his own reference purposes.

(ii) Task Statement. This parameter is used to specify a task. The user may list each of his Task statements in a separate EXECUTE paragraph, he may list them together in a single EXECUTE paragraph, or he may group them in various ways in EXECUTE paragraphs.

Example

The statement given in the Example of Section 2.3.2 which requests the creation of the File 'STORFILE1' constitutes an EXECUTE paragraph when enclosed by the system names: "EXECUTE:" and "END".
h. Descriptions

A description is a system name, followed by a sequence of paragraphs, followed by another system name.

A description consists of a sequence of tasks that are to be executed, together with the specification of all the necessary structures for executing the tasks. For a description to be meaningful, all structures referred to in the task statements must be specified in the description.
4.1 DESCRIPTION's

DESCRIPTION [ ( description name ) ]:

DESCRIBE paragraph

.: EXECUTE paragraph

[ DESCRIBE paragraph

.: EXECUTE paragraph ]

DESCRIPTIONEND

(i) the optional parameter, description name, is a user-defined name.

(ii) DESCRIBE paragraph is a paragraph with the format described in Section 3.1.

(iii) EXECUTE paragraph is a paragraph with the format described in Section 3.2.

A DESCRIPTION is used to specify completely the information needed to execute one or more tasks. Each EXECUTE paragraph in a DESCRIPTION must be preceded
by DESCRIBE paragraphs containing specifications of the structures needed to execute the tasks specified in the EXECUTE paragraph.

(i) description name. This optional parameter allows the user to assign a name to the DESCRIPTION. The description name may also be used for comments.

(ii) DESCRIBE paragraph. This parameter is used to specify the structures needed to execute the tasks specified by parameter (iii). The structures needed may be specified in as many DESCRIBE paragraphs as the user desires.

(iii) EXECUTE paragraph. This parameter is used to specify the tasks to be executed. DESCRIBE paragraphs containing the specifications of structures needed to execute the tasks may occur in any order, so long as they precede the EXECUTE paragraph in question.

To specify completely that the task specified in the Example of Section 2.3.2 (to create the File 'STORFILE1'), the DESCRIBE paragraph given in the Example of Section 3.1, a DESCRIBE paragraph containing the specification of the External State 'EXTERS', and the ASSOCIATE statement, 'ASSOCERS', and the EXECUTE paragraph given in the Example of Section 3.2 must enclosed by the system names: "DESCRIPTION:" and "DESCRIPTIONEND".
PART II EXAMPLES

The set of examples presented here demonstrate the descriptive power of the DDL. Each example illustrates a different application of the DDL:

Example 1 illustrates the interfacing of files with new programs and programming languages.
Example 2 illustrates the interfacing of files with new operating systems.
Example 3 illustrates the reorganizing of files.

The data and storage structures selected for use in these examples are actual structures which have been implemented and are currently in use.
Example 1.

Use of the DDL to Convert Files for Interfacing with New Programs and Programming Languages

This example demonstrates how the DDL can be used to describe the conversion of a file created by a program written in an Assembly Language to a form in which the data contained in the file can be used by a COBOL program. The records to be converted are records produced by the Extended Data Management Facility (EDMF) of the University of Pennsylvania. This conversion requires the description of a set of data in a complex structure at the record level.

To perform this conversion, all that the DDL processor requires is:

1) a DDL description of the organization of the EDMF records;
2) a DDL description of an organization which can be processed by a COBOL program;
3) a DDL description of the relationships between values in the COBOL record and those values in the EDMF record; and
4) the EDMF records.

The conversion process for this example is illustrated in Figure 1-1.

The descriptions of the EDMF record organization, the COBOL record organization and the relationship description are discussed separately.
1.1 The EDMF Record Organization

The EDMF provides its users with a very comprehensive record organization feature. Data in the EDMF consists of two parts: the first part is called an attribute and the second part is called a value. An EDMF record is a collection of such attribute-value pairs. These pairs can be organized into a hierarchic structure in which attributes may repeat, and occur optionally, and in which values may have fixed or variable sizes and may be interpreted as numeric or alphanumerical strings.

Figure 1-1 EDMF to COBOL Record Conversion
In this example each of the user's EDMF records is to contain data on a book. This data will include:

i) a code number for each book. The code number has the attribute CODE NUMBER. This attribute occurs exactly once in each record. The values have a fixed size of 8 bytes and are stored as an alphanumeric string.

ii) the authors' names. An author has the attribute AUTHOR. This attribute must occur one or more times in each record. The values for this attribute have variable sizes and are stored as alphanumeric strings.

An example of a collection of attribute-value pairs satisfying this description is given in Figure 1-2.

```
CODE NUMBER, BINER540
AUTHOR, BIVENS, R.L.
AUTHOR, METROPOLIS, N.
```

Figure 1-2 Example of a Collection of Attribute-Value Pairs Forming an EDMF Record

These attribute-value pairs are stored in the organization illustrated in Figure 1-3.
<table>
<thead>
<tr>
<th>3 BYTES</th>
<th>SIZE OF RECORD</th>
</tr>
</thead>
<tbody>
<tr>
<td>5 BYTES</td>
<td>REFERENCE NUMBER, UNPACKED</td>
</tr>
<tr>
<td>1 BYTE</td>
<td>CONTROL INFORMATION</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>3 BYTES</th>
<th>LENGTH OF ATTR.-VALUE ENTRY</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 BYTE</td>
<td>CONTROL INFORMATION</td>
</tr>
<tr>
<td>1 BYTE</td>
<td>NUMBER OF DIRECTORY LISTS</td>
</tr>
<tr>
<td>2 BYTES</td>
<td>LENGTH OF ATTRIBUTE</td>
</tr>
<tr>
<td>VARIABLE</td>
<td>ATTRIBUTE</td>
</tr>
<tr>
<td>3 BYTES</td>
<td>LENGTH OF VALUE</td>
</tr>
<tr>
<td>VARIABLE</td>
<td>VALUE</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>3 BYTES</th>
<th>LENGTH OF ATTR.-VALUE ENTRY</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 BYTE</td>
<td>CONTROL INFORMATION</td>
</tr>
<tr>
<td>1 BYTE</td>
<td>NUMBER OF DIRECTORY LISTS</td>
</tr>
<tr>
<td>2 BYTES</td>
<td>LENGTH OF ATTRIBUTE</td>
</tr>
<tr>
<td>VARIABLE</td>
<td>ATTRIBUTE</td>
</tr>
<tr>
<td>3 BYTES</td>
<td>LENGTH OF VALUE</td>
</tr>
<tr>
<td>VARIABLE</td>
<td>VALUE</td>
</tr>
</tbody>
</table>

Figure 1-3  EDMF Record

The length specified in the 3 byte "Size of Record" entry includes the 9 byte Header size.

It is assumed that the user's EDMF records will be passed to the DDL processor on cards in this organization. The DDL processor must be able to extract the values from these records that are to be organized for use with the COBOL program. To do this the DDL processor requires a DDL description of the EDMF records, and specifically, of the location in the EDMF organization of those values to be converted.
The DDL description of the EDMF records is given below. Following the DDL description are explanations of the DDL statements appearing in the DDL description (as needed).

**DDL Description of EDMF Records:**

1. **RECORD ('BOOK EDMF', 'BOOK GROUP')**

2. **GROUP ('BOOK GROUP', SPEC;**
   - ('HEADER', M, F, 1),
   - ('DATA1', M, F, 1),
   - ('DATA2', M, V, NOLIM)

3. **GROUP ('HEADER', SPEC;**
   - ('RCD SIZE', M, F, 1),
   - ('REF NO', M, F, 1),
   - ('CONTROL INFO', M, F, 1)

4. **FIELD ('RCD SIZE', F, 'E', 3, B)

5. **FIELD ('REF NO', F, 'E', 5, 'E')

6. **FIELD ('CONTROL INFO', F, 'E', 1, B)

7. **GROUP ('DATA1', SPEC;**
   - ('A-V HEADER', M, F, 1),
   - ('CODE ENTRY', M, F, 1)

8. **GROUP ('A-V HEADER', SPEC;**
   - ('A-V LENGTH', M, F, 1),
   - ('CONTROL INFO', M, F, 1),
   - ('DIREC NO', M, F, 1)

9. **FIELD ('A-V LENGTH', F, 'E', 3, B)

10. **FIELD ('DIREC NO', F, 'E', 1, 'E')
11. GROUP ( 'CODE ENTRY', SPEC; 
    ( 'ATT LENGTH', M, F, 1 ),
    ( 'CODE NUMBER ATTRIB', M, F, 1 ),
    ( 'VAL LENGTH', M, F, 1 ),
    ( 'CODE NUMBER', M, F, 1 ) )

12. FIELD ( 'ATT LENGTH', F, 'E', 2, B )
13. FIELD ( 'CODE NUMBER ATTRIB', F, 'E', 11, 'E' )
14. FIELD ( 'VAL LENGTH', F, 'E', 3, 'E' )
15. FIELD ( 'CODE NUMBER', F, 'E', 8, 'E' )

16. GROUP ( 'DATA2', SPEC; 
    ( 'A-V HEADER', M, F, 1 ),
    ( 'AUTH ENTRY', M, F, 1 ) )

17. GROUP ( 'AUTH ENTRY', SPEC; 
    ( 'ATT LENGTH', M, F, 1 ),
    ( 'AUTH ATTRIB', M, F, 1 ),
    ( 'VAL LENGTH', M, F, 1 ),
    ( 'AUTHOR', M, F, 1 ) )

18. FIELD ( 'AUTH ATTRIB', F, 'E', 6, 'E' )
19. FIELD ( 'AUTHOR', V, 'E', 'VAL LENGTH' OF 'AUTH ENTRY', 'E' )
20. CHAR ( 'E', 8, EBCDIC )

Explanations:

Statement 19. In this FIELD statement the length parameter is given indirectly. For each occurrence of the field 'AUTHOR' the length of the value is the value of the field 'VAL LENGTH' in the group 'AUTH ENTRY' in which the value of 'AUTHOR' occurs.
For a COBOL program to use the data described in part 1 several modifications must be made:

(i) all variable length values must be converted to fixed length.
(ii) all groups and fields which may be repeated an unlimited number of times must be converted to groups which repeat no more than a fixed maximum number of times, where the number of times the group or field repeats is stored as a variable in the record.
(iii) all unnecessary fields may be eliminated.

Thus, each COBOL record must include the following data:

(i) a code number. The code number appears exactly once in each record. It has a fixed size of 8 bytes and is stored as an alphanumeric string.
(ii) the authors' names. There may be no more than three authors' names in each record. Each author's name has a fixed size of 20 bytes and is stored as an alphabetic string.
(iii) a counter to contain the number of authors' names.

The organization of these values is illustrated in Figure 1-4.
The COBOL description of such a record follows:

01 BOOK-RECORD.
   02 DATA-2-COUNTER PICTURE IS S9999 USAGE IS COMPUTATIONAL.
   02 CODE-NUMBER PICTURE IS X(8).
   02 DATA-2 OCCURS 3 TIMES DEPENDING ON DATA-2-COUNTER.
      03 AUTHOR PICTURE IS X(20).

The DDL Processor will use the DDL description of the EDMF record to extract the values giving the code and the authors from each EDMF record.

The DDL description of the COBOL records is given below. Following the DDL description are explanations (as needed) of the DDL statements.
DDL Description of the COBOL Records:

1. RECORD ('BOOK COBOL', 'BOOK-GROUP')

2. GROUP ('BOOK-GROUP', SPEC;
   ( 'DATA-2-COUNTER', M, F, 1, 'COUNT3' ),
   ( 'CODE-NUMBER', M, F, 1 ),
   ( 'DATA-2', M, V, 'DATA-2-COUNTER' ) )

3. FIELD ('DATA-2-COUNTER', F, 'E', 2, 8 )

4. FIELD ('CODE-NUMBER', F, 'E', 8, 'E' )

5. GROUP ('DATA-2', SPEC; ( 'AUTH', M, F, 1 ) )

6. FIELD ('AUTH', F, 'E', 20, 'E' )

7. CRITERION ('COUNT3', 'COUNT3EXP';
   RPLVAL ( 'DATA-2-COUNTER', CONSTANT ( 3, 'E' ) ) )

8. CRITEX ('COUNT3EXP',
   ( 'DATA-2-COUNTER' ) LE ( CONSTANT ( 3, 'E' ) ) )

Explanations:

Statement 2. In this GROUP statement, the name 'COUNT3' refers
to a criterion used to check the value of the field 'DATA-2-COUNTER'.
If the value exceeds 3, it is replaced by the value 3. The number
of occurrences of the group 'DATA-2' is specified indirectly. It
is the value of the field 'DATA-2-COUNTER'.
1.3 The Relationship Description

For the DDL processor to be able to convert from the EDMF record format to COBOL record format, the user must specify where each value for the COBOL records is to be found in the EDMF record. This is specified in the single DDL statement given below:

ASSOCIATE ( 'ASSOC LIST',
( 'DATA-2-COUNTER', REP ( 'DATA 2' OF 'BOOK EDMF' ) ),
( 'CODE-NUMBER', 'CODE NUMBER' OF 'BOOK EDMF' ),
( 'AUTH', 'AUTHOR' OF 'AUTH ENTRY' OF
 'DATA 2' IN 'BOOK EDMF' ) )

This DDL Statement specifies that, given the values of an EDMF record organized as described in the structure 'BOOK EDMF':

(i) the number of times the group 'DATA 2' occurs is to be the value of the field 'DATA-2-COUNTER' in the COBOL record.

(ii) the value of the field 'CODE NUMBER' is the value of the field 'CODE-NUMBER' in the COBOL record.

(iii) the values of the repeating field 'AUTHOR' are to be the values of the repeating field 'AUTH' in the COBOL record. The number of values for 'AUTH' is fixed for each COBOL record by the value of the field 'DATA-2-COUNTER' to be equal to the number of values for 'AUTHOR' occurring in the EDMF record or 3, whichever is less. If, in the EDMF record, 'AUTHOR' has more than three values, these are lost.

These relationships are illustrated in Figure 1-5.
Figure 1-5  Relationships between fields in the COBOL record and fields in the EDMF record
The Complete Description of the Conversion

The DDL statements given in parts 1.1, 1.2 and 1.3 of this example are used by the DDL processor to convert the EDMF records to COBOL records.

It is assumed that the EDMF records are input as a card deck. The first card contains the string: START. The end card contains the string: END OF DATA. This input file is assigned the name: EDMF BOOK FILE.

As the records are converted they are output onto cards. The first card contains the string: START. The end card contains the string: END OF DATA. This output file is assigned the name: COBOL BOOK FILE.

The complete description for converting the EDMF file to a COBOL file is given below.

DDL Description of the Conversion description:

1. DESCRIBE:

2. STORFILE ('EDMF BOOK FILE', 'EFILE STRUCTURE')

3. SEQUEN ('EFILE STRUCTURE'; 'LINK ERECORDS')

4. LINK ('LINK ERECORDS'; 'EDMF RECORD', 'EDMF RECORD'; NOORD, F, 1)

5. STORRCD ('EDMF RECORD', 'BOOK EDMF', 'INPUT CARDS')

6. The DDL statements describing the EDMF records appear here.

7. CARDIN ('INPUT CARDS', 'CARD DECK')

8. STORFILE ('COBOL BOOK FILE', 'CFILE STRUCTURE')

9. SEQUEN ('CFILE STRUCTURE'; 'LINK CRECORDS')
10. LINK ( 'LINK CRECORDS'; 'COBOL RECORD', 'COBOL RECORD';
  NOORD, F, 1 )

11. STORRCD ( 'COBOL RECORD', 'BOOK COBOL', 'OUTPUT CARDS' )

12. The DDL statements describing the COBOL records

13. CARDCUT ( 'OUTPUT CARDS', 'CARD DECK 2' )

14. BBLOCK ( 'CARD DECK'; V, NOLIM; WHOLE, V, NOLIM, 1;
  SPEC; ( 'BOOK EDMF', M, NOLIM );
  HDR, CONSTANT ( START, 'E' );
  TLR, CONSTANT ( END OF DATA, 'E' ) )

15. BBLOCK ( 'CARD DECK 2'; V, NOLIM; WHOLE, V, NOLIM, 1;
  SPEC; ( 'BOOK COBOL', M, NOLIM );
  HDR, CONSTANT ( START, 'E' );
  TLR, CONSTANT ( END OF DATA, 'E' ) )

16. END

17. EXECUTE:

18. CREATE ( 'COBOL BOOK FILE', 'EDMF BOOK FILE', 'ASSOC LIST' )

19. END
Example 2.

Use of the DDL to Convert Files for Interfacing with New Operating Systems

This example demonstrates how the DDL can be used to describe the conversion of a file created under one operating system to a form in which it can be accessed under a different operating system. The data to be converted is in the form of fixed length records stored on tape by the Sequential Access Method (SAM) of the RCA SPECTRA 70/46 Time Sharing Operating System (TSOS). This file is to be converted into a file on disk that can be accessed by the Indexed Sequential Access Method (ISAM) of the RCA SPECTRA 70/46 Tape Disk Operating System (TDOS). This conversion requires the description of sets of data in complex structures at the file and storage structure levels.

To perform the conversion, the DDL processor requires:

1) a DDL description of the data structure of the SAM file and of its placement on tape;

2) a DDL description of the data structure of the ISAM file and of its placement on disk;

3) a DDL description of the relationships between data items in the SAM file and those data items in the ISAM file; and

4) the SAM file.

The conversion process is illustrated in Figure 2-1.

The descriptions of the SAM file, the ISAM file and the relationship description are discussed separately.
2.1 The TSOS SAM File Organization

The TSOS SAM File of this example is a file produced by a COBOL program. The file contains information on bills and is assigned the name BILL-FILE. Each record, called BILL-RECORD, is of fixed length with the organization shown in Figure 2-2.
The records are stored on tape. They are not ordered. The COBOL description of this file follows:

```
INPUT-OUTPUT SECTION.

FILE-CONTROL.

SELECT BILL-FILE
    ASSIGN TO SYS12 UTILITY
    ACCESS IS SEQUENTIAL

DATA DIVISION.
```
FILE SECTION.

FD BILL-FILE

RECORDING MODE IS F

LABEL RECORD IS STANDARD

DATA RECORD IS BILL-RECORD

01 BILL-RECORD.

02 VENDOR-NAME PICTURE IS X(30).

02 BILL-NO PICTURE IS X(10).

02 PART-NO PICTURE IS X(10).

02 SERIAL-NO PICTURE IS X(10).

02 AMT PICTURE IS X(6).

02 COST PICTURE IS X(6).

02 DATE PICTURE IS X(6).

02 DEPT-NO PICTURE IS X(2).

The file was produced using TSOS SAM, which produces a tape with the organization illustrated in Figure 2-3.

![Figure 2-3 Storage Image of BILL-FILE](image)

Each of the labels contains 80 bytes. Each physical block, 'BILL-BLK', on the tape has the organization illustrated in Figure 2-4.
Each 'BILL-BLK' contains 25 'BILL-RECORD' record occurrences.

The DDL description of the TSOE SAM File is given below. Following the description are explanations (as needed) of the DDL statements.

**DDL Description of the TSOE SAM File BILL-FILE:**

1. **STORFILE** ( 'BILL-FILE'; 'BILL-SEQ' )
2. **SEQUEN** ( 'BILL-SEQ'; 'BILL-LINK' )
3. **LINK** ( 'BILL-LINK'; 'STOR-BILL-RCD', 'STOR-BILL-RCD'; NOORD, F, 1 )
4. **STORRCD** ( 'STOR-BILL-RCD', 'BILL-RECORD', 'BILL-TAPEIN' )
5. DDL statements describing the record 'BILL-RECORD' appear here.
6. **TAPEIN** ( 'BILL-STATEIN'; 'BILL-STATE', 'BILL-STATE' )
7. BLOCK ( 'BILL-TAPE'; SPEC;
       ( 'BILL-BLK', M, NOLIM );
       HDR, 'STANDARD VOLUME LABEL';
       HDR, 'STANDARD FILE HDR LABEL';
       HDR, 'TM';
       TLR, 'STANDARD FILE TLR LABEL';
       TLR, 'TM';
       TLR, 'TM' )

8. BBLOCK ( 'BILL-BLK'; F, 2064;
       WHOLE F, 25, 1, SPEC; ( 'BILL-RECORD', M, NOLIM );
       HDR, 'KEY' )

9. RECORD ( 'STANDARD VOLUME LABEL', 'SVL-STRUC' )

10. GROUP ( 'SVL-STRUC', SPEC;
         ( 'VLABEL-ID', M, F, 1 ),
         ( 'VOL-NO', M, F, 1 ),
         ( 'VOL-SEGMENT', M, F, 1 ),
         ( 'VOL-SEC', M, F, 1 ),
         ( 'BLANK', M, F, 3 ),
         ( 'CODE', M, F, 1 ),
         ( 'ENDBLK', M, F, 1 ) )

11. FIELD ( 'VLABEL-ID', F, 'E', 3, 'E' )

12. FIELD ( 'VOL-NO', F, 'E', 3, 'E' )

13. FIELD ( 'VOL-SEGMENT', F, 'E', 6, 'E' )

14. FIELD ( 'VOL-SEC', F, 'E', 1, 'E' )

15. FIELD ( 'BLANK', F, 'E', 10, 'E' )

16. FIELD ( 'CODE', F, 'E', 10, 'E' )

17. FIELD ( 'ENDBLK', F, 'E', 29, 'E' )

18. CHAR ( 'E', 8, EBCDIC )
19. RECORD ( 'TM', 'TM-STRUC' )

20. GROUP ( 'TM-STRUC', SPEC;

   ( 'TAPEMARK', M, F, l )
)

21. FIELD ( 'TAPEMARK', F, 'E', l, B )

22. RECORD ( 'STANDARD FILE HDR LABEL', 'SFHL-STRUC' )

23. RECORD ( 'STANDARD FILE TLR LABEL', 'SFHL-STRUC' )

24. GROUP ( 'SFHL-STRUC', SPEC;

   ( 'FLABEL-ID', M, F, l ),
   ( 'FILE-NO', M, F, l ),
   ( 'FILE-ID', M, F, l ),
   ( 'FILE-SERIAL', M, F, l ),
   ( 'VOL-SEQ', M, F, l ),
   ( 'FILE-SEQ', M, F, l ),
   ( 'GEN-NO', M, F, l ),
   ( 'VERS-NO', M, F, l ),
   ( 'CREATE-DTE', M, F, l ),
   ( 'EXPIRE-DTE', M, F, l ),
   ( 'FILE-SEC', M, F, l ),
   ( 'BLK-CNT', M, F, l ),
   ( 'RESVD', M, F, l )
)

25. FIELD ( 'FLABEL-ID', F, 'E', 3, 'E' )

26. FIELD ( 'FILE-NO', F, 'E', l, 'E' )

27. FIELD ( 'FILE-ID', F, 'E', l, 'E' )

28. FIELD ( 'FILE-SERIAL', F, 'E', 6, 'E' )

29. FIELD ( 'VOL-SEQ', F, 'E', 4, 'E' )
30. FIELD ( 'FILE-SEQ', F, 'E', 4, 'E' )
31. FIELD ( 'GEN-NO', F, 'E', 4, 'E' )
32. FIELD ( 'VERS-NO', F, 'E', 2, 'E' )
33. FIELD ( 'CREATE-DTE', F, 'E', 6, 'E' )
34. FIELD ( 'EXPIRE-DTE', F, 'E', 6, 'E' )
35. FIELD ( 'FILE-SEC', F, 'E', 1, 'E' )
36. FIELD ( 'BLK-CNT', F, 'E', 6, 'E' )
37. FIELD ( 'RESVD', F, 'E', 20, 'E' )
38. RECORD ( 'KEY', 'KEY-STRUC' )
39. GROUP ( 'KEY-STRUC', SPEC;
    ( 'NAME-FLO', M, F, 1 ),
    ( 'VERSION-FLO', M, F, 1 ),
    ( 'RCD-COUNT', M, F, 1 ),
    ( 'CODE', M, F, 1 ) )
40. FIELD ( 'NAME-FLO', F, 'E', 4, 'E' )
41. FIELD ( 'VERSION-FLO', F, 'E', 1, 'E' )
42. FIELD ( 'RCD-COUNT', F, 'E', 3, 'E' )
43. FIELD ( 'CODE', F, 'E', 8, 'E' )

Explanations:

Statements 1 - 5. These statements describe the file and record structure of the file 'BILL-FILE'.

Statements 6 - 43. These statements describe the storage structure of the file 'BILL-FILE'.
The TDOS ISAM File Organization

For the file described in part 2.1 to be accessible to TDOS ISAM several modifications must be made:

(i) the records of the file must be ordered sequentially.
(ii) the records must be stored on disk under TDOS ISAM conventions.
(iii) hierarchic indices must be created.

The organization of the records of BILL-FILE does not need to be changed.

The new file will be assigned the name BILL-FL to conform with TDOS file naming conventions. BILL-FL will be organized as follows:

(i) the records will be ordered sequentially according to the values of the field: SERIAL-NO.
(ii) the records will be blocked into sentences. Each sentence will be the size of a disk track. To simplify the example, it will be assumed that no overflow tracks are to be reserved. 19 tracks per cylinder will be used to store records.
(iii) three levels of indices will be created:

1) a track index for each data cylinder. One track per cylinder of records will be reserved for an index. Each entry in this index will point to a sentence. Records are ordered sequentially within sentences. Entries are not blocked.

2) an index cylinder. To simplify the example, it will be assumed that only one index cylinder is needed. Each entry in this index will point to a track index of a data cylinder. Entries are blocked into sentences.
Each sentence will be the size of a disk track.

3) a track index for the index cylinder. One track of the index cylinder is reserved for an index. Each entry in this index will point to a sentence. Entries are not blocked.

The organization of BILL-FL is illustrated in Figure 2-5.
The RCA TDOS ISAM description of this file follows:

```
BILL-FL  DTFS  BLKSIZE=7241,
       KEYLEN=10,
       KEYLOC=51,
       PRIDEVT=DISK90,
       PRIDEX=0,
       RECFORM=FIXED,
       RECSIZE=80,
       UPINDEX=DISK90
```

The DDL processor will use the DDL description of the TDOS SAM file, BILL-FILE, to extract the records from the tape. Using the following DDL description, the DDL processor will store the records on disk in the desired sequence and will create the necessary indices.

The DDL description of the file BILL-FL is given below. Following the description are explanations (as needed) of the DDL statements.

**DDL Description of TDOS ISAM File BILL-FL:**

1. **STORFILE ( 'BILL-FL'; 'BILLS-SEQ', 'BILLS-INDEX' )**
2. **SEQUEN ( 'BILLS-SEQ'; 'BILLS-SEQ-LINK' )**
3. **LINK ( 'BILLS-SEQ-LINK'; 'STOR-BILLS-RCD', 'STOR-BILLS-RCD'; 'BILLS-SEQ-CRIT', F, 1 )**
4. **CRITERION ( 'BILLS-SEQ-CRIT', 'BSQ-CRITEX' )**
5. **CRITEX ( 'BSQ-CRITEX', ( 'BSQ-CR1' ) AND ( 'BSQ-CR2' ) )**
6. **CRITEX ( 'BSQ-CR1', ( 'SERIAL-NO' OF OCC ( 'BILL-RECORD', S ) )
            LT ( 'SERIAL-NO' OF OCC ( 'BILL-RECORD', T ) ) )**
7. CRITEX ( 'BSQ-CR2', ALLOC ( X1; NOT ( (( 'SERIAL-NO' OF OCC ( 'BILL-RECORD', X1 )) LT ('SERIAL-NO' OF OCC ( 'BILL-RECORD', T ))) ) AND (( 'SERIAL-NO' OF OCC ( 'BILL-RECORD', S )) LT ('SERIAL-NO' OF OCC ( 'BILL-RECORD', X1 )))) )

8. STORRCD ( 'STOR-BILLS-RCD'; 'BILL-RECORD', 'DISK-OUT' )

9. [The DDL statements describing the record 'BILL-RECORD' appear here.]

10. DIREC ( 'BILLS-INLIX'; 'BILLS-SEQ-LINK', 'BINDX', 'BX-PTR, BLOCK, 'D-DATA-SENTENCE' )

11. STORFILE ( 'BINDX'; 'BINDX-SEQ', 'BINDX-INDX' )

12. SEQUEN ( 'BINDX-SEQ'; 'BINDX-SEQ-LINK' )

13. LINK ( 'BINDX-SEQ-LINK'; 'STOR-BINDX-RCD', 'STOR-BINDX-RCD'; NOORD, F, 1 )

14. STORRCD ( 'STOR-BINDX-RCD', 'BINDX-RCD', 'DISK-OUT' )

15. [The DDL statements describing the record 'BINDX-RCD' appear here.]

16. DIREC ( 'BINDX-INDX'; 'BINDX-SEQ-LINK', 'BINDXX', 'BXX-PTR', BLOCK, 'D-INDEX-TRACK' )

17. STORFILE ( 'BINDXX'; 'BINDXX-SEQ', 'BINDXX-INDX' )

18. SEQUEN ( 'BINDXX-SEQ'; 'BINDXX-SEQ-LINK' )

19. LINK ( 'BINDXX-SEQ-LINK'; 'STOR-BINDXX-RCD', 'STOR-BINDXX-RCD'; NOORD, F, 1 )

20. STORRCD ( 'STOR-BINDXX-RCD', 'BINDXX-RCD', 'DISK-OUT' )

21. [The DDL statements describing the record 'BINDXX-RCD' appear here.]
The DDL statements describing the record 'BINDXXX-RCD' appear here.

20. DISKOUT ( 'DISK-OUT'; 'FILE-BLOCK;
     'INDEX-CYL', 'DATA-CYL';
     'I-INDEX-TRACK', 'I-INDEX-TRACK';
     'I-DATA-TRACK', 'I-INDEX-TRACK';
     'I-DATA-SENTENCE', 'I-DATA-SENTENCE' )

21. BLOCK ( 'FILE-BLOCK', SPEC;
     ( 'INDEX-CYL', 0, 0, 'CNT-CRIT' ),
     ( 'DATA-CYL', M, NOLIM ) )

22. CRITERION ( 'CNT-CRIT', 'CNT-CRITEX' )

23. CRITEX ( 'CNT-CRITEX',
     ( REP ( 'DATA-CYL' ) ) \text{GE} ( \text{CONSTANT} ( 2, 'E' ) ) )

24. BLOCK ( 'INDEX-CYL', SPEC;
     ( 'I-INDEX-TRACK', M, 0 ),
     ( 'I-DATA-TRACK', M, 18 ) )

25. BBLOCK ( 'I-INDEX-TRACK', F, 7294;
     \text{WHOLE}, F, 19, 1, PSEC;
     ( 'BINDXXX-RCD', M, 18 ) )
34. BLOCK ('I-DATA-TRACK', SPEC
     ( 'D-DATA-SENTENCE', M, 0 ))

35. BBLOCK ('I-DATA-SENTENCE', F, 7241;
          WHOLE, F, 210, l, SPEC;
          ( 'BINIXX-RCD', M, NOLIM ))

36. BLOCK ('DATA-CYL', SPEC;
     ( 'D-INDEX-TRACK', M, 0 ),
     ( 'D-DATA-TRACK', M, 0 ))

37. BBLOCK ('D-INDEX-TRACK', F, 7294;
          WHOLE, F, 19, l, SPEC;
          ( 'BINIXX-RCD', M, 18 ))

38. BLOCK ('D-DATA-SENTENCE', M, 0 )

39. BBLOCK ('D-DATA-SENTENCE', F, 7241;
          WHOLE, F, 90, l, SPEC;
          ( 'BILL-RECORD', M, NOLIM ))

Explanations:

Statements 1 - 9. These statements describe the organization of
record occurrences in BILL-FL.

Statements 10 - 15. These statements describe the track index for
each data cylinder.

Statements 16 - 21. These statements describe the index cylinder.

Statements 22 - 27. These statements describe the track index for
the index cylinder.

Statements 28 - 39. These statements describe the storage struc-
ture for BILL-FL.
2.3 The Relationship Description

For the DDL processor to be able to convert data from the TSOS SAM file organization to the TDOS ISAM file organization, the user must specify where each value for the ISAM record is to be found in the SAM record. This is specified in the single DDL statement given below:

```
ASSOCIATE ( 'BILL-ASSOC', ( 'BILL-RECORD-GROUP',
     'BILL-RECORD-GROUP' OF 'BILL-FL' ) )
```

where: 'BILL-RECORD-GROUP' refers to a GROUP statement which describes the structure of the record.

Since the same record organization is used for both the TSOS SAM file and the TDOS ISAM file, the record description is the same in the respective file descriptions.

2.4 The Complete Description of the Conversion

The DDL statements given in parts 2.1, 2.2 and 2.3 of this example are used by the DDL processor to convert the TSOS SAM file, BILL-FILE, into the TDOS ISAM file, BILL-FL.

The complete description for this conversion is given below.

**DDL Description of the Conversion:**

1. DESCRIBE:
2. The DDL description of the TSOS SAM file, BILL-FILE appear here.
3. The DDL description of the TDOS ISAM file, BILL-FL appear here.
4. The ASSOCIATE statement appears here.
5. END
6. EXECUTE:

7. CREATE ('BILL-FL', 'BILL-FILE', 'BILL-ASSOC')

8. END
Example 3.

Use of the DDL to Reorganize Files

This example demonstrates how the DDL can be used to describe the reorganization of a file in which it is necessary to split and to merge records. The file to be converted is in the form of variable length records stored on tape by the Sequential Access Method (SAM) of the RCA SPECTRA 70/46 Time Sharing Operating System (TSOS). It is assigned the name TITLE-FILE. This file is to be converted into a new file on tape that can be accessed by SAM. The new file is to be assigned the name AUTHOR-FILE.

To perform the conversion, the DDL processor requires:

1) a DDL description of the TITLE-FILE;
2) a DDL description of the AUTHOR-FILE;
3) a DDL description of the relationships between data items in the AUTHOR-FILE and those in the TITLE-FILE; and
4) the TITLE-FILE file to be converted.

The conversion process is illustrated in Figure 3-1.

The descriptions of the TITLE-FILE, the AUTHOR-FILE and the relationship description are discussed separately.
3.1 The TITLE-FILE Organization

The TITLE-FILE of this example consists of variable length records with the organization shown in Figure 3-2.

<table>
<thead>
<tr>
<th>REC-SIZE</th>
<th>TITLE</th>
<th>YEAR</th>
<th>AUTHOR(1)</th>
<th>AUTHOR(n)</th>
</tr>
</thead>
</table>

Figure 3-2 TITLE-RECORD

These records are organized sequentially by TITLE.

The file was produced using TSOS SAM, which produces a tape with the same organization as illustrated in Example 2 by Figure 2-3.
The DDL description of the file TITLE-FILE is given below.

DDL Description of TITLE-FILE:

1. **DESCRIBE:**

2. **STORFILE** ( 'TITLE-FILE', 'TF-STRUC' )

3. **SEQUEN** ( 'TF-STRUC'; 'TITLE-LINK' )

4. **LINK** ( 'TITLE-LINK'; 'STOR-TITLE-RCD', 'STOR-TITLE-RCD';

   'TITLE-CRIT', F, 1 )

5. **CRITERION** ( 'TITLE-CRIT', 'TITLE-CRITEX' )

6. **CRITEX** ( 'TITLE-CRITEX',

   ( 'T-CR1' ) AND ( 'T-CR2' ) )

7. **CRITEX** ( 'T-CR1', ( 'TITLE' OF OCC ( 'TITLE-RECORD', S ) )

   LT ( 'TITLE' OF OCC ( 'TITLE-RECORD', T ) )

8. **CRITEX** ( 'T-CR2', ALLOCC ( X1; NOT ( ( ( 'TITLE' OF OCC ( 'TITLE-RECORD', X1 ) ) LT

   ( 'TITLE' OF OCC ( 'TITLE-RECORD', T ) ) ) AND

   ( ( 'TITLE' OF OCC ( 'TITLE-RECORD', S ) ) LT

   ( 'TITLE' OF OCC ( 'TITLE-RECORD', X1 ) ) ) ) )

9. **STORRCD** ( 'STOR-TITLE-RCD', 'TITLE-RECORD', 'TITLE-TAPEIN' )

10. **RECORD** ( 'TITLE-RECORD', 'T-GROUP' )

11. **GROUP** ( 'T-GROUP', SPEC;

   ( 'REC-SIZE', M, F, 1 ),

   ( 'TY-GROUP', M, F, 1 ),

   ( 'AUTHOR', M, V, NOLIM ) )

12. **FIELD** ( 'REC-SIZE', F, 'E', 2, B;

    CONCODE ( CONSTANT ( 11, 'E' ) DELIM, PHX ) )
13. CHAR ( 'E', 8, EBCDIC )
14. GROUP ( 'TY-GROUP', SPEC;  
        ( 'TITLE', M, F, 1 ),  
        ( 'YEAR', M, F, 1 ) )
15. FIELD ( 'TITLE', VARIABLE, 'E', NOLIM, 'E';  
           CONCODE ( CONSTANT ( , , 'E' ) DELIM, PX ) )
16. FIELD ( 'YEAR', FIXED, 'E', 4, 'E' )
17. FIELD ( 'AUTHOR', VARIABLE, 'E', NOLIM, 'E';  
           CONCODE ( CONSTANT ( ; , 'E' ) DELIM, INX ) )
18. TAPEIN ( 'TITLE-TAPEIN'; 'TITLE-TAPE', 'TITLE-BLK' )
19. BLOCK ( 'TITLE-TAPE', SPEC;  
        ( 'TITLE-BLK', MANDATORY, NOLIM );  
        HDR, 'STANDARD VOLUME LABEL';  
        HDR, 'STANDARD FILE LABEL';  
        HDR, 'TM';  
        TLR, 'STANDARD FILE TLR LABEL';  
        TLR, 'TM';  
        TLR, 'TM' )
20. [DDL Description of HDR and TLR labels appear here.]
21. BLOCK ( 'TITLE-BLK'; F, 2064;  
          WHOLE, V, NOLIM, SPEC;  
          ( 'TITLE-RECORD', MANDATORY, NOLIM );  
          HDR, 'KEY' )
22. [DDL Description of HDR label appears here.]
23. END
3.2 The AUTHOR-FILE Organization

The AUTHOR-FILE is to consist of variable length records with the organization shown in Figure 3-3.

<table>
<thead>
<tr>
<th>REC-SIZE</th>
</tr>
</thead>
<tbody>
<tr>
<td>AUTHOR</td>
</tr>
<tr>
<td>TITLE(1)</td>
</tr>
<tr>
<td>YEAR</td>
</tr>
<tr>
<td>.</td>
</tr>
<tr>
<td>.</td>
</tr>
<tr>
<td>TITLE(m)</td>
</tr>
<tr>
<td>YEAR</td>
</tr>
</tbody>
</table>

Figure 3-3 AUTHOR-RECORD

These records are to be organized sequentially by AUTHOR.

This file is to have the same storage structure as the TITLE-FILE.

The DDL description of the file AUTHOR-FILE is given below.

DDL Description of AUTHOR-FILE:

1. DESCRIBE:
2. STORFILE ( 'AUTHOR-FILE', 'AF-STRUC' )
3. SEQUEN ( 'AF-STRUC'; 'AUTH-LINK' )
4. LINK ( 'AUTH-LINK'; 'STOR-AUTH-RCD', 'STOR-AUTH-RCD';
          'AUTH-CRIT', F, 1 )
5. CRITEX ( 'AUTH-CRITEX', ( 'A-CRL' ) AND ( 'A-CR2' ) )
6. CRITEX ( 'A-CRL', ( 'AUTHOR' OF OCC ( 'AUTHOR RECORD', S ) ) LT ( 'AUTHOR' OF OCC ( 'AUTHOR-RECORD', T ) ) )

7. CRITEX ( 'A-CR2', ALLOC ( X1;
        NOT ( ( ( 'AUTHOR' OF OCC ( 'AUTHOR-RECORD', X1 ) )
        LT ( 'TITLE' OF OCC ( 'AUTHOR-RECORD', T ) )
        AND ( ( 'AUTHOR' OF OCC ( 'AUTHOR-RECORD', S )
        LT ( 'AUTHOR' OF OCC ( 'AUTHOR-RECORD', X1 ) ) ) ) ) )

8. CRITERION ( 'AUTH-CRIT', 'AUTH-CRITEX' )

9. STORRCD ( 'STOR_AUTH_RCD', 'AUTHOR-RECORD', 'AUTHOR-TAPEOUT' )

10. RECORD ( 'AUTHOR-RECORD', 'A-GROUP' )

11. GROUP ( 'A-GROUP', SPEC;
        ( 'REC-SIZE', M, F, 1 ),
        ( 'REC-GROUP', M, F, 1 ) )

12. GROUP ( 'REC-GROUP', SPEC;
        ( 'AUTHOR', M, F, 1 ),
        ( 'TY-GROUP', M, V, NOLIM ) )

13. FIELD ( 'REC-SIZE', FIXED, 'E', 2, B;
        CONCODE ( CONSTANT ( 11, 'E' ) DELIM, PRX ) )

14. CHAR ( 'E', 8, EBCDIC )

15. FIELD ( 'AUTHOR', VARIABLE, 'E', NOLIM, 'E';
        CONCODE ( CONSTANT ( ; , 'E' ) DELIM, PTX ) )

16. GROUP ( 'TY-GROUP', SPEC;
        ( 'TITLE', M, F, 1 ),
        ( 'YEAR', M, F, 1 ) )

17. FIELD ( 'TITLE', VARIABLE, 'E', NOLIM, 'E';
        CONCODE ( CONSTANT ( ; , 'E' ) DELIM, PTX ) )
18. FIELD ( 'YEAR', FIXED, 'E', 4, 'E' )
19. TAPEOUT ( 'AUTHOR-TAPEOUT'; 'AUTHOR-TAPE',
   'AUTHOR-BLK' )
20. BLOCK ( 'AUTHOR-TAPE'; SPEC;
   ( 'AUTHOR-BLK', MANDATORY, NOLIM );
   HDR, 'STANDARD VOLUME LABEL';
   HDR, 'STANDARD FILE LABEL';
   HDR, 'TM';
   TLR, 'STANDARD FILE TLR LABEL';
   TLR, 'TM'; TLR, 'TM' )
21. DDL Description of HDR and TLR labels appear here.
22. BLOCK ( 'AUTHOR-BLK'; F, 2064;
   WHOLE, V, NOLIM, SPEC;
   ( 'AUTHOR-RECORD', MANDATORY, NOLIM );
   HDR, 'KEY' )
23. DDL Description of HDR label appears here.
24. END
3.3 The Relationship Description

For the DDL processor to be able to reorganize the data in TITLE-FILE to create the AUTHOR-FILE, the user must specify where each value for the AUTHOR-RECORD is to be found in the TITLE-RECORD. In TITLE-FILE, data is stored about books. For each book there is a record occurrence containing book title, year of publication and authors. In AUTHOR-FILE, data is to be stored about authors. For each author whose name occurs in TITLE-FILE there is to be a record occurrence containing the author's name and an entry for each book he has written containing the book's title and year of publication. These relationships are illustrated in Figure 3-4.

![Figure 3-4 Relationship Between Values in TITLE-RECORD and AUTHOR-RECORD occurrences](image-url)
These relationships are specified in the single DDL statement given below:

ASSOCIATE ( 'TITLE-AUTHOR',

   ( 'REC-SIZE', LENGTH ( 'REC-GROUP' OF 'AUTHOR-RECORD', C ) ),
   ( 'AUTHOR', DISJOIN ( 'AUTHOR' OF 'TITLE-RECORD', VARIABLE, NOLIM ) ),
   ( 'TY-YEAR', COMBINE ( 'TY-GROUP' OF 'TITLE-RECORD', VARIABLE, NOLIM, 'COMB-CRIT' ) ) )

CRITERION ( 'COMB-CRIT', 'COMB-CRITEX' )

CRITEX ( 'COMB-CRITEX' ,

   ( 'AUTHOR' OF OCC ( 'TITLE-RECORD', S ) ) )

EQ ( 'AUTHOR' OF OCC ( 'TITLE-RECORD', T ) )

3.4 The Complete Description of the Conversion

The DDL statements given in parts 3.1, 3.2 and 3.3 of this example are used by the DDL processor to convert TITLE-FILE into AUTHOR-FILE.

The complete description for this conversion is given below:

DDL Description of the Conversion:

1. [DDL paragraph describing TITLE-FILE appears here.]
2. [DDL paragraph describing AUTHOR-FILE appears here.]
3. DESCRIBE:
4. [ASSOCIATE statement appears here.]
5. END
6. EXECUTE:
7. CREATE ( 'AUTHOR-FILE', 'TITLE-FILE',
       'TITLE-AUTHOR-ASSOC' )

8. END