ASG-DataManager™ DL/I Interface: DOS

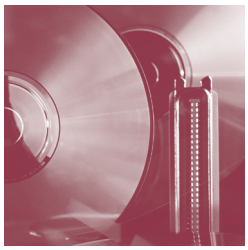
Version: 2.5

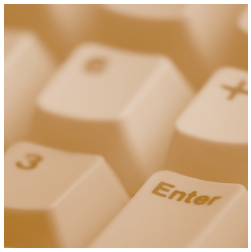
Publication Number: DMR0200-25-DLI

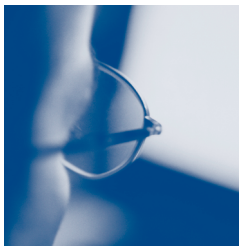
Publication Date: June 1982

The information contained herein is the confidential and proprietary information of Allen Systems Group, Inc. Unauthorized use of this information and disclosure to third parties is expressly prohibited. This technical publication may not be reproduced in whole or in part, by any means, without the express written consent of Allen Systems Group, Inc.

© 1998-2001 Allen Systems Group, Inc. All rights reserved.
All names and products contained herein are the trademarks or registered trademarks of their respective holders.







ASG Worldwide Headquarters Naples, Florida USA | asg.com
1333 Third Avenue South, Naples, Florida 34102 USA Tel: 941.435.2200 Fax: 941.263.3692 Toll Free: 1.800.932.5536

ASG Documentation/Product Enhancement Fax Form

Please FAX comments regarding ASG products and/or documentation to (941) 263-3692.

Company Name	Telephone Number	Site ID	Contact name

Product Name/Publication	Version #	Publication Date
Product:		
Publication:		
Tape VOLSER:		

Enhancement Request:

© 2001 Allen Systems Group, Inc.
All names and products are trademarks or registered trademarks of their respective holders.

ASG Support Numbers

ASG provides support throughout the world to resolve questions or problems regarding installation, operation, or use of our products. We provide all levels of support during normal business hours and emergency support during non-business hours. To expedite response time, please follow these procedures.

Please have this information ready:

- Product name, version number, and release number
- List of any fixes currently applied
- Any alphanumeric error codes or messages written precisely or displayed
- A description of the specific steps that immediately preceded the problem
- The severity code (ASG Support uses an escalated severity system to prioritize service to our clients. The severity codes and their meanings are listed below.)

If You Receive a Voice Mail Message:

1. Follow the instructions to report a production-down or critical problem.
2. Leave a detailed message including your name and phone number. A Support representative will be paged and will return your call as soon as possible.
3. Please have the information described above ready for when you are contacted by the Support representative.

Severity Codes and Expected Support Response Times

Severity	Meaning	Expected Support Response Time
1	Production down, critical situation	Within 30 minutes
2	Major component of product disabled	Within 2 hours
3	Problem with the product, but customer has work-around solution	Within 4 hours
4	"How-to" questions and enhancement requests	Within 4 hours

ASG provides software products that run in a number of third-party vendor environments. Support for all non-ASG products is the responsibility of the respective vendor. In the event a vendor discontinues support for a hardware and/or software product, ASG cannot be held responsible for problems arising from the use of that unsupported version.

Business Hours Support

Your Location	Phone	Fax	E-mail
United States and Canada	800.354.3578 1.941.435.2201 **Secondary Numbers:** 800.227.7774 800.525.7775	941.263.2883	support@asg.com
Australia	61.2.9460.0411	61.2.9460.0280	support.au@asg.com
England	44.1727.736305	44.1727.812018	support.uk@asg.com
France	33.141.028590	33.141.028589	support.fr@asg.com
Germany	49.89.45716.300	49.89.45716.400	support.de@asg.com
Singapore	65.224.3080	65.224.8516	support.sg@asg.com
All other countries:	1.941.435.2201		support@asg.com

Non-Business Hours - Emergency Support

Your Location	Phone	Your Location	Phone
United States and Canada	800.354.3578 1.941.435.2201 **Secondary Numbers:** 800.227.7774 800.525.7775 **Fax:** 941.263.2883		
Asia	011.65.224.3080	**Japan/Telecom**	0041.800.9932.5536
Australia	0011.800.9932.5536	**New Zealand**	00.800.9932.5536
Denmark	00.800.9932.5536	**South Korea**	001.800.9932.5536
France	00.800.9932.5536	**Sweden/Telia**	009.800.9932.5536
Germany	00.800.9932.5536	**Switzerland**	00.800.9932.5536
Hong Kong	001.800.9932.5536	**Thailand**	001.800.9932.5536
Ireland	00.800.9932.5536	**United Kingdom**	00.800.9932.5536
Israel/Bezeq	014.800.9932.5536		
Japan/IDC	0061.800.9932.5536	**All other countries**	1.941.435.2201

ASG Web Site

Visit http://www.asg.com, ASG's World Wide Web site.

Submit all product and documentation suggestions to ASG's product management team at http://www.asg.com/products/suggestions.asp

If you do not have access to the web, FAX your suggestions to product management at (941) 263-3692. Please include your name, company, work phone, e-mail ID, and the name of the ASG product you are using. For documentation suggestions include the publication number located on the publication's front cover.

Contents

Preface v
About this Publication v
Publication Conventions vi

1 DataManager DL/I Interface Facilities 1

2 The DL/I Environment and DataManager 5
Introduction 5
Segments 5
Databases 10
Application View 13
Further Information 14
Segments 14
DL/I Data Fields 15
DL/I Databases 16
Special DataManager Member Types 17
Application View 18

3 DataManager Data Definition Statements for a DL/I Environment 21
Introduction 21
DataManager Data Definition Statements for DL/I Segments 21
Outline of the SEGMENT Data Definition Statement 21
Specification of the Data Definition Statement for a Segment that Resides in a Physical Database 24
Specification of the Data Definition Statement for a SEGMENT that Resides in a Logical Database 42
Specification of the Data Definition Statement for a SEGMENT that Resides in a Secondary Index Database 46
DataManager Data Definition Statements for DL/I Databases 56
Outline of the DL/I-DATABASE Data Definition Statement 56
Specification of the Data Definition Statement for a HSAM Type DL/I Database 57
Specification of the Data Definition Statement for a HISAM Type DL/I Database 62
Specification of the Data Definition Statement for a HDAM Type DL/I Database 65
Specification of the Data Definition Statement for a HIDAM Type DL/I Database 69

Specification of the Data Definition Statement for a LOGICAL Type DL/I Database 74
Specification of the Data Definition Statement for a SECONDARY-INDEX Type DL/I Database 78
DataManager Data Definition Statements for DL/I Program Communication Blocks 82
DataManager System, Program, and Module Data Definition Statements for a DL/I Environment 91
Outline of the SYSTEM, PROGRAM, and MODULE Data Definition Statements for a DL/I Environment 91
Specification of the PROCESSES Clause 92

4 Extensions to DataManager Commands for DL/I Databases 99
Introduction 99
DL/I Member-type Keywords 99
Condition Keywords for Which and What Commands 102
Introduction 102
Examples 103
Member Type Interrogations 106
Interrogation Syntax 112
Alternative Verb Keywords 130

5 DL/I Source Language Generation from DataManager 131
Introduction 131
Generating DL/I DBD Control Statements 131
Generating DL/I PSB Control Statements 137
Generation of COBOL, PL/I, or Assembler Data Description Statements for Segment Input/Output Areas 140
The PRODUCE Command 140
Installation Macros 141
Segment Input/Output Areas: Items Defined as BINARY or BITS 141
Simple Physical Segments 142
Logical Child Segments 142
Destination Parent Segments 143
Index Target and Index Source Segments 143
Logical Segments and Logical Concatenated Segments 143
Variable Length Segments 144
Path Calls 145
Index Pointer Segments 145
Miscellaneous DL/I Fields 148
Generation of COBOL, PL/I, or Assembler Data Description Statements for Segment Sensitive Fields Input/Output Areas 148
Generation of COBOL, PL/I, or Assembler Data Description Statements for PCB Masks 150
Generation of COBOL, PL/I, or Assembler Data Description Statements for Segment Search Arguments 153

Appendix
Macros for Tailoring the DataManager DL/I Interface 159

Implementation of the DL/I Interface Macros 159
The Macros DGDBD and DGPSB 160
The Macros DGSCOB, DGSPLI, DGSBAL, and DGSREC 161

Index 163

Preface

This *ASG-DataManager DL/I Interface: DOS* describes the the DOS version of the DL/I Interface facility. This facility (additional to the basic set-up, maintenance, and interrogation features) enables the user fully to define DL/I databases in the dictionary and to produce DL/I DBD and PSB control statements, PCB masks, segment search arguments, and segment input/output area data description directly from ASG-DataManager (herein called DataManager) data definitions.

The scope of the DOS version of this interface encompasses the Data Language/I (DL/I) facility of the IMS/VS subsystem available under VS (excluding OS/VS).

The OS version of the interface is described in *ASG-DataManager DL/I Interface: OS*.

This interface does not include the Data Communications (DC) facility of IMS/VS, for which a separate interface is available.

Allen Systems Group, Inc. (ASG) provides professional support to resolve any questions or concerns regarding the installation or use of any ASG product. Telephone technical support is available around the world, 24 hours a day, 7 days a week.

ASG welcomes your comments, as a preferred or prospective customer, on this publication or on any ASG product.

About this Publication

This publication consists of these chapters:

- Chapter 1, "DataManager DL/I Interface Facilities," summarizes the interfaces between DataManager and DL/I.
- Chapter 2, "The DL/I Environment and DataManager," discusses very briefly the concept of DL/I databases and illustrates how a DL/I database can be defined to DataManager.
- Chapter 3, "DataManager Data Definition Statements for a DL/I Environment," gives the specifications of the DataManager data definition statements for DL/I databases and their constituents.
- Chapter 4, "Extensions to DataManager Commands for DL/I Databases," describes DataManager's interrogation and documentation facilities for reporting on DL/I databases.
- Chapter 5, "DL/I Source Language Generation from DataManager," describes the interface between DL/I and the DataManager source language generation facility.

Publication Conventions

These conventions apply to syntax diagrams that appear in this publication.

Diagrams are read from left to right along a continuous line (the "main path"). Keywords and variables appear on, above, or below the main path.

Convention	Represents
➤➤	At the beginning of a line indicates the start of a statement.
➤◄	At the end of a line indicates the end of a statement.
———➤	At the end of a line indicates that the statement continues on the line below.
➤———	At the beginning of a line indicates that the statement continues from the line above.

Keywords are in upper-case characters. Keywords and any required punctuation characters or symbols are highlighted. Permitted truncations are not indicated.

Variables are in lower-case characters.

Statement identifiers appear on the main path of the diagram:

```
➤———— COMMAND ————————————————————————————————➤
```

A required keyword appears on the main path:

```
➤———— COMMAND ——— KEYWORD ————————————————————➤
```

An optional keyword appears below the main path:

```
➤———— COMMAND ————————————————————————————————➤
              └── KEYWORD ──┘
```

Where there is a choice of required keywords, the keywords appear in a vertical list; one of them is on the main path:

```
➤———— COMMAND ———┌── KEYWORD1 ──┐
                 ├── KEYWORD2 ──┤————————————➤
                 └── KEYWORD3 ──┘
```

or

```
➤———— COMMAND ———┌── KEYWORD1 ──┐
                 ├── KEYWORD2 ──┤————————————➤
                 └── KEYWORD3 ──┘
```

Where there is a choice of optional keywords, the keywords appear in a vertical list, below the main path:

Convention	Represents

```
➤──────COMMAND──────┬──────────────────┬──────────────────────────────➤
                     │──KEYWORD1────────│
                     └──KEYWORD2────────┘
```

The repeat symbol, <<<<<<, above a keyword or variable, or above a whole clause, indicates that the keyword, variable, or clause may be specified more than once:

```
                          <<<<<<<<
➤──────COMMAND────────variable──────────────────────────────────────➤
```

A repeat symbol broken by a comma indicates that if the keyword, variable, or clause is specified more than once, a comma must separate each instance of the keyword, variable, or clause:

```
                          <<< , <<
➤──────COMMAND────────variable──────────────────────────────────────➤
```

The repeat symbol above a list of keywords (one of which appears on the main path) indicates that any one or more of the keywords may be specified; at least one must be specified:

```
                                  <<<<<<<<<<<<<<<<
➤──────COMMAND──────┬──────────────────┬──────────────────────────────➤
                     │──KEYWORD1────────│
                     └──KEYWORD2────────┘
```

The repeat symbol above a list of keywords (all of which are below the main path) indicates that any one or more of the keywords maybe specified, but they are all optional:

```
                                  <<<<<<<<<<<<<<<<
➤──────COMMAND──────┬──────────────────┬──────────────────────────────➤
                     │──KEYWORD1────────│
                     └──KEYWORD2────────┘
```

Allen Systems Group, Inc. uses these conventions in publications:

Convention	Represents
ALL CAPITALS	Directory, path, file, dataset, member, database, program, command, and parameter names.
Initial Capitals on Each Word	Window, field, field group, check box, button, panel (or screen), option names, and names of keys. A plus sign (+) is inserted for key combinations (e.g., Alt+Tab).
`lowercase italic monospace`	Information that you provide according to your particular situation. For example, you would replace *`filename`* with the actual name of the file.
`Monospace`	Characters you must type exactly as they are shown. Code, JCL, file listings, or command/statement syntax. Also used for denoting brief examples in a paragraph.

1 DataManager DL/I Interface Facilities

The DL/I Interface in DataManager provides facilities for all users in an DL/I environment. It enables users to perform these tasks:

- Define DL/I databases and segments to DataManager (in a simpler manner than that available from the use of DL/I Database Description Control Statements); to hold the definitions in the data dictionary; and to document them, to interrogate them and to process them by the standard DataManager commands
- Generate from the data dictionary and to insert into the required source library complete sets of Database Description (DBD) Control Statements to allow a DBD generation process
- Define at SYSTEM/PROGRAM/MODULE data definition level and Program Communication Block (PCB) data definition level the application view of the databases used by programs
- Generate from the data dictionary and to insert into the appropriate source library complete sets of Program Specification Block (PSB) Control Statements to allow a PSB generation process
- Generate record layouts and/or COBOL, PL/I, or Assembler data descriptions for segment input/output areas
- Generate record layouts and/or COBOL, PL/I, or Assembler data descriptions for segment input/output areas for sensitive fields
- Generate record layouts and/or COBOL, PL/I, or Assembler data descriptions for Program Communication Block (PCB) masks
- Generate record layouts and/or COBOL, PL/I, or Assembler data descriptions for segment search arguments (SSAs)

The ability to define DL/I databases, segments, and Program Communication Blocks demands three additional member types in DataManager:

- To define a database, the member type is DL/I-DATABASE. In the DataManager member type hierarchy, this database member type is at the same level as the FILE member type.
- To define a Program Communication Block, the member type is PROGRAM-COMMUNICATION-BLOCK or PCB, which comes between the MODULE member type and the DL/I-DATABASE member type in the DataManager member type hierarchy. The two member type identifiers PROGRAM-COMMUNICATION-BLOCK and PCB are synonymous.
- To define a segment, the member type is SEGMENT, which comes between the DL/I-DATABASE member type and the GROUP member type in the DataManager member type hierarchy.

The DL/I-DATABASE data definition statement, the PROGRAM-COMMUNICATION-BLOCK/PCB data definition statement and the SEGMENT data definition statement are discussed further in Chapter 2, "The DL/I Environment and DataManager," on page 5 and are specified in Chapter 3, "DataManager Data Definition Statements for a DL/I Environment," on page 21.

Also required are facilities at the SYSTEM, PROGRAM, and MODULE data definition levels to allow the application view of databases to be specified. The relevant formats of the SYSTEM, PROGRAM, and MODULE data definition statements are discussed in Chapter 2, "The DL/I Environment and DataManager," on page 5 and are specified in Chapter 3, "DataManager Data Definition Statements for a DL/I Environment," on page 21.

To enable the definitions of DL/I databases, Program Communication Blocks and segments to be processed by DataManager in the same way as other members of the data dictionary, the keywords DL/I-DATABASES, PROGRAM-COMMUNICATION-BLOCKS, PCBS, and SEGMENTS are added to the member-type keywords available for use in these basic DataManager commands:

- BULK
- GLOSSARY
- LIST
- PERFORM
- WHICH

Any of the alternative forms DL/I-DATABASES, DLI-DATABASES, and DLI-DATABASES are accepted for the keyword DL/I-DATABASES.

These keywords are also added to the commands to enable interrogation and documentation in respect of members of internal member types:

- SEQUENCE-KEYS
- DL/I-DATASETS (with the alternative forms DL/1-DATASETS, DLI-DATASETS, or DL1-DATASETS)
- INDEX-SEARCH-FIELDS
- SYSTEM-RELATED-FIELDS
- CONCATENATED-KEY-NAMES

These members are generated by DataManager (see "Special DataManager Member Types" on page 17). As members of internal types have no source records, a BULK ENCODE or BULK PRINT command selecting members of these types is meaningless.

Other extensions to the syntax of basic DataManager interrogation and documentation commands provide powerful facilities for reporting on the structure of DL/I database systems. These facilities are specified in Chapter 4, "Extensions to DataManager Commands for DL/I Databases," on page 99.

The ability to generate DL/I Control Statements, data descriptions for segment input/output areas, PCB masks, and segment search arguments requires the use of the Source Language Generation facility (selectable unit DMR-SL5). The fundamentals of the Source Language Generation facility, including the output of data descriptions in COBOL, PL/I, and Assembler, are described in the *ASG-Manager Products Source Language Generation* publication.

Enhancements to the Source Language Generation facility which enable it to output DL/I Control Statements and COBOL, PL/I, and Assembler data descriptions for segment input/output areas, PCB masks, and segment search arguments are specified in Chapter 5, "DL/I Source Language Generation from DataManager," on page 131.

For an installation that is implementing an DL/I database management system for the first time, ASG recommends this approach:

- Study in depth the concepts and facilities both of DL/I and of DataManager.
- Design the DL/I database structures required for the initial implementation.
- Set up a DataManager data dictionary in which the definitions of the data structures and the application views can be developed.
- Write DataManager data definitions of the databases, the segments, and the constituent groups and items, and add them to the data dictionary.
- Add program and module data definitions and PCB members for the application views.
- Using DataManager's Source Language Generation facility, generate the DL/I Control Statements and the data descriptions for segment input/output areas, PCB masks, and segment search arguments.

You will find that this approach is simple and offers additional checks on accuracy over implementation using DL/I facilities alone.

2 The DL/I Environment and DataManager

Introduction

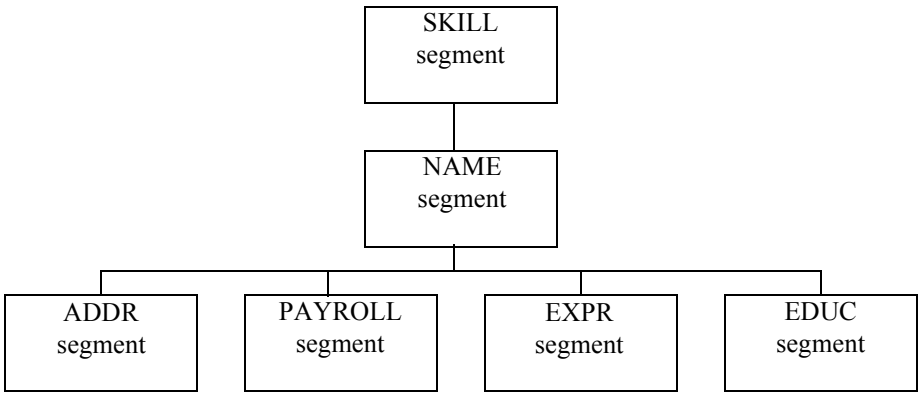
Segments

One of the fundamental concepts of DL/I is that it is not the physical organization of the data that is significant, but rather the logical structures of the data as viewed by specific applications.

The basic element of data in a DL/I environment is the segment. Regardless of where or how segments are physically stored, a DL/I database system is effectively a logical collection of segments, which happen to occur in one or more physical databases, some or all of which are required for specific applications.

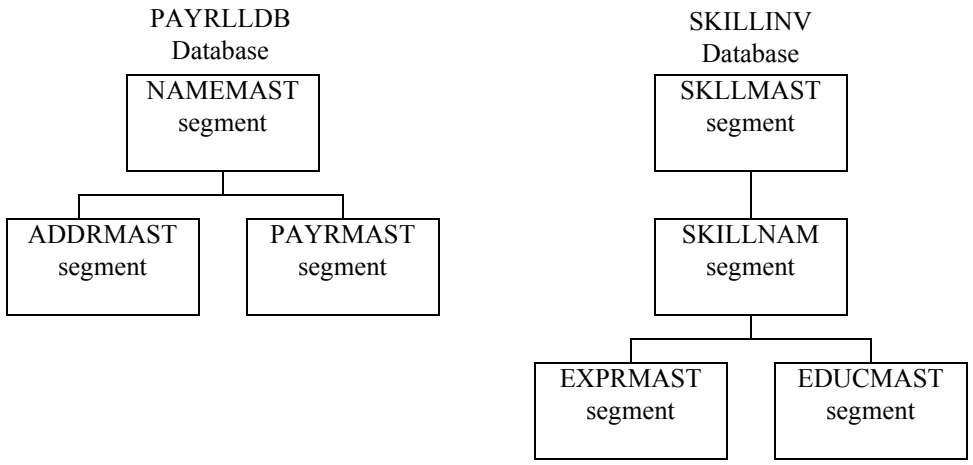
Figure 1 illustrates the concept of a logical data structure for an Employee database, named SKILLEMP.

Figure 1. Logical Structure of an Employee Database, SKILLEMP



However, the six segments in Figure 1 on page 5 may actually represent segments stored in one or more physical databases. If, for example, the six segments were stored in two physical databases, one a Payroll database and the other a Skills Inventory database, then Figure 2 indicates a possible hierarchical structure of the segments within their physical databases, linked by the segment SKILLNAM.

Figure 2. Physical Storage of the Employee Database, SKILLEMP



Using the DataManager DL/I Interface, each of the segments shown in Figure 1 on page 5 and Figure 2 can be defined as a data dictionary member of a member type called SEGMENT.

If certain assumptions are made regarding the specific attributes of the segments, then this would be the method of using DataManager data definition statements to define these segments:

- For the segments in Figure 1 on page 5:

```
ADD SKILL;
SEGMENT LOGICAL
CQNTAINS SKLLMAST
;
ADD NAME;
SEGMENT LOGICAL
CONTAINS SKILLNAM,NAMEMAST
;
ADD ADDR;
SEGMENT LOGICAL
CONTAINS ADDRMAST
;
ADD PAYROLL;
SEGMENT LOGICAL
CONTAINS PAYRMAST
;
ADD EXPR;
SEGMENT LOGICAL
CONTAINS EXPRMAST
;
ADD EDUC;
SEGHENT LOGICAL
CONTAINS EDUCMAST
;
```

- For the segments in Figure 2 on page 6:

```
ADD NAMEMAST;
SEGMENT PHYSICAL
RELATED-AS DESTINATION-PARENT
ATTRIBUTES
     CONTAINS INITIAL,SURNAME,SEX
     FREQUENCY 100
     SEQUENCE-KEY SURNAME DUPLICATED
     INSERT-POSITION LAST
;
ADD ADDRMAST;
SEGHENT PHYSICAL
ATTRIBUTES
     CONTAINS HOUSE,STREET,TOWN,COUNTY
     INSERT-POSITION LAST
;
ADD PAYRMAST;
SEGMENT PHYSICAL
ATTRIBUTES
     CONTAINS PAYRNUMB,STATUS,RATE
     SEQUENCE-KEY PAYRNUMB UNIQUELY
```

```
;
ADD SKLLMAST;
SEGMENT PHYSICAL
ATTRIBUTES
     CONTAINS SKLLCODE,SKLLTYPE
     SEQUENCE-KEY SKLLCODE UNIQUELY
;
ADD SKILLNAM;
SEGMENT PHYSICAL
RELATED-AS UNDIRECTIONAL-CHILD TO NAMEMAST
ATTRIBUTES
     SEQUENCE-KEY EMPLOYEE-NO UNIQUELY
     CONTAINS EMPLOYEE-NO
;
ADD EXPRMAST;
SEGMENT PHYSICAL
ATTRIBUTES
     CONTAINS EXPRCDDE,EXPRTIME
     INSERT-POSITION FIRST
;
ADD EDUCMAST;
SEGMENT PHYSICAL
ATTRIBUTES
     CONTAINS QUALCODE
     SEQUENCE-KEY QUALCODE UNIQUELY
;
```

Another feature of DL/I is the secondary indexing facility. Briefly, this enables the user to access a segment in a physical or logical database, based on data located in one of its dependent segments; and also optionally to process the database as if its structure has been inverted, with the segment being accessed as the root of the structure. In a secondary index database, an occurrence of an index pointer, segment is generated for each occurrence of the index source segment containing the search-field data, on which accessing the index target segment is to be based.

Figure 3 on page 9 illustrates the concept of secondary indexing for an Automobile Register database.

Figure 3. An Example of Secondary Indexing

Using the DataManager DL/I Interface, each of the segments shown in Figure 3 on page 9 can be defined as a data dictionary SEGMENT type member.

If certain assumptions are made regarding the specific attributes of the segments, then this would be the method of using DataManager data definition statements to define these segments:

```
ADD CITY;
SEGMENT PHYSICAL
ATTRIBUTES
    CONTAINS CITYNAME,STATE,CITYCODE
    SEQUENCE-KEY CITYCODE UNIQUELY
;
ADD NAMEID;
SEGMENT PHYSICAL
RELATED-AS TARGET-SEGMENT
ATTRIBUTES
    CONTAINS INITIAL,SURNAME,IDENTCDE
    SEQUENCE-KEY IDENTCDE UNIQUELY
;
ADD AUTOMBLE;
SEGMENT PHYSICAL
RELATED-AS SOURCE-SEGMENT
ATTRIBUTES
    CONTAINS MODEL,COLOR,WEIGHT
    INSERT-POSITION LAST
;
ADD COLORSEG;
SEGMENT INDEX-POINTER
RELATED-TO NAMEID ON COLCODE
    SOURCE AUTOMBLE
    SEARCH-KEY-FIELD COLOR
ATTRIBUTES
    SEQUENCE-KEY COLORTYP
;
```

In this example, COLCODE is the name of the search-field (XDFLD) that can be used in the segment search argument for the calls issued to DL/I to access the index target segment.

Databases

As indicated in "Segments" on page 5, an essential feature of an DL/I database system is the ability to overlay multiple logical data structures on non-repetitive physical data structures, where the logical data structures are designed in a manner that satisfies the functional requirements of specific applications. Logical databases (using logical relationships specified for segments of physical databases) define structural relationships among actual segments of one or more physical databases, which can differ from the structural relationships in the physical database(s). Segments from any given physical database can belong to many logical databases.

DL/I also offers the facility to access segments in physical or logical databases, in a sequence specified by a secondary index database.

In "Segments" on page 5 it was shown how DataManager SEGMENT data definition statements are used to define the characteristics and the logical or secondary indexing relationships of segments.

Data definition statements for a data dictionary member type called DL/I-DATABASE are used to define the access and organization methods of the databases to DataManager, and to specify the hierarchy of the segments that they contain.

If certain assumptions are made regarding the specific attributes of the databases shown in Figure 1 on page 5, Figure 2 on page 6, and Figure 3 on page 9, respectively, then this would be the method of using DataManager data definition statements to define those databases:

- For the database in Figure 1 on page 5:

```
ADD SKILLEMP;
DL/I-DATABASE LOGICAL
     CONTAINS SKILL,
              NAME PARENT SKILL,
              ADDR PARENT NAME,
              PAYROLL PARENT NAME,
              EXPR PARENT NAME,
              EDUC PARENT NAME
;
```

- For the databases in Figure 2 on page 6:

```
ADD PAYRLLDB;
DL/I-DATABASE HDAM
DATASETS PRIME PAYRF BLOCK 1024
         DEVICE 3340
     CONTAINS NAMEMAST,
              ADDRMAST PARENT NAMEMAST,
              PAYRMAST PARENT NAMEMAST
;
ADD SKILLINV;
DL/I-DATABASE HISAM
DATASETS PRIME SKLLF BLOCK 2048
         DEVICE 3340
     CONTAINS SKLLMAST,
              SKILLNAM PARENT SKLLMAST,
              EXPRMAST PARENT SKILLNAM,
              EDUCMAST PARENT SKILLNAM
;
```

- For the databases in Figure 3 on page 9:

```
ADD AUTOREG;
DL/I-DATABASE HDAM
RANDOMIZING-MODULE AUTRTNE
            ANCHOR-POINTS 1
            RELATIVE-BLOCK-MAXIMUM 500
            INSERTION-BYTES-MAXIMUM 824
            DATASETS PRIME AUTOF BUFFER 1648
                                 DEVICE 2314
                                 SCAN      5
     CDNTAINS CITY,
              NAMEID PARENT CITY,
              AUTOMBLE PARENT NAMEID
```

```
;
ADD AUTOCOL;
DL/I-DATABASE SECONDARY-INDEX
DATASETS PRIME COLORF BUFFER 4
         OVERFLOW COLORFO BUFFER 4
         DEVICE 2314
     CONTAINS COLORSEG
;
```

Application View

Finally, when specifying an DL/I database system, the different applications view the databases and segments where access must be defined. This must be done before a DL/I application program can issue calls to DL/I to access the databases.

Views are defined in the data dictionary by using these DataManager DL/I Interface language facilities:

- PROGRAM-COMMUNICATION-BLOCK or PCB member type; a member of this type defines a Program Communication Block accessed by an application program.
- PROCESSES clause which lists the PCB members relevant to the application; this is inserted in the data definition statements for SYSTEM, PROGRAM, and MODULE members and it enables these statements:
  - Program Specification Block (PSB) control statements for an application to be produced from the listed PCB members.
  - SEGMENT-SEARCH-ARGUMENT (SSA) statements to be defined to the data dictionary. These statements can be used by the Source Language Generation facility when generating DBD Control Statements (see "Application View" on page 18 and "Generating DL/I DBD Control Statements" on page 131).

Generating Segment Search Arguments (SSAs) and Program Communication Block (PCB) masks are described in "Generation of COBOL, PL/I, or Assembler Data Description Statements for PCB Masks" on page 150 and "Generation of COBOL, PL/I, or Assembler Data Description Statements for Segment Search Arguments" on page 153 respectively.

If certain assumptions are made, then this would be the method of using the PROCESSES clause to describe an application's processing of the databases SKILLEMP and AUTOREG illustrated in Figure 1 on page 5 and Figure 3 on page 9, respectively:

```
PROCESSES DL/I
    CONTAINS SKILLEMP-PCB, AUTOREG-PCB
        SEGMENT-SEARCH-ARGUMENTS
        SEGMENT SKILL USED-IN SKILL-SSA
                              COMMAND-CODE FIRST-OCCURRENCE
                              QUALIFIED-ON SKLLTYPE EG
        SEGMENT EXPR USED-IN EXPR-SSA
                              QUALIFIED-ON EXPRCODE EG
                              AND EXPRTIME GT
        SEGMENT NAMEID USED-IN NAMEID-SSA
                              COMMAND-CODE LAST-OCCURRENCE
                              QUALIFIED-ON COLCODE EQ
        SEGMENT CITY USED-IN CITY-SSA

ADD SKILLEMP-PCB;
PCB STRUCTURE
    BY GET ONLY
    SEGMENT SKILL
    SEGMENT NAME
    SEGMENT EXPR
;
ADD AUTOREG-PCB;
PCB STRUCTURE
    BY GET
    SEGMENT NAMEID SECONDARY-SEQUENCE
    SEGMENT CITY
;
```

Further Information

Segments

The least that can be recorded by DataManager, in the data definition for a segment, is the keyword SEGMENT followed by one of the keywords PHYSICAL, LOGICAL, or INDEX-POINTER. This specifies that the segment resides in a physical database, a logical database, or a secondary index database respectively.

When a SEGMENT member is being encoded, DataManager checks that it is not contained by the wrong type of database; for example, a logical segment cannot be contained by an HDAM database.

However, a SEGMENT data definition may be used for Source Language Generation; for example, to produce Database Description (DBD) control statements or COBOL, PL/I, or Assembler data descriptions for segment input/output areas. For these purposes, the data definition must be complete; that is, it must define the physical characteristics and attributes of the segment (for example, what fields it contains, and/or its sequence key field) and any logical or secondary indexing relationships in which it participates.

When a segment specified as participating in a logical or secondary indexing relationship is encoded, DataManager checks these items:

- That it is not related to the wrong type of segment; for example, a logical child segment must not refer to another logical child segment as its destination parent.
- That segments referring to the segment being encoded will not be made invalid because they are related to it in a manner that is invalid in the context of the relationship being specified.
- That the database that contains the segment being encoded is the type of database that permits a segment participating in the specified logical or secondary index relationship; for example, an HSAM database cannot contain segments that participate in such relationships.

All complete SEGMENT data definition statements, excepting those for logical child segments and index pointer segments, must include a CONTAINS list naming the fields that constitute the segment.

A logical child segment requires a CONTAINS list only if it has intersection data; DataManager automatically handles the concatenated key of its destination parent. A pair of logical child segments participating in a physically paired logical relationship must, if there is any intersection data, have CONTAINS lists whose respective constituent fields reflect the same total length for the intersection data (because when DL/I updates the intersection data for one of the logical child segments, it also automatically updates the intersection data for its physically paired logical child segment). The respective constituent fields may, however, specify different data dictionary members.

A virtual logical child segment does not physically exist in storage, but represents the real logical child segment with which it is paired as viewed from the logical parent segment, thus it never has a CONTAINS list specified for it; DataManager automatically obtains any intersection data from the real logical child segment.

An index pointer segment for a secondary index database requires a CONTAINS list only to specify any user data. Index pointer segments for the primary indexes of HIDAM databases are not held on the dictionary as members, but are generated automatically by the Source Language Generation Facility when producing Database Description (DBD) Control Statements for the primary index database. If the name for the primary index pointer segment and the name for its sequence key field have not been specified in the data definition of the HIDAM database nor in the Source Language Generation facility's PRODUCE command (see "Specification of the Data Definition Statement for a LOGICAL Type DL/I Database" on page 74 and "Generating DL/I DBD Control Statements" on page 131), then they are created by suffixing I to the respective names of the HIDAM root segment and the HIDAM root segment's sequence key field.

DL/I Data Fields

As stated in "Segments" on page 14, the CONTAINS lists in the dictionary SEGMENT data definitions specify the data fields that constitute the segments.

The CONTAINS list names ordinary data dictionary GROUP members and/or ITEM members, optionally with a version specified for ITEM members. The form of the GROUP and ITEM members is not specified, as the form is assumed in this priority:

- HELD-AS
- DEFAULTED-AS
- ENTERED-AS
- REPORTED-AS

The CONTAINS list may specify any number of variable length ITEM members, either directly or indirectly. When required, DataManager will calculate the minimum and maximum lengths for the segment; and when generating COBOL, PL/I, or Assembler data description statements for segment input/output areas, will generate size fields.

DL/I Databases

The least that can be recorded by DataManager in the data definition for a database is the keyword DL/I-DATABASE, followed by a keyword specifying the type of database; for example, LOGICAL, SECONDARY-INDEX, HSAM, or HDAM.

However, a DL/I-DATABASE data definition may be used for Source Language Generation to produce; for example, Database Description (DBD) or Program Specification Block (PSB) control statements. For these purposes, the data definition must be complete; that is, it must define the access method and storage organization of the database, and the hierarchical structure of the segments that constitute the database.

When an DL/I database is encoded, DataManager checks that the segments it contains are of a type that is valid for the type of database, and that the relationships in which its segments participate are valid for the type of database.

A primary index database for a HIDAM database is not held in the dictionary as a separate member; its access method and storage organization are specified as part of the data definition for the HIDAM database. When the Source Language Generation facility produces DBD control statements for an HIDAM database, it immediately follows them with DBD control statements for its primary index database. If the library member name for the DBD control statements and the database name for the primary index database have not been specified in the data definition of the HIDAM database nor in the PRODUCE command (see "Specification of the Data Definition Statement for a LOGICAL Type DL/I Database" on page 74 and "Generating DL/I DBD Control Statements" on page 131), then they are created by suffixing I to the respective names for the HIDAM database.

Special DataManager Member Types

For the DL/I Interface, DataManager automatically generates and maintains members of special internal types. The internal member types are for these items:

- Sequence key fields
- Data sets
- Index search fields (XDFLDs)
- System related fields
- Concatenated key names

Members of these types cannot be inserted into the data dictionary by users.

In normal circumstances, a segment's sequence key field is one of the dictionary GROUP or ITEM members that directly or indirectly constitute the segment. However, for a logical child segment, it may sometimes be required that the sequence key field consist of more than one (or part of more than one) of the key fields constituting the destination parent's concatenated key or any part of the destination parent's concatenated key plus part of the intersection data

In these circumstances, the SEGMENT data definition statement permits the specification of each of the contiguous fields that are to constitute the sequence key field, and a DL/I name that is to be applied to the sequence key field.

When the segment is encoded, DataManager then generates a member of a special internal type, giving it the specified sequence key name. If the segment specifying the sequence key is deleted, the special internal member for the sequence key field is also deleted, unless this internal member is referred to by other members, in which case it is made into a dummy member.

An internal member is always generated for the sequence key field specified in the data definition for an index pointer segment.

The DL/I-DATABASE data definition statements can include the names and definitions of the databases' constituent data sets. When a database is encoded, DataManager creates a member of a special internal type for each of the ddnames specified. When a database member is deleted so are any of the internal members that were created for its constituent data sets, except that if any of these internal members are referred to by other members they are made into dummy members.

The data definition for an index pointer segment specifies the name to be applied to the index search field (XDFLD). When such a member is encoded, DataManager creates a member of a special internal type, giving it the name specified for the index search field. If the index pointer segment is deleted, the special internal member created for the index search field is also deleted, unless this internal member is referred to by other members, in which case it is made into a dummy member.

The SEGMENT PHYSICAL data definition statement for an index source segment allows system related fields to be defined. These can be any part of the source segment's concatenated key or fields from which DL/I generates four byte unique values in the corresponding index pointer segment.

System related fields of the former type are handled by DataManager in the same way as sequence key fields; that is, each of those fields of the index source segment's concatenated key that are to form the system related field can be specified.

A name can be specified for each system related field of either type. The oblique stroke (/) that must be the first character of the name is added by DataManager when the Source Language Generation facility is used to produce Database Description (DBD) control statements for the database that contains the index source segment. DataManager creates an internal data dictionary member having the name specified for the system related field (that is, without the /). If the index source segment is deleted, then so are any special internal members that were created for system related fields specified by the segment, except that if any of these internal members are referred to by other members, they are made into dummy members.

A logical child segment always includes the concatenated key of its destination parent segment. Index pointer segments sometimes include the concatenated key of the index target segment (see "Specification of the Data Definition Statement for a SEGMENT that Resides in a Secondary Index Database" on page 46). The concatenated key is constructed automatically by DataManager when generating COBOL, PL/I, or Assembler data descriptions for segment input/output areas. The SEGMENT data definition statement allows a name to be specified for the concatenated key. When the segment is encoded, DataManager creates a member of a special internal type, giving it the name specified for the concatenated key. If the segment is deleted, the special internal member created for the concatenated key is also deleted, unless the member is referred to by other members, in which case it is made into a dummy member.

Normally, members of special internal types are transparent to the user. However, the DL/I Interface allows the member types to be made available to the user, for accessing in certain interrogation commands (for further details, including other documentation commands that can handle them, see Chapter 4, "Extensions to DataManager Commands for DL/I Databases," on page 99). Also, the user is able to produce COBOL, PL/I, and Assembler data description statements from the internal DataManager members created for the sequence key fields, index search fields (XDFLDs), system related fields, and concatenated key fields (see Chapter 5, "DL/I Source Language Generation from DataManager," on page 131 for further information).

Application View

As stated in "Application View" on page 13, an application's view of the segments that it accesses is defined to DataManager by PCB members and the PROCESSES clause, which can be specified in the data definition statements for data dictionary SYSTEM, PROGRAM, and MODULE members.

The PROCESSES clause specifies a CONTAINS clause listing each logical data structure, GSAM database, and output message destination (alternate) PCB that the application is to access.

When producing Program Specification Block (PSB) control statements for an application, the Source Language Generation facility produces a Program Communication Block (PCB) from each PCB member listed in the CONTAINS clause.

A PROCESSES clause can be defined for a data dictionary SYSTEM member. Usually, in an DL/I database system, the PROCESSES clause would be applicable to the data definition for a data dictionary PROGRAM member. However, to permit the definition of a modularized application, the DataManager DL/I Interface also allows the PROCESSES clause to be specified in the data definition for data dictionary MODULE members.

Whichever member relates to the control module in the application (and this may be of either SYSTEM, PROGRAM, or MODULE member type) will require a CONTAINS clause within its PROCESSES clause. This CONTAINS clause must list each PCB that the DL/I Interface will be passing to the control module when invoked. The CONTAINS clause is used by the Source Language Generation facility in producing its PSB Control Statements.

A PCB member defines a logical data structure, GSAM database, or output message destination which is to be accessed by the application. A logical data structure PCB also specifies all the segments to which any application SYSTEM/PROGRAM/MODULE containing the PCB is sensitive. In turn, each appropriate SEGMENT clause in a logical data structure PCB can define, via a SENSITIVE-FIELDS subordinate clause, the individual fields to which the application is sensitive. It is these definitions that the DataManager Source Language Generation facility uses to generate COBOL, PL/I, or Assembler data descriptions of the segment input/output areas for the sensitive fields.

The data dictionary SYSTEM, PROGRAM, or MODULE member may also include a PROCESSES clause containing a SEGMENT-SEARCH-ARGUMENTS subordinate clause. This clause defines the SEGMENT-SEARCH-ARGUMENTS specifying the segments (with their respective USED-IN clauses) which can then be used by the Source Language Generation facility in generating DBD Control Statements.

When generating the Database Description (DBD) control statements for a database, the user can specify in the PRODUCE command whether DataManager is to generate DL/I FIELD Control Statements for all the fields constituting each segment in the database, or only for each segment's search fields, sensitive fields, and fields required for secondary indexing. (XDFLDs, system related fields, sequence key fields, and fields specified in the GENERATES clause of the segment data definition statement are always generated.) If FIELD statements for a database are to be generated only for search fields, sensitive fields, and fields required for secondary indexing, then these actions must be taken to ensure that DataManager will recognize the fields:

- Each SYSTEM, PROGRAM, and MODULE member for each application that accesses segments in the database by means of qualified segment search arguments must name the search fields in a USED-IN subordinate clause within a PROCESSES clause of the SYSTEM, PROGRAM, or MODULE member data definition.
- Each structure type PCB member must name, using a SENSITIVE-FIELDS subordinate clause within each SEGMENT clause of the PCB definition, the fields to which it is sensitive in each segment contained in the database.
- Each index pointer segment that uses an index source segment contained by the database must specify, in its SEARCH, SUBSEQUENCE, and DUPLICATE-DATA lists, the GROUP and ITEM members contained by the index source segment that are required for secondary indexing.

3 DataManager Data Definition Statements for a DL/I Environment

Introduction

DataManager offers users in a DL/I environment 3 member types for defining segments, databases, and Program Communication Blocks, and an extra facility in the data definition statements for SYSTEM, PROGRAM, and MODULE members to enable the application view of the segments and databases that they use to be fully defined.

Users can define these 3 member types in an IMS (DL/I) environment:

Segment. See "DataManager Data Definition Statements for DL/I Segments" on page 21 for more information on the SEGMENT member type.

Database. See "DataManager Data Definition Statements for DL/I Databases" on page 56 for more information on the member types IMS-DATABASE and DL/l-DATABASE. (Any of the alternative forms DL/1-DATABASE, DLI-DATABASE, or DL1-DATABASE are accepted for the member type identifier DL/I-DATABASE.)

Program Communication Block. See "DataManager Data Definition Statements for DL/I Program Communication Blocks" on page 82 for more information on the member type PROGRAM-COMMUNICATION-BLOCK (or PCB), which is used to specify the application view of a database and the segments that the application uses.

You can fully define the application view of the segments and databases that they use, using the PROCESSES clause in the SYSTEM, PROGRAM, and MODULE member types (see "Specification of the PROCESSES Clause" on page 92).

DataManager Data Definition Statements for DL/I Segments

Outline of the SEGMENT Data Definition Statement

DL/I provides a comprehensive selection of keywords and operands in its SEGMENT statement in order to define all possible attributes and relationships of segments, which can reside in several fundamentally different types of database.

To simplify the description of DataManager's SEGMENT data definition statement, the format of the statement is specified separately for each different type if DL.I database. This is the overall outline format of the SEGMENT member type:

```
SEGMENT  {physical-database-segment          }
         {logical-database-segment           }
         {secondary-index-database-segment   }
[common clauses]
{;}
{.}
```

where:

physical-database-segment is the definition for the type of segment that resides in a physical database (see "Specification of the Data Definition Statement for a Segment that Resides in a Physical Database" on page 24).

logical-database-segment is the definition for the type of segment that resides in a logical database; that is, a logical segment or logical concatenated segment (see "Specification of the Data Definition Statement for a SEGMENT that Resides in a Logical Database" on page 42).

secondary-index-database-segment is the definition for the type of segment that resides in a secondary index database; that is, an index pointer segment (see "Specification of the Data Definition Statement for a SEGMENT that Resides in a Secondary Index Database" on page 46).

common clauses are any of these clauses (as defined in the *ASG-Manager Products Dictionary/Repository User's Guide*):

ACCESS-AUTHORITY	FREQUENCY
ADMINISTRATIVE-DATA	NOTE
ALIAS	OBSOLETE-DATE
CATALOG	QUERY
COMMENT	SECURITY-CLASSIFICATION
DESCRIPTION	SEE
EFFECTIVE-DATE	

It should be noted that there is no definition for the index pointer segment that resides in a primary index database. This type of segment definition is entirely handled by DataManager when required. It is required only by the Source Language Generation facility, to be used for the DBD control statements for the primary index database that will be generated automatically following the DBD control statements for a HIDAM database. This is one instance of an internal member type.

The names to be applied to the primary index pointer segment and to its sequence key field can be specified in the data definition of the HIDAM database (see "Specification of the Data Definition Statement for a HIDAM Type DL/I Database" on page 69) or in the Source Language Generation facility's PRODUCE command (see "Generating DL/I DBD Control Statements" on page 131). If they are omitted from both of these, then the name applied to the primary index pointer segment is the name of the HIDAM root segment suffixed with I, and the name of the sequence key field for the index pointer segment is the name of the sequence key field of the HIDAM root segment suffixed with I.

For each type of SEGMENT, the definition comprises:

- A segment type keyword.
- A RELATED-AS clause (for a physical-database-segment) or a RELATED-TO clause (for a secondary-index-database-segment), to define the logical and secondary indexing relationships in which the segment participates. There is no RELATED clause for a logical-database-segment definition.
- An ATTRIBUTES clause, to define the physical characteristics of the segment in relation to the database in which it resides.

For a segment that resides in a physical database, the RELATED-AS clause must precede the ATTRIBUTES clause, if both are present. For a segment that resides in a secondary index database, the ATTRIBUTES clause and the RELATED-TO clause can be in either order.

Both the RELATED-AS or RELATED-TO clause and the ATTRIBUTES clause must, if present, precede any common clauses that may be present.

Syntax

Figure 4. SEGMENT PHYSICAL Data Definition Statement Format

```
SEGMENT PHYSICAL

 [RELATED-AS
  DESTINATION-PARENT-SEGMENT [rules]
  UNIDIRECTIONAL-CHILD-SEGMENT [TO destination-parent-name] [rules] [CONCATENATED-KEY-NAME concatenated-key-name] [RENAMES renames-clause]
  REAL-PAIRED-CHILD-SEGMENT [TO destination-parent-name] [POINTERS [ { FORWARD-LOGICAL-TWIN } ] [ { SINGLE-LOGICAL-CHILD } ]] [rules]
                                                                    { BACKWARD-LOGICAL-TWIN }   { DOUBLE-LOGICAL-CHILD }
      [CONCATENATED-KEY-NAME concatenated-key-name] [RENAMES renames-clause]
  VIRTUAL-PAIRED-CHILD-SEGMENT [WITH real-paired-child-name] [TO destination-parent-name] [CONCATENATED-KEY-NAME concatenated-key-name]
      [RENAMES renames-clause]

 [TARGET-SEGMENT]
 [SOURCE-SEGMENT [CONCATENATED-KEY-FIELDS { group-item-name   }[, { group-item-name   }]... AS CKxxxxx
                                          { sequence-key-name }   { sequence-key-name }
                  [, { group-item-name   }[, { group-item-name   }]... AS CKxxxxx]...]
                     { sequence-key-name }   { sequence-key-name }
                [UNIQUE-KEY-FIELDS SXxxxxx]]

ATTRIBUTES

 [ALIGNED   ] [CONTAINS content [ELSE content [IF clause]] [ELSE content [IF clause]]...]
  UNALIGNED
  NOT-ALIGNED
                        [,content [ELSE content [IF clause]] [ELSE content [IF clause]]...]...]
                                  [IF clause]
 [SEQUENCE-KEY group-item-name [ { WITH group-item-name [,group-item-nane]... AS sequence-key-name } ] [ { UNIQUELY   } ]
                                 { AS sequence-key-name                                              }   { DUPLICATED }
               [,group-item-name [ { WITH group-item-name [,group-item-name]... AS sequence-key-name } ] [ { UNIQUELY   }]]...]
                                   { AS sequence-key-name                                              }   { DUPLICATED }

 INSERT-POSITION { FIRST } ]
                 { LAST  }
                 { HERE  }
 POINTERS [ SINGLE-TWIN ] [ FIRST-CHILD ]
          [ DOUBLE-TWIN ] [ LAST-CHILD  ]
          [ NOTWIN      ]
 EDIT-COMPRESSION-EXIT module-name [OPEN-CLOSE]]
 GENERATES   group-item-name [ { OF } ] { group-name            } ]... } [ group-item-name [ { OF } ] { group-name            } ]... } ]...]
                             [ { IN } ] { concatenated-key-name } ]   }   [                 { IN } ] { concatenated-key-name } ]   }
             sequence-key-name                                          , [ sequence-key-name                                        ]
             concatenated-key-name                                        [ concatenated-key-name                                    ]

[common clauses]

 ;
 .
```

where:

rules is a clause in this format:

```
RULES  [INSERT {PHYSICAL} ]  [DELETE {PHYSICAL} ]
               {LOGICAL }            {LOGICAL }
               {VIRTUAL }            {VIRTUAL }

       [REPLACE {PHYSICAL} ]
                {LOGICAL }
                {VIRTUAL }
```

destination-parent-name is the name of a segment that is a PHYSICAL DESTINATION-PARENT-SEGMENT.

concatenated-key-name is used when a logical child segment is being defined to specify the name to be given to the destination parent's concatenated key.

renames-clause is a clause in this format:

```
{group-item-name   } { AS         } local-name
{sequence-key-name } {   KNOWN-AS }
[, {group-item-name   } { AS         } local-name ] ...
   {sequence-key-name } {   KNOWN-AS }
```

where:

group-item-name is the name of a group or the name of an item.

sequence-key-name is a 1- to 8-character unique alphanumeric name.

local-name is a name, conforming to the rules for member names stated in the *ASG-Manager Products Dictionary/Repository User's Guide*, that can be used instead of the name or alias of the member named immediately prior to *local-name*, when DBD control statements, record layouts, or source language data descriptions are generated from this data definition by the DataManager Source Language Generation facility. The *local-name* is not separately recorded in the data dictionary (i.e., no dummy data entries record and no index record is created for *local-name* when the data definition in which it appears is encoded) so local-name cannot be interrogated and can be the same as another name, an alias, or a catalog classification in the data dictionary. The *local-name* is the name by which the member is known only within the segment defined by this data definition.

content declares an item, a group or an array, in this format:

Figure 5. Format of a Content Declaration in a SEGMENT PHYSICAL or SEGMENT INDEX-POINTER Data Definition

```
{ { item-name [version] } [ { ALIGNED     } ] [KNOWN-AS local-name]                                                    }
{ { group-name          }   { UNALIGNED   }                                                                            }
{                           { NOT-ALIGNED }                                                                            }
{ { (integer)                } { item-name [version] } [ { ALIGNED     } ] [KNOWN-AS local-name] [INDEXED-BY index-name] }
{ { (item-name-a [version]   } { group-name          } [ { UNALIGNED   } ]                                            }
{                                                        [ { NOT-ALIGNED } ]                                            }
```

where:

item-name is the name of an item.

version is an unsigned integer in the range 1 to 15, being a number specifying which version of the relevant item is relevant to this segment. The version is within the HELD-AS form, or within a defaulted form as stated in remark 31 on page 32. If version is omitted or if the stated *version does not exist*, the lowest numbered existing version is assumed to be relevant.

group-name is the name of a group.

local-name is a name that can be used instead of the name or alias of the contained member, as described above.

integer is an unsigned integer of from 1 to 18 digits, being the number of times *item-name* or *group-name* occurs in the array.

item-name-a is the name of an item. This form of array declaration declares that when the segment where defined is processed by an application program or module, the number of times *item-name* or *group-name* occurs in the array is contained in the item *item-name-a*.

index-name is a name, conforming to the rules for member names stated in the *ASG-Manager Products Dictionary/Repository User's Guide*, that is to be used as the index name when COBOL data descriptions are generated by the DataManager Source Language Generation facility. The index-name is not separately recorded in the data dictionary (that is, no dummy data entries record and no index record is created for *index-name* when the data definition in which it appears is encoded) so *index-name* cannot be interrogated and can be the same as another name, an alias, or a catalog classification in the data dictionary.

IF clause is a clause containing from 1 to 15 conditional terms. A conditional term compares the contents of an item with a comparand; it has the 3 elements item-name, operator, and comparand. If there are two or more conditional terms in the IF clause, they must be separated by an AND or OR keyword; they will be evaluated from left to right in a Boolean logical manner. The IF clause is declared in this format:

```
IF item-name-b [version-b]      { EQ  } { item-name-c [version-c] }
                                { =   } { literal                 }
                                { NE  }
                                { GT  }
                                { >   }
                                { GE  }
                                { LT  }
                                { <   }
                                { LE  }

[ { AND } item-name-b [version-b] { EQ  } { item-name-c [version-c] } ] ...
  { OR  }                         { =   } { literal                 }
                                  { NE  }
                                  { GT  }
                                  { >   }
                                  { GE  }
                                  { LT  }
                                  { <   }
                                  { LE  }
```

where:

item-name-b is the name of the item, the contents of which are to be compared with the comparand.

version-b is an unsigned integer in the range 1 to 15, being a number specifying the version (within the HELD-AS form, or within a defaulted form as stated in remark 34 on page 33) of *item-name-b* that is relevant to the comparison. If *version-b* is omitted, a default value of 1is assumed. The operators have these meanings:

EQ or = means equal to.
NE means not equal to.
ET or > means greater than.
GE means greater than or equal to.
E or < means less than.
LE means less than or equal to.

item-name-c is the name of the item, the contents of which are the comparand.

version-c is an unsigned integer in the range 1 to 15 being a number specifying the version (within the same type of form as that here being defined) of *item-name-c* the contents of which are the comparand. If *version-c* is omitted, a default value of 1 is assumed.

`literal` is a literal comparand, and must be compatible with the form-description in `item-name-b`'s data definition. (If `item-name-b`'s data definition contains a CONTENTS clause, literal should also be compatible with `item-name-b`'s contents-description.) The literal can be one of these:

- A character string of not more than 256 printable and/or non-printable characters, enclosed in quotes
- A numeric literal, that is:
  - — A signed or unsigned decimal number of not more than 18 digits, optionally with a decimal point, and not enclosed in quotes
  - — A signed or unsigned floating point number (as defined in the *ASG-Manager Products Dictionary/Repository User's Guide*) not enclosed in quotes

`module-name` is the name of a module.

`common clauses` are any of these clauses (as defined in the *ASG-Manager Products Dictionary/Repository User's Guide*):

ACCESS-AUTHORITY	FREQUENCY
ADMINISTRATIVE-DATA	NOTE
ALIAS	OBSOLETE-DATE
CATALOG	QUERY
COMMENT	SECURITY-CLASSIFICATION
DESCRIPTION	SEE
EFFECTIVE-DATE	

Remarks

1. The keyword PHYSICAL must always appear as the first keyword after the member type identifier, to indicate that a segment residing in a physical database is being defined. A SEGMENT PHYSICAL can be contained by any number of physical databases provided that it does not participate in a logical or secondary indexing relationship (that is, that it does not have a RELATED-AS clause in its definition).

2. The RELATED-AS clause must be present if the segment participates in a logical relationship or a secondary indexing relationship. If present, the RELATED-AS clause must immediately follow the PHYSICAL keyword. The RELATED-AS clause is not valid for a segment that resides in a HSAM, SIMPLE HSAM, or SIMPLE HISAM database. If a segment that participates in a logical relationship (other than as a DESTINATION-PARENT-SEGMENT) is to be completely defined, the TO clause and, if appropriate, the WITH clause must be present in the RELATED-AS clause.

3. If the segment participates in a logical relationship then one of these clauses must be specified in the RELATED-AS clause:
   - DESTINATION-PARENT-SEGMENT, which specifies that the segment being defined is either a logical parent segment or the physical parent segment of a real logical child segment in a virtually paired logical relationship.
   - UNDIRECTIONAL-CHILD-SEGMENT, which specifies that the segment being defined is a logical child segment in a unidirectional logical relationship.
   - REAL-PAIRED-CHILD-SEGMENT, which specifies that the segment being defined is a real logical child segment in a virtually paired logical relationship. This type of segment must reside in a HDAM or HIDAM database.
   - VIRTUAL-PAIRED-CHILD-SEGMENT, which specifies that the segment being defined is a virtual logical child segment in a virtually paired logical relationship.

4. `TO` *`destination-parent-name`* states the destination parent segment to which the logical child segment being defined is related. If the segment being defined is a virtual logical child segment, the destination parent segment is the physical parent of the real logical child segment with which it is paired; otherwise, the destination parent segment is the logical parent segment.

5. The POINTERS clause specifies the type of pointer that connects a real logical child segment and its logical parent segment.

6. In the POINTERS clause, either FORWARD-LOGICAL-TWIN or BACKWARD-LOGICAL-TWIN can be stated. If neither is stated, FORWARD-LOGICAL-TWIN is assumed. FORWARD-LOGICAL-TWIN specifies that a 4-byte logical twin forward pointer field is reserved in the prefix of the segment being defined. BACKWARD-LOGICAL-TWIN specifies that both a 4-byte logical twin forward pointer field and a 4-byte logical twin backward pointer field are reserved in the prefix of the segment being defined.

7. In the POINTERS clause, either SINGLE-LOGICAL-CHILD or DOUBLE-LOGICAL-CHILD can be stated. If neither is stated, SINGLE-LOGICAL-CHILD is assumed. SINGLE-LOGICAL-CHILD specifies that a 4-byte logical child first pointer field is reserved in the prefix of the logical parent segment of the segment being defined. DOUBLE-LOGICAL-CHILD specifies that both a 4-byte logical child first pointer field and a 4-byte logical child last pointer field are reserved in the prefix of the logical parent segment of the segment being defined.

8. The RULES clause specifies the rules for inserting, deleting and replacing a segment.

9. The CONCATENATED-KEY-NAME clause can be used when a logical child segment is defined, to specify the name that is to be given to the concatenated key of the destination parent segment. When the logical child segment definition is encoded, a member of a special internal type is created for the concatenated key, and gives it the name specified in the CONCATENATED-KEY-NAME clause. This internal member has no entries in the uses table, as the elements that constitute the concatenated key are not obtained until the Source Language Generation facility is used (see remark 11 on page 30). However, the internal member can still be referred to by other members; for example, it may be used as a segment search field or as a sensitive field.

10. Interrogations can be performed on the concatenated key internal member type (see "Condition Keywords for Which and What Commands" on page 102). However, meaningful results will only be obtained in response to interrogations concerning members which use the internal member type, as the member type has no entries in the uses table.

11. The destination parent's concatenated key is constructed automatically when the Source Language Generation facility is being used to generate DBD control statements, record layouts, or COBOL, PL/I, or Assembler data descriptions. If a CONCATENATED-KEY-NAME clause is present in the segment's data definition, the concatenated key is given the name specified in the clause.

12. The RENAMES clause can be used to specify a local name for any field that directly constitutes the destination parent's concatenated key, that is, any field that has been directly specified as a sequence key in any of the segments along the hierarchical path to the destination parent segment, and including destination parent segment. The rules governing a local name are as defined in the syntax.

13. A segment cannot be a logical child segment and a destination parent segment.

14. A segment cannot be a logical child segment if it is the root segment of a database.

15. A logical child segment cannot have a logical child, destination parent, or target segment as a dependant at any lower level of the hierarchy.

16. A destination parent segment cannot have another destination parent segment as a dependant at any lower level of the hierarchy.

17. A virtual logical child segment cannot have physical child segment.

18. The keyword TARGET-SEGMENT specifies that the segment being defined is an index target segment. A segment cannot be an index target segment and also a logical child segment or a dependent segment of a logical child segment at any lower level.

19. The keyword SOURCE-SEGMENT specifies that the segment being defined is an index source segment.

20. The CONCATENATED-KEY-FIELDS clause defines any number of system related fields of the type that enables any part of the concatenated key of the index source segment to be used in the subsequence or duplicate data fields of the corresponding index pointer segment. The definition of each such system related field comprises:

    - The names of any number of groups, items, and/or sequence keys that are to comprise the system related field. The members named must be contiguous within the index source segment's concatenated key. They can be:

      — Members contained directly or indirectly in the segment's sequence key; and/or

      — Members contained directly or indirectly in the sequence key of any segment along the hierarchical path to and including the index source segment

    - A clause AS CK*xxxxx*, which specifies the name to be applied to the system related field. The name must be unique, must be 3 to 7 characters in length, and must commence with CK.

21. When a source segment definition that contains a CONCATENATED-KEY-FIELDS clause is encoded, a member of a special internal type for each system related field defined by the clause is created. This member is given uses table entry for each item, group and sequence key member specified in the CONCATENATED-KEY-FIELDS clause. Members of this special internal type can be referred to by other members, for example, by an index pointer segment, and they can also be interrogated (see "Condition Keywords for Which and What Commands" on page 102). The Source Language Generation facility can operate on members of this type.

22. The UNIQUE-KEY-FIELDS clause defines any number of system related fields of the type that prompts DL/I to generate a unique 4-byte value of the source segment's VSAM relative block address and to place it in the subsequence field of the corresponding index pointer segment. SX*xxxxx* specifies the name to be applied to a system related field of this type. The name must be unique, must be 3 to 7 characters in length, and must commence with SX.

23. When a source segment definition that contains a UNIQUE-KEY-FIEDS clause is encoded, a member of a special internal type for the system related field defined by the clause is created. This member does not refer to any other members and therefore has no entries in the uses table. However, members of this special internal type can be referred to by other members, for example, by an index pointer segment, and they can also be interrogated (although meaningful results will only be obtained in response to interrogations about members that use the internal member, as it has no constituent members). The Source Language Generation facility can operate on members of this type.

24. A segment cannot be an index source segment and a logical child segment.

25. The ATTRIBUTES clause must be present if the segment is to be completely defined.

26. The first element within the ATTRIBUTES clause can be one of the keywords ALIGNED, UNALIGNED, or NOT-ALIGNED. If none is declared in the data definition statement, a default of UNALIGNED is taken.

27. ALIGNED is the equivalent of COBOL SYNCHRONIZED or PL/I ALIGNED. It means that (subject to remark 31 on page 32) all binary items and all floating point items declared as being contained in the segment are aligned to half word, full word or double word boundaries, thus:

    - Binary items having a length of 4 decimal digits or less occupy a complete half word
    - Binary items having a length of from 5 to 9 decimal digits occupy a full word
    - Binary items having a length of from 10 to 18 decimal digits occupy two full words, but are not necessarily aligned to a double word boundary
    - Floating-point items having 6 digits or less in the mantissa occupy a full word
    - Floating-point items having from 7 to 16 digits in the mantissa occupy a double word

    ALIGNED also causes any bit string items to be output with alignment to byte boundaries when the Source Language Generation facility is used. The way in which this is achieved is dependent upon the language being generated, and is described further in the *ASG-Manager Products Source Language Generation* publication.

28. UNALIGNED means that (subject to remark 31 on page 32) binary items and floating point items declared (as individual items or as elements of an array) as being contained in the segment are not necessarily aligned to word or half word boundaries, and that bit string items are not aligned to byte boundaries. (The amount of space occupied is the same as for ALIGNED items, but the positioning relative to boundaries can differ.)

29. NOT-ALIGNED means the same as UNALIGNED. For the sake of simplicity, they are regarded in the following remarks as being the same keyword; so that any reference to the UNALIGNED keyword should be interpreted as applying equally to the NOT-ALIGNED keyword.

30. The ALIGNED or UNALIGNED keyword does not apply to items contained within groups declared as being contained in the segment. The data definitions of the groups determine the alignment or non-alignment of such indirectly referenced items.

31. The ALIGNED or UNALIGNED keyword can be overridden for individual content declarations (that is, for particular items or groups declared as being contained in the segment) by including the keyword UNALIGNED or ALIGNED respectively in the particular content declaration, preceding any associated ELSE and/or IF clauses (see remark 35 on page 33 through remark 39 on page 34). It is not meaningful to include either of these keywords in a content declaration that declares a group or an array of groups (see remark 30 on page 32).

32. The CONTAINS clause specifies the GROUP and/or ITEM members and/or arrays that constitute the successive parts of the segment being defined. It must be present unless the segment being defined is a logical child segment.

    If the segment being defined is a virtual logical child segment then the CONTAINS clause must not be present, as the segment's constituent members are obtained from the real logical child segment with which it is paired.

    For a logical child segment that is not a virtual logical child, the CONTAINS clause is required only to define the intersection data. If there is no intersection data then the CONTAINS clause must be omitted.

    The destination parent's concatenated key is automatically constructed when it is required for the Source Language Generation facility.

33. The entries in the CONTAINS clause must include, directly or indirectly, references to these fields, if they are applicable to the segment being defined:

    - The sequence key fields
    - The segment search fields
    - The fields that are to be included in the index search field, subsequence fields, and duplicate data fields of the corresponding index pointer segment, if the segment being defined is an index source segment
    - Sensitive fields

34. Any direct or indirect reference from the CONTAINS clause to an item is assumed to be the HELD-AS form of that item. If the item has no HELD-AS form, default assumptions are made as to the relevant form of the item, in the order DEFAULTED-AS, ENTERED-AS, and REPORTED-AS. The form first encountered in this order is taken as the defaulted form, and version is applied within that form as stated in the syntax.

35. Entries in the CONTAINS clause may be conditional (IF clauses, see remark 37 on page 33) and/or may have alternative content declarations (ELSE clauses, see remark 36 on page 33), which also may be conditional: so that the definition of each part of the segment comprises a content declaration and any associated ELSE clauses and/or IF clauses. If the segment comprises two or more parts, the definition of each part except the last must be followed by a comma, which can optionally be followed by spaces.

36. Any part of the segment can be specified as having any number of alternative contents. The alternative content declarations are separated by the keyword ELSE. The alternative contents need not occupy the same amount of physical storage.

    The expression *ELSE clause* thus refers to:

    `ELSE` *`content`*

    where *`content`* is as defined above.

37. Any content declaration can be specified as conditional (i.e., applied only if a stated condition or combination of conditions is satisfied). For a content declaration to be conditional, content must be immediately followed by an IF clause.

38. It follows that any part of the segment can have alternative conditional contents, declared in this form:

    ```
    content IF clause ELSE content IF clause
    [ELSE content IF clause]...
    ```

    and that any combination of conditional and non-conditional alternative contents can be declared for any part of the segment.

39. In a content declaration, the ALIGNED, UNALIGNED, or NOT-ALIGNED element, the KNOWN-AS clause and the INDEXED-BY clause can, if applicable, be declared in any order; but they must not precede any of the other elements of the content declaration (though they must precede any associated ELSE clauses and/or IF clauses).

40. The SEQUENCE-KEY clause specifies the field that is the sequence key of the segment being defined. For a virtual logical child segment, it specifies the field that is the sequence key of the paired real logical child segment when accessed from its logical parent segment.

41. Only one entry may be specified in the SEQUENCE-KEY clause, unless the segment being defined is a virtual logical child segment, in which case any number of entries can be specified. The term entry in this context means group/item name, optionally followed by a WITH and/or an AS clause and optionally followed by one of the keywords UNIQUELY or DUPLICATED. WITH and AS clauses are only valid for a virtual logical child schedule.

42. For a segment that is not a logical child segment, the field named in the SEQUENCE-KEY clause must be directly or indirectly contained in the segment. If the reference to the field from the CONTAINS clause is indirect, the field must not appear as an array in the data definition of its containing group.

43. For a logical child segment, the field named in the SEQUENCE-KEY clause must be:

    - Directly or indirectly contained in the segment being defined, or, if the segment is a virtual logical child segment, directly or indirectly contained in its paired real child segment.
    - Directly/indirectly contained in the destination parent's concatenated key; that is, directly/indirectly contained in the sequence key field of any segment along the hierarchical path to and including the destination parent segment.

    If the reference to the field is indirect, the field must not appear as an array in the data definition of its containing group.

44. The WITH clause can be used only if a virtual logical child segment is being defined. It enables contiguous parts of the destination parent's concatenated key and/or contiguous parts of the segment's intersection data to be included as part of the segment's sequence key field.

45. Each GROUP/ITEM listed in the WITH clause, must be the name of one of these:

    - A sequence key field or a member contained directly or indirectly in a sequence key field of any segment along the hierarchical path to and including the destination parent segment.
    - A field contained directly or indirectly in the intersection data of the logical child segment, or if a virtual logical child segment is being defined then in the intersection data of its paired real logical child segment.

    The fields named in the list must be contiguous.

46. The AS clause specifies the name that is to be applied to the sequence key field constituted by the members named in the associated WITH clause and the GROUP or ITEM name immediately preceding that WITH clause.

47. When no WITH clause is specified, the AS clause specifies an alternative name for the GROUP or ITEM name that immediately precedes it. This allows an alternative name to be given to one of the fields in the destination parent's concatenated key, if that field is the sequence key of the logical child segment.

48. When a logical child segment definition containing an AS clause (with or without any WITH clause) is encoded, a member of a special internal type is created for the sequence key. This member is given an entry in the uses table for each member that is named between the SEQUENCE-KEY and As keywords. Sequence key internal members can be referred to by other members, for example, as segment search arguments (SSAs) or sensitive fields, and they can also be interrogated (see "Condition Keywords for Which and What Commands" on page 102). The Source Language Generation facility can operate on members of this type.

49. The relevant version of any item, to which reference is made directly or indirectly from the SEQUENCE-KEY clause, is assumed to be the same as the version of that item that is relevant to the CONTAINS clause of the segment in which it is contained.

50. UNIQUELY indicates that only unique values are allowed in the sequence key field being defined. DUPLICATED indicates that duplicate values are allowed. If neither are specified, UNIQUELY is assumed. All of the sequence keys for a virtual logical child segment must be uniformly defined as either UNIQUELY or DUPLICATED.

51. You must specify a sequence key field for the root segment of a HDAM database. A unique sequence key field must be specified for the root segment of a HISAM, SIMPLE HISAM, or HIDAM database.

52. If the segment is a destination parent segment, then a sequence key field should be specified for it and for each of the segments on which it depends. It is strongly recommended that each of the sequence key fields be unique.

53. A sequence key field must be specified for an index target segment.

54. The INSERT-POSITION clause is omitted if the segment resides in a HSAM or SIMPLE HSAM database. Otherwise, it must be present if a unique sequence key field has not been specified.

55. The INSERT-POSITION clause specifies where an occurrence of the segment is inserted. Thus, FIRST states that:

    - If SEQUENCE-KEY is not specified, a new occurrence of the segment is inserted in front of all existing occurrences.
    - If SEQUENCE-KEY is DUPLICATED, a new occurrence of the segment is inserted in front of all existing occurrences that contain the same sequence key.

    LAST (the default) states that:

    - If SEQUENCE-KEY is not specified, a new occurrence of the segment is inserted behind all existing occurrences.
    - If SEQUENCE-KEY is DUPLICATED, a new occurrence of the segment is inserted behind all existing occurrences that contain the same sequence key.

    HERE states that:

    - If position has been established on an occurrence of the segment by a previous DL/I call, a new occurrence of the segment is inserted in front of the occurrence that satisfied that call.
    - If the current position is not within occurrences of the segment, a new occurrence of the segment is inserted as for FIRST.

56. The POINTERS clause in the ATTRIBUTES clause is applicable only to segments that reside in a HDAM or HIDAM database and are not virtual logical child segments except for the COUNTER keyword, which is also valid for segments residing in a HISAM database.

57. SINGLE-TWIN specifies that a 4-byte physical twin forward pointer field is reserved in the prefix of the segment.

58. DOUBLE-TWIN specifies that a 4-byte physical twin forward pointer field and a 4-byte physical twin backward pointer field are reserved in the prefix of the segment.

59. NOTWIN specifies that the segment may occur only once per physical parent.

60. FIRST-CHILD specifies that a 4-byte physical child first pointer field is to be placed in the prefix of the segment's physical parent segment.

61. LAST-CHILD specifies that a 4-byte physical child first pointer field and a 4-byte physical child last pointer field are to be placed in the prefix of the segment's physical parent segment.

62. The EDIT-COMPRESSION-EXIT clause specifies the selection of an edit and/or compression user exit option. The clause is invalid if the segment resides in a HSAM, SIMPLE HSAM, or SIMPLE HISAM database or is a virtual logical child segment or is a source segment.

63. OPEN-CLOSE specifies that initialization and termination processing control is required by the segment edit routine; that is, that the edit/compression routine will gain control after database open and after database close.

64. The GENERATES clause enables the user to specify the fields for which DBD FIELD control statements are always to be generated when DBD control statements are produced. It is in addition to those fields required by IMS (DL/I) for which the Source Language Generation facility always provides DBD FIELD control statements (see "Generating DL/I DBD Control Statements" on page 131). It is not necessary to include sequence key field names in the GENERATES clause. This is because DBD FIELD control statements are always generated for these fields; however, sequence key names, as well as group names and/or item names, are accepted in the GENERATES clause in case the user wishes to include them in the list of specified fields.

65. The OF/IN subordinate clause of the GENERATES clause can be used when the segment contains multiple occurrences of a field, to allow the user to specify which occurrence of the field is to have a DBD FIELD control statement generated for it. If the OF/IN clause is used, all occurrences of the field other than the one specified in the clause are ignored.

66. The facility (described in "Generating DL/I DBD Control Statements" on page 131), which automatically generates DBD FIELD control statements for the fields described below, cannot be used when fields are duplicated across segments, because it is assumed there is no such duplication. Instead, the GENERATES clause must be used if it is required to generate DBD FIELD control statements for these fields:

    - Fields that are used as segment search fields via the PROCESSES clause of SYSTEM, PROGRAM, or MODULE members
    - Fields that are used as sensitive fields in PCB members
    - Fields that are used for secondary indexing via the SEARCH, SUBSEQUENCE, or DUPLICATE-DATA lists of the appropriate index pointer segments, when an index source segment is being processed

    Therefore if data has been duplicated across segments and you wish to generate DBD FIELD control statements for the types of fields listed above, then:

    - The GENERATES clause must be specified in the definition of each segment of the database to be processed to specify the fields for which DBD FIELD control statements are to be generated.
    - The GENERATES-FIELDS keyword must be used in the PRODUCE DL/I DBDGEN command to indicate that DBD FIELD control statements are to be generated only for the fields specified in the GENERATES clause.

67. The length of the segment is not specified in the segment definition, because it is automatically calculated when required.

68. If the segment resides in a HSAM, SIMPLE HSAM, or SIMPLE HISAM database or a database that does not use the VSAM operating system access method, it must not be variable length.

69. If the segment does not reside in a HSAM, SIMPLE HSAM, or SIMPLE HISAM database, or a database that does not use the VSAM operating system access method, any field contained in the segment may be variable length except the following:

    - A sequence key field or any of its constituent members
    - Any fields preceding the sequence key field

70. For a variable length segment, the minimum length must include the length of the sequence key field and must not change the offset of the sequence key. When the EDIT-COMPRESSION-EXIT clause is specified, the minimum length cannot be less than 4 bytes.

71. If a variable length segment is encountered when the Source Language Generation facility is being used to generate DBD control statements, record layouts, or COBOL, PL/I, or Assembler data description statements, the 2-byte-size field required by IMS(DL/I) for the segment is generated automatically (see "Variable Length Segments" on page 144).

72. A variable length segment is defined by specifying that the segment contains, directly or indirectly, a variable length ITEM member.

73. It should be noted that a variable length segment must be defined to the VS COBOL compiler by specifying a variable length array.

    A segment that directly or indirectly contains a variable length array is not recognized as a variable length segment.

    If COBOL data description statements are to be generated for a variable length segment, the segment must contain, directly or indirectly, a variable length ITEM member, and this member must be redefined by a variable length array. For example, if COBOL data descriptions are generated from the following data definition:

    ```
    CONTAINS
    ITEMA ELSE (ITEMB) ITEMC
    ;
    ```

    The VS COBOL compiler will output a warning message, but the compilation will continue. However, it should be noted that the following definition:

    ```
    CONTAINS
    (ITEMB) ITEMC ELSE ITEMA
    ;
    ```

    will cause the VS COBOL compiler to output an error message and compilation will fail.

74. Common clauses can be present in any type of data definition statement; they are therefore documented separately, in the *ASG-Manager Products Dictionary/Repository User's Guide*. Not more than one of each of these clauses can be declared. If a common clause has a subordinate clause or keyword, the subordinate clause identifier or subordinate keyword must not be truncated to an extent where it becomes ambiguous with any other clause identifier or other keyword available in the data definition syntax for this member type.

75. The common clauses can be declared in any order. If present, they must follow the RELATED-AS and ATTRIBUTES clauses, if these are present. If the latter clauses are both present, the RELATED-AS clause must precede the ATTRIBUTES clause.

76. Within the RELATED-AS clause, the subordinate clauses can be in any order; and, if a subordinate clause has subordinate clauses and optional keywords, such clauses and keywords can be in any order within the subordinate clause.

77. Within the ATTRIBUTES clause, the subordinate clauses can be in any order (but see remark 26 on page 31). The optional keywords in the POINTERS clause can be in either order.

78. A record containing the segment's data definition statement can be inserted into the repository's source dataset by a suitable command (see the *ASG-Manager Products Dictionary/Repository User's Guide*), and an encoded record can subsequently be generated and inserted into the data entries dataset. If, when the encoded record is generated, any item, group, module, or segment which name appears in the segment's data definition statement has no data entries record, Manager Products creates a dummy data entries record for that member. The dummy record is created as:

    - A dummy module if the name appears in an EDIT-COMPRESSION-EXIT or EXIT-LIST clause
    - A dummy segment if the name is a destination-parent-name, a physically-paired-child-name or a real-paired-child-name
    - A dummy group if the name appears in the OF/IN subordinate clause of the GENERATES clause
    - A dummy item in all the other cases

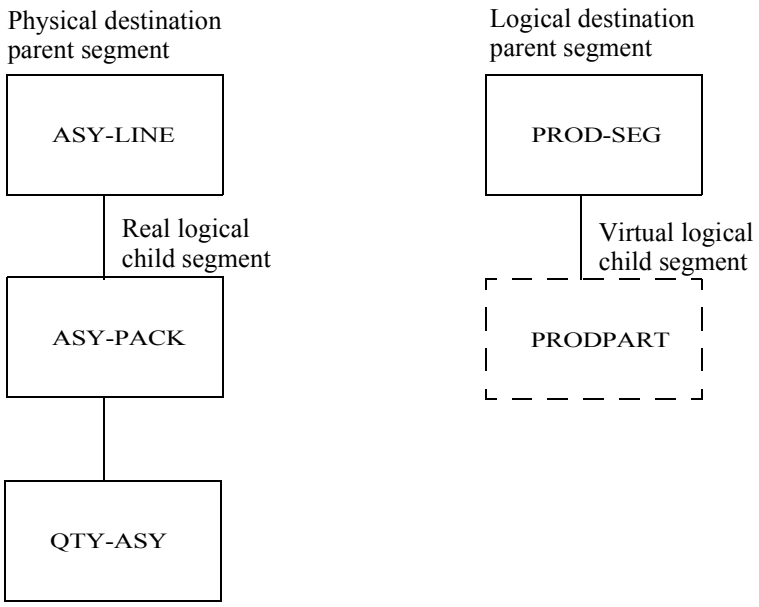
79. If an encoded segment record is deleted, any internal repository member that it created which is not referred to by other members is deleted, together with any references that the internal member made to other members. Any internal member that is referred to by other members is made into a dummy internal member rather than being deleted altogether.

Examples

For a comprehensive cross section of examples showing the ATTRIBUTES clause in the data definition statement for a SEGMENT PHYSICAL, see the examples illustrated by Figure 2 on page 6 and Figure 3 on page 9. Also in those examples are segments participating in a unidirectional logical relationship, and an index target segment.

Figure 6 illustrates two physical data structures that contain segments participating in a virtually paired logical relationship.

Figure 6. Example of Physical Data Structures With Segments Participating in a Virtually Paired Logical Relationship



In Figure 6:

- ASY-LINE is the physical segment for an assembly line which assembles packs of assembly parts to make a product.
- ASY-PACK is the physical segment for a pack of assembly parts being assembled on that assembly line.
- QTY-ASY is the physical segment for the number of those packs of assembly parts assembled on that assembly line.
- PROD-SEG is the physical segment for a product.
- PROD-PART is the physical segment for the parts that are used to make that product.

Below are examples of the data definition statements that could be used to define the segments illustrated in Figure 6 on page 40. The examples also show the use of complex SEQUENCE-KEY clauses.

```
ADD ASY-LINE;
SEGMENT PHYSICAL
RELATED-AS DESTINATION-PARENT-SEGMENT
ATTRIBUTES
                    CONTAINS ASY-CODE
SEQUENCE-KEY ASY-CODE UNIQUELY
;
ADD ASY-PACK;
SEGMENT PHYSICAL
RELATED-AS REAL-PAIRED-CHILD-SEGMENT TO PROD-SEG
    POINTERS SYMBOLIC DIRECT-ADDRESS
ATTRIBUTES
 CONTAINS PACK.NO, PART. COLOUR, QTY-REQD
 SEQUENCE-KEY PROD-NO WITH PACK-NO AS PACKKEY
INSERT-POSITION LAST
;
ADD QTY-ASY;
SEGMENT PHYSICAL
ATTRIBUTES
CONTAINS QTY
INSERT-POSITION LAST
;
ADD PROD-SEG;
SEGMENT PHYSICAL
RELATED-AS DESTINATION-PARENT-SEGMENT
ATTRIBUTES
CONTAINS PROD-NO, DESCRIPT
SEQUENCE-KEY PROD-NO UNIQUELY
;
ADD PRODPART;
SEGMENT PHYSICAL
RELATED-AS VIRTUAL-PAIRED-CHILD-SEGMENT WITH ASY-PACK
     TO ASY-LINE
ATTRIBUTES
SEQUENCE-KEY PART WITH COLOUR, QTY-REQD AS PART-KEY,
     ASY-CODE
INSERT-POSITION LAST
;
```

For examples of logical data structures that can be defined from the virtually paired logical relationship illustrated above, see Figure 7 on page 44 and the accompanying narrative.

Specification of the Data Definition Statement for a SEGMENT that Resides in a Logical Database

Syntax

```
SEGMENT LOGICAL

   [ATTRIBUTES]
       CONTAINS physical-segment-name [IN physical-database-name
             [,destination-parent-name
   ]

[common clauses]

{;}
{.}
```

where:

physical-segment-name is the name of a PHYSICAL SEGMENT.

destination-parent-name is a PHYSICAL DESTINATION-PARENT-SEGMENT.

physical-database-name is the name of a HDAM or HIDAM database.

common-clauses are any of these clauses (as defined in the *ASG-Manager Products Dictionary/Repository User's Guide*):

ACCESS-AUTHORITY	FREQUENCY
ADMINISTRATIVE-DATA	NOTE
ALIAS	OBSOLETE-DATE
CATALOG	QUERY
COMMENT	SECURITY-CLASSIFICATION
DESCRIPTION	SEE
EFFECTIVE	

Remarks

1. The keyword LOGICAL must immediately follow the SEGMENT member type identifier, to indicate that a segment residing in a logical database is being defined.

2. The keyword ATTRIBUTES can be omitted for a logical segment; it is included in the statement specification in order to maintain the general format of the segment data definition statements.

3. The CONTAINS clause specifies the physical segments that the logical segment represents. The clause must be present if the segment is to be completely defined.

4. The `physical-segment-name` specified can be the name of a segment of any type that resides in a HDAM or HIDAM database, unless a logical concatenated segment is being defined, in which case it must be the name of a logical child segment.

5. If the physical segment resides in more than one physical database, the IN subordinate clause can be used to specify the name of the physical database relevant to this logical segment. The name of the physical database is required when DL/I DBD control statements are being produced for any logical database that contains this logical segment. If the IN clause is not specified, then when DL/I DBD control statements are produced, Manager Products finds an appropriate physical database in one of the ways described in "Specification of the Data Definition Statement for a LOGICAL Type DL/I Database" on page 74.

6. The `destination-parent-name` is specified only if a logical concatenated segment is defined, in which case it must be the name of the destination parent segment to which the logical child segment specified by `physical-segment-name` relates.

   If the `physical-segment-name` specifies a logical child segment but the `destination-parent-name` is omitted, then the Source Language Generation facility assumes that a logical concatenated segment is being defined. The destination parent to which it is related is also assumed.

7. The sequence key for a concatenated segment is the sequence key of the logical child segment.

8. Common clauses can be present in any type of data definition statement; they are therefore defined separately, in the *ASG-Manager Products Dictionary/Repository User's Guide*. Not more than one of each of these clauses can be declared. If a common clause has a subordinate clause or keyword, the subordinate clause identifier or subordinate keyword must not be truncated to an extent where it becomes ambiguous with any other clause identifier or other keyword available in the data definition syntax for this member type.

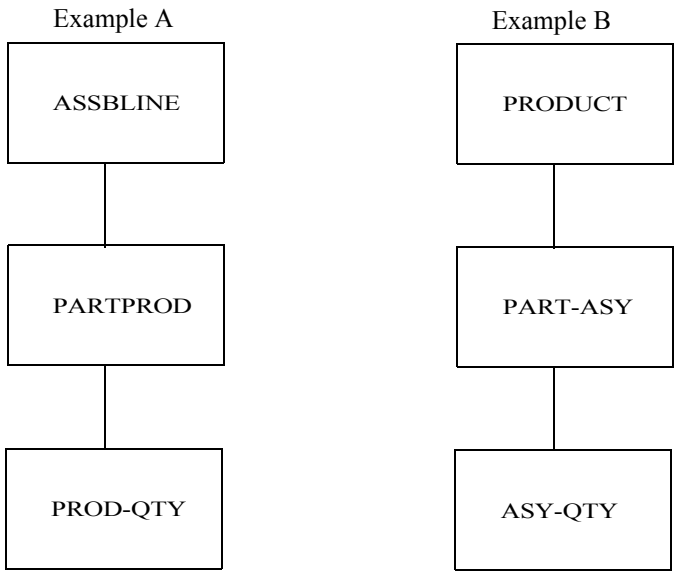
9. The common clauses can be in any order. If present, they must follow the ATTRIBUTES clause, if that clause is present.

10. A record containing the segment's data definition statement can be inserted into the data dictionary's source dataset by a suitable command (see *ASG-Manager Products Dictionary/Repository User's Guide*), and an encoded record can subsequently be generated and inserted into the data entries dataset. If, when the encoded record is generated, any segment or database whose name appears in this segment's data definition statement has no data entries record, a dummy segment or database data entries record is created for that member.

Examples

Figure 7 illustrates logical data structures that can be defined from the physical data structures illustrated by Figure 6 on page 40.

Figure 7. Examples of Logical Data Structures



In Figure 7, example A:

- ASSBLINE is a logical segment representing an assembly line
- PARTPROD is a logical concatenated segment representing assembly parts assembled on that assembly line and the product that they make
- PROD-QTY is a logical segment representing the number of those products being assembled on that assembly line

These are examples of the data definition statements that could define the segments illustrated in Figure 7, example A:

```
ADD ASSBLINE;
SEGMENT LOGICAL
CONTAINS ASY-LINE
;
ADD PARTPROD;
SEGMENT LOGICAL
CONTAINS ASY~PACKf PROD-SEG
;
ADD PROD-QTY;
SEGMENT LOGICAL
CONTAINS QTY-ASY
;
```

In Figure 7 on page 44, example B:

PRODUCT is a logical segment representing a product.

PART-ASY is a logical concatenated segment representing the parts that are used to make this product and the assembly line where they are assembled.

ASY-QTY is a logical segment representing the number of those assembly parts assembled on that assembly line.

These are examples of the data definition statements that could define the segments illustrated in Figure 7 on page 44, example B:

```
ADO PRODUCT;
SEGMENT LOGICAL
CONTAINS PROD-SEG
;
ADD PART-ASY;
SEGMENT LOGICAL
CONTAINS PRODPART, ASY-LINE
;
ADD ASY-QTY
SEGMENT LOGICAL
CONTAINS QTY-ASY
;
```

Syntax

Figure 8. SEGMENT INDEX-POINTER Data Definition Statement Format

```
SEGMENT INDEX-POINTE
                                                    { }
ATTRIBUTES
  [{ALIGNED   }] [CONTAINS content[{ELSE content [IF clause] } [ELSE content [IF clause]]...]
   {UNALIGNED }                    {IF clause                }
   {NOT-ALIGNED}]
                          [,content[{ELSE content [IF clause] } [ELSE content [IF clause]]...]]...]
                                    {IF clause                }
  SEQUENCE-KEY key-name    UNIQUELY
                           DUPLICATED
  [GENERATES{ item-name                [{IN} {group-name                }]...{[{item-name                [{IN} {group-name                }]...}]...
            { group-name               [{OF} {CKxxxxx                   }    {group-name               [{OF} {CKxxxxx                   }    }
            { SXxxxxx                        {index-search-field-name}        {SXxxxxx                       {index-search-field-name}    }
            { CKxxxxx                        {sequence-key-name       }        {CKxxxxx                       {sequence-key-name       }    }
            { index-search-field-name}                                          {index-search-field-name}                                }
            {
            { sequence-key-name                                                 {    sequence-key-name                                   }
RELATED-TO
  target-segment-name ON index-search-field-name
  [SOURCE source-segment-name]
  SEARCH-KEY-FIELDS {group-name} [, {group-name} ]...
                    {item-name }    {item-name }
  [SUBSEQUENCE-FIELDS { CKxxxxx }   [, { CKxxxxx } ]...]
                      { SXxxxxx }      { SXxxxxx }
                      { group-name }   { group-name }
                      { item-name }    { item-name }
  DUPLICATE-DATA-FIELDS { CKxxxxx }   [, { CKxxxxx } ]...]
                        { group-name }   { group-name }
                        { item-name }    { item-name }
  SUPPRESSING-ON { CHARACTER c     }]
                 { HEXADECIMAL hh  }
                 { BITS bbbbbbbb   }
                 { BLANKS          }
                 { ZEROS           }
                 { ZEROES          }
  [MAINTENANCE-EXIT module-name]
[common clauses]

 ;
 .
```

where:

`content` declares an item, a group or an array, in the format shown in Figure 5 on page 26.

`item-name` is the name of an item.

`version` is an unsigned integer in the range 1 to 15, being a number specifying which version of the relevant item is relevant to this segment. The version is within the HELD-AS form, or within a defaulted form as stated in remark 31 on page 54. If version is omitted or if the stated *version does not exist*, the lowest numbered existing version is assumed to be relevant.

`group-name` is the name of a group.

`local-name` is a name that can be used instead of the name or alias of the contained member, as described above.

`integer` is an unsigned integer of from 1 to 18 digits, being the number of times `item-name` or `group-name` occurs in the array.

`item-name-a` is the name of an item. This form of array declaration declares that when the segment where defined is processed by an application program or module, the number of times `item-name` or `group-name` occurs in the array is contained in the item `item-name-a`.

`index-name` is a name, conforming to the rules for member names stated in the *ASG-Manager Products Dictionary/Repository User's Guide*, that is to be used as the index name when COBOL data descriptions are generated by the DataManager Source Language Generation facility. The index-name is not separately recorded in the data dictionary (that is, no dummy data entries record and no index record is created for `index-name` when the data definition in which it appears is encoded) so `index-name` cannot be interrogated and can be the same as another name, an alias, or a catalog classification in the data dictionary.

IF clause is a clause containing from 1 to 15 conditional terms. A conditional term compares the contents of an item with a comparand; it has the 3 elements item-name, operator, and comparand. If there are two or more conditional terms in the IF clause, they must be separated by an AND or OR keyword; they will be evaluated from left to right in a Boolean logical manner. The IF clause is declared in this format:

```
IF item-name-b [version-b]      { EQ  } { item-name-c [version-c] }
                                { =   } { literal                 }
                                { NE  }
                                { GT  }
                                { >   }
                                { GE  }
                                { LT  }
                                { <   }
                                { LE  }

[ { AND } item-name-b [version-b] { EQ  } { item-name-c [version-c] } ] ...
  { OR  }                         { =   } { literal                 }
                                  { NE  }
                                  { GT  }
                                  { >   }
                                  { GE  }
                                  { LT  }
                                  { <   }
                                  { LE  }
```

where:

item-name-b is the name of the item, the contents of which are to be compared with the comparand.

version-b is an unsigned integer in the range 1 to 15, being a number specifying the version (within the HELD-AS form, or within a defaulted form as stated in remark 34 on page 33) of *item-name-b* that is relevant to the comparison. If *version-b* is omitted, a default value of 1is assumed. The operators have these meanings:

EQ or = means equal to.
NE means not equal to.
ET or > means greater than.
GE means greater than or equal to.
E or < means less than.
LE means less than or equal to.

item-name-c is the name of the item, the contents of which are the comparand.

version-c is an unsigned integer in the range 1 to 15 being a number specifying the version (within the same type of form as that here being defined) of *item-name-c*, the contents of which are the comparand. If *version-c* is omitted, a default value of 1 is assumed.

`literal` is a literal comparand, and must be compatible with the `form-description` in `item-name-b`'s data definition. (If `item-name-b`'s data definition contains a CONTENTS clause, literal should also be compatible with `item-name-b`'s contents-description.) The literal can be:

- A character string of not more than 256 printable and/or non-printable characters, enclosed in quotes
- A numeric literal, that is:
  - A signed or unsigned decimal number of not more than 18 digits, optionally with a decimal point, and not enclosed in quotes
  - A signed or unsigned floating point number (as defined in the *ASG-Manager Products Dictionary/Repository User's Guide*) not enclosed in quotes

`group-name` is the name of a group.

`item-name` is the name of an item.

`version` is an unsigned integer in the range 1 to 15, being a number specifying which version of the relevant item is relevant to this segment. The version is within the HELD-AS form, or within a defaulted form as stated in remark 10. If version is omitted or if the stated version does not exist, the lowest numbered existing version is assumed to be relevant.

`key-name` is a 1- to 8-character unique alphanumeric name.

`sequence-key-name` is a 1- to 8-character unique alphanumeric name.

`target-segment-name` is the name of a segment that is a PHYSICAL TARGET-SEGMENT.

`index-search-field-name` is a 1- to 8-character unique alphanumeric name.

`source-segment-name` is the name of a segment that is a PHYSICAL SOURCE-SEGMENT.

`c` is any printable character.

`hh` is a hexadecimal representation of any printable or non-printable character.

`bbbbbbbb` is a 1-byte bit string representation of any printable or non-printable character.

`module-name` is the name of a module.

`common clauses` are any of these clauses (as defined in the *ASG-Manager Products Dictionary/Repository User's Guide*):

ACCESS-AUTHORITY	FREQUENCY
ADMINISTRATIVE-DATA	NOTE
ALIAS	OBSOLETE-DATE
CATALOG	QUERY
COMMENT	SECURITY-CLASSIFICATION
DESCRIPTION	SEE
EFFECTIVE-DATE	

Remarks

1. The keyword INDEX-POINTER must immediately follow the SEGMENT member type identifier, to indicate that an index pointer segment residing in a secondary index database is being defined.

2. The ATTRIBUTES clause must be present if the segment is to be completely defined.

3. The first element within the ATTRIBUTES clause can be one of the keywords ALIGNED, UNALIGNED, or NOT-ALIGNED. If none is declared in the data definition statement, a default of UNALIGNED is taken.

4. ALIGNED is the equivalent of COBOL SYNCHRONIZED, or PL/I ALIGNED. It means that (subject to remark 8 on page 51) all binary items and all floating point items declared as being contained in the segment are aligned to half word, full word or double word boundaries, thus:

   - Binary items having a length of 4 decimal digits or less occupy a complete half word
   - Binary items having a length of from 5 to 9 decimal digits occupy a full word
   - Binary items having a length of from 10 to 18 decimal digits occupy 2 full words, but are not necessarily aligned to a double word boundary
   - Floating-point items having 6 digits or less in the mantissa occupy a full word
   - Floating-point items having from 7 to 16 digits in the mantissa occupy a double word

   ALIGNED also causes any bit string items to be output with alignment to byte boundaries when the Source Language Generation facility is used. The way in which this is achieved depends on the language being generated, and is described for COBOL, PL/I, and Assembler in the publication *ASG-Manager Products Source Language Generation*.

5. UNALIGNED means that (subject to remark 8 on page 51) binary items and floating point items declared as being contained in the segment are not necessarily aligned to word or half word boundaries and that bit string items are not aligned to byte boundaries. (The amount of space occupied is the same as for ALIGNED items, but the positioning relative to boundaries can differ.)

6. NOT-ALIGNED means the same as UNALIGNED. For the sake of simplicity, they are regarded in the following remarks as being the same keyword; so that any reference to the UNALIGNED keyword should be interpreted as applying equally to the NOT-ALIGNED keyword.

7. The ALIGNED or UNALIGNED keyword does not apply to items contained within groups declared as being contained in the segment. The data definitions of the groups determine the alignment or non-alignment of such indirectly-referenced items.

8. The ALIGNED or UNALIGNED keyword can be overridden for individual content declarations (that is, for particular items or groups declared as being contained in the segment) by including the keyword UNALIGNED or ALIGNED, respectively, as the last element in the particular content declaration, preceding any associated ELSE and/or IF clauses (see remark 11 on page 51 through remark 15 on page 52). It is not meaningful to include either of these keywords in a content declaration that declares a group (see remark 7 on page 51).

9. The CONTAINS clause specifies the GROUP and/or ITEM members that constitute the successive parts of the index pointer segment's user data. If there is no user data, the CONTAINS clause must be omitted. The main part of the index pointer segment from the SEARCH-KEY-FIELDS, SUBSEQUENCE-FIELDS, and DUPLICATE-DATA-FIELDS subordinate clauses specified in the RELATED-TO clause is automatically constructed.

10. Any direct or indirect reference from the CONTAINS clause to an item is assumed to be to the HELD-AS form of that item. If the item has no HELD-AS form, default assumptions are made as to the relevant form of the item, in the order DEFAULTED-AS, ENTERED-AS, REPORTED-AS. The form first encountered in this order is taken as the defaulted form, and version is applied within that form as stated in the syntax.

11. Entries in the CONTAINS clause may be conditional (IF clauses, see remark 13 on page 52) and/or may have alternative content declarations (ELSE clauses, see remark 12 on page 51), which also may be conditional, so that the definition of each part of the segment comprises a content declaration and any associated ELSE clause and/or IF clauses. If the segment comprises two or more parts, the definition of each part except the last must be followed by a comma which can optionally be followed by spaces.

12. Any part of the segment can be specified as having any number of alternative contents. The alternative content declarations are separated by the keyword ELSE. The alternative contents need not occupy the same amount of physical storage.

    The expression *ELSE clause* thus refers to:

    `ELSE` *`content`*

    where *`content`* is as defined above.

13. Any content declaration can be specified as conditional (i.e., applied only if a stated condition or combination of conditions is satisfied). For a content declaration to be conditional, content must immediately be followed by an IF clause.

14. It follows that any part of the segment can have alternative conditional contents, declared in the form:

    ```
    content IF clause ELSE content IF clause
    [ELSE content IF clause]...
    ```

    and that any combination of conditional and non-conditional alternative contents can be declared for any part of the segment.

15. In a content declaration, the ALIGNED, UNALIGNED, or NOT-ALIGNED element, the KNOWN-AS clause and the INDEXED-BY clause can, if applicable, be declared in any order; however, they must not precede any of the other elements of the content declaration (though they must precede any associated ELSE clauses and/or IF clauses).

16. The SEQUENCE-KEY clause specifies the name that is to be applied to the sequence key of the index pointer segment. Manager Products constructs the sequence-key, for which a member of a special internal type is generated. A member of this type is given the following uses table entries:

    - An entry for the *index-search-field-name* (XDFLD) when specified for the segment (see remark 24 on page 54)
    - An entry for each entry specified in the SUBSEQUENCE-FIELDS clause in the segment definition

    The sequence key internal member type can be referred to by other members, for example, as a segment search argument, or as a sensitive field. Sequence key internal members can be interrogated, and the Source Language Generation facility can operate on such members.

17. UNIQUELY specifies that the sequence key of the index pointer segment is to contain unique values only. DUPLICATED specifies that duplicate values are allowed in the sequence key. If neither of these keywords is specified, then UNIQUELY is assumed.

18. The GENERATES clause enables the user to specify fields for which DBD FIELD control statements are always to be generated when DBD control statements are produced, in addition to those fields required by IMS (DL/I) for which the Source Language Generation facility always provides DBD FIELD control statements. (See "Generating DL/I DBD Control Statements" on page 131) It is not necessary to include the sequence key field name in the GENERATES clause, because a DBD FIELD Control Statement is always generated for this field; but the sequence key is accepted in the GENERATES clause in case the user wishes to include it in the list of specified fields.

19. The OF/IN subordinate clause of the GENERATES clause can be used when the segment contains multiple occurrences of a field, to allow the user to specify which occurrence of the field is to have a DBD FIELD control statement generated for it. If the OF/IN clause is used, all occurrences of the field other than the one specified in the clause are ignored.

20. When specified for an index pointer segment, the GENERATES clause has the additional function of forcing DBD FIELD control statements to be generated for fields that are in the main part of the index pointer segment [i.e., if the search, subsequence and duplicate-data fields, and fields constituting the concatenated key of the index target segment are present (see remark 35 on page 55 and remark 36 on page 55)]. Normally, DBD FIELD control statements are only generated for the sequence key field and for fields in the user data (see "Generating DL/I DBD Control Statements" on page 131).

    If it is required to generate DBD FIELD control statements for the fields that constitute the search, subsequence or duplicate-data fields then each field must be specified in the GENERATES clause of the index pointer segment definition.

21. When there is duplication of fields across segments, the GENERATES clause must be used if DBD FIELD control statements are to be generated for these fields:

    - Fields that are used as segment search fields via the PROCESSES clause of SYSTEM, PROGRAM, or MODULE members
    - Fields that are used as sensitive fields in PCB members

    These fields must be part of the user-data.

    The facility (described in "Generating DL/I DBD Control Statements" on page 131) which automatically generates the DBD FIELD control statements for the fields described above cannot be used when fields are duplicated across segments, as Manager Products assumes that there is no such duplication.

    If data has been duplicated across segments and you want to generate DBD FIELD control statements for the types of fields listed above, then:

    - The GENERATES clause must be specified in the definition of the segment to specify the fields for which DBD FIELD control statements are to be generated.
    - The GENERATES-FIELDS keyword must be used in the PRODUCE DLI DBDGEN command to indicate that DBD FIELD control statements are to be generated only for the fields specified in the GENERATES clause.

22. The RELATED-TO clause must be present if the segment is to be completely defined. It specifies:

    - The index target segment to which the segment is related
    - The index source segment to which the segment is related
    - The fields that are used to construct the CONSTANT, search, subsequence, and duplicate-data portion of the segment

23. The RELATED-TO keyword must be immediately followed by the *target-segment-name*, which identifies the PHYSICAL-TARGET-SEGMENT to which the index pointer segment points.

24. `ON` *`index-search-field-name`* specifies the name to be applied to the search field (XDFLD) of the index pointer segment that can be used as a segment search field for the index target segment. Manager Products constructs the index search field, for which a member of a special internal type is generated. This member is given a uses table entry for each member specified in the SEARCH-KEY-FIELDS clause.

    Index search field (XDFLD) internal members can be referred to by other members, for example, as a segment search argument. Members of this type can also be interrogated and the Source Language Generation facility can operate on them.

25. The SOURCE clause identifies the index source segment from which the index pointer segment is generated. The clause can be omitted if the index target segment is also the index source segment; otherwise the index source segment must be a dependent segment of the index target segment, at any lower level.

26. The SEARCH-KEY-FIELDS clause lists the names of one to 5 GROUP or ITEM members that are contained directly or indirectly by the corresponding index source segment, and that constitute the index search field (XDFLD) in the index pointer segment. The sequence of the entries in the list is the sequence in which the field values are concatenated in the index pointer segment's search field. None of these fields or their constituent members may be variable length.

27. The SUBSEQUENCE-FIELDS clause lists the names of one to 5 groups, items, and/or system related fields that are defined in the corresponding index source segment, and that constitute the subsequence field in the index pointer segment. The sequence of the entries in the list is the sequence in which the field values are concatenated in the index pointer segment's subsequence field.

28. The combined length of the fields declared by CONSTANT, SEARCH-KEY-FIELDS, and SUBSEQUENCE-FIELDS must not exceed 236 bytes.

29. The DUPLICATE-DATA-FIELDS clause lists the names of one to 5 groups, items and/or system related fields (of the type whose names begin with CK) that are defined in the corresponding index source segment, and that constitute the duplicate data field in the index pointer segment. The sequence of the entries in the list is the sequence in which the field values are concatenated in the index pointer segment's duplicate data field.

30. The SUPPRESSING-ON clause specifies that the creation of the index pointer segment is suppressed if each of the fields of the index source segment that are used to construct the search field of the index pointer segment contains the specified value in every byte.

31. The MAINTENANCE-EXIT clause specifies that a user-supplied index maintenance exit routine is used to suppress the creation of selected index pointer segments.

32. The length of the index pointer segment is not included as part of the segment definition as the Source Language Generation facility calculates it when required, allowing for:
    - The length of the key
    - Any duplicate data fields
    - Any user data

33. Common clauses can be present in any type of data definition statement; they are therefore defined separately in the *ASG-Manager Products Dictionary/Repository User's Guide*. Not more than one of each of these clauses can be declared. If a common clause has a subordinate clause or keyword, the subordinate clause identifier or subordinate keyword must not be truncated to an extent where it becomes ambiguous with any other clause identifier or other keyword available in the data definition syntax for this member type.

34. The common clauses can be declared in any order. If present, they must follow the ATTRIBUTES and RELATED-TO clauses, if they are present. The latter clauses can be in either order. Within the ATTRIBUTES clause the subordinate clauses can be in any order. Within the RELATED-TO clause the subordinate clauses can follow index-search-field-name in any order.

35. A record containing the segment's data definition statement can be inserted into the repository's source dataset by a suitable command (see the *ASG-Manager Products Dictionary/Repository User's Guide*), and an encoded record can subsequently be generated and inserted into the data entries dataset. If, when the encoded record is generated, any item, group, module, or segment whose name appears in the segment's data definition statement has no data entries record, a dummy data entries record is created for that member. The dummy record is created as:
    - A dummy module if the name appears in a MAINTENANCE-EXIT clause
    - A dummy segment if the name is a `target-segment-name` or a `source-segment-name`
    - A dummy group if the name appears in the OF/IN subordinate clause of the GENERATES clause
    - A dummy item in all other cases

    Similarly, when the encoded record is generated, if a member of an internal member type has not already been generated for any name appearing in a SUBSEQUENCE-FIELDS clause or a DUPLICATEDATA-FIELDS clause, then a dummy data entries record is created for that member. (The record is a dummy item because the internal member type will be defined in the physical source segment's definition.)

36. If an encoded segment record is deleted, any internal member that it created that is not referred to by other members is deleted, together with any references that the internal member made to other members. Any internal member that is referred to by other members is made into a dummy internal member rather than being deleted altogether.

Example

For an example of a SEGMENT INDEX-POINTER see the example illustrated by Figure 3 on page 9.

DataManager Data Definition Statements for DL/I Databases

Outline of the DL/I-DATABASE Data Definition Statement

DL/I provides a number of different organizations and access methods. To simplify the description of DataManager's DL/I-DATABASE data definition statement, the format of the statement is specified separately for each different type of DL/I database organization/access method. The member type identifier IMS-DATABASE, which is synonymous with DL/I-DATABASE, as included in the syntax for compatibility with the OS version of the interface.

This is the overall syntax of the IMS-DATABASE member type:

```
{ IMS-DATABASE   }     hsam-access
{ DL/I-DATABASE  }     hisam-access
{ DL/1-DATABASE  }     hdam-access
{ DLI-DATABASE   }     hidam-access
{ DL1-DATABASE   }     logical-access
                       secondary-index-access

[common clauses]

{ ; }
{ . }
```

where:

hsam-access, *hisam-access*, *hdam-access*, *hidam-access*, *logical-access*, and *secondary-index-access* are the definitions for particular types of database organization/access method, as specified in "Specification of the Data Definition Statement for a SECONDARY-INDEX Type DL/I Database" on page 78.

common clauses are any of these clauses (as defined in the *ASG-Manager Products Dictionary/Repository User's Guide*):

ACCESS-AUTHORITY	FREQUENCY
ADMINISTRATIVE-DATA	NOTE
ALIAS	OBSOLETE-DATE
CATALOG	QUERY
COMMENT	SECURITY-CLASSIFICATION
DESCRIPTION	SEE
EFFECTIVE-DATE	

For each type of database organization/ access method, the definition comprises:

- An organization type/access method keyword or keywords.
- An ACCESS clause, to specify the access method of the database. There is no ACCESS clause for a logical-access definition.
- A DATASETS clause, to specify the attributes of the dataset groups into which the database is divided. There is no DATASETS clause for a logical-access definition.
- A CONTAINS clause, to list the segments that reside in the database.

The ACCESS, DATASETS, and CONTAINS clauses must, if present, be in that order and must precede any common clause that may be present.

Specification of the Data Definition Statement for a HSAM Type DL/I Database

Syntax

```
{ IMS-DATABASE  }  HISAM [SIMPLE]
{ DL/I-DATABASE }
{ DL/1-DATABASE }
{ DLI-DATABASE  }
{ DL1-DATABASE  }

  [ACCESS]

  [DATASETS INPUT ddname [RECORD length]                    ]
  [         OUTPUT ddname [RECORD length]                   ]
  [          DEVICE device [ASSIGN SYSmmm SYSnnn]           ]
  [CONTAINS physical-segment-name                           ]
  [          [,physical-segment-name PARENT physical-segment-name]

[common clauses]

{; }
{. }
```

where:

ddname is 1 to 7 alphanumeric characters, being the logical name used in the job control to identify the physical file.

length is an unsigned integer other than 0, being the maximum length (in bytes) of a logical record.

device is one of these keywords or numbers:

DRUM 2311 3310 3370 3420

CELL 2314 3330 3375

TAPE 2319 3340 3380

2301 2321 3344 3400

2305 2400 3350 3420

FBA

mmm, *nnn* are the 3 digit integers in the range 001 to 240.

physical-segment-name is the name of any PHYSICAL segment.

common clauses are any of these clauses (as defined in the *ASG-Manager Products Dictionary/Repository User's Guide*):

ACCESS-AUTHORITY	FREQUENCY
ADMINISTRATIVE-DATA	NOTE
ALIAS	OBSOLETE-DATE
CATALOG	QUERY
COMMENT	SECURITY-CLASSIFICATION
DESCRIPTION	SEE
EFFECTIVE-DATE	

Remarks

1. The HSAM keyword must immediately follow the member type identifier to indicate that a HSAM database is being defined.

2. The keyword SIMPLE specifies that the database being defined is a SIMPLE HSAM database. If present, it must immediately follow the keyword HSAM.

3. The ACCESS clause can be omitted, but if it is present it must immediately follow the HSAM [SIMPLE] keyword(s).

4. The DATASETS clause defines a dataset group within this database. It must be present if the definition of the database is to be complete. Only one DATASETS clause is permitted.

5. INPUT *ddname* specifies the logical file name of the input dataset. It must be unique within the data dictionary.

6. OUTPUT *ddname* specifies the logical file name of the output dataset. It must be unique within the data dictionary.

7. If a RECORD subordinate clause is present in either of the INPUT or OUTPUT clauses, a RECORD subordinate clause must be present in both. The length specified in the RECORD clause for the output file must be equal to or greater than the length specified in the RECORD clause for the input file.

8. The DEVICE clause specifies the physical storage device for these files. If the physical storage device is a magnetic tape unit, the ASSIGN clause is mandatory. SYS*nnn* represents the symbolic tape unit to be associated with the INPUT ddname. SYS*mmm* represents the symbolic tape unit to be associates with the OUTPUT ddname.

9. The CONTAINS clause must be present if the definition of the database is to be complete. It must follow the DATASETS clause if both clauses are present.

10. The CONTAINS clause for a SIMPLE HSAM database states the name of the one segment that resides in the database.

11. The CONTAINS clause for a HSAM database lists the names of from 1 to 255 segments that reside in the database. The segments must be listed in hierarchical sequence, that is from top to bottom and left to right.

12. The PARENT clauses identify the physical parents of the segment whose names are listed in the CONTAINS clause. A PARENT clause must not be present for the first name listed (that of the root segment) but must follow each of the other names listed in the CONTAINS clause.

13. Common clauses can be present in any type of data definition statement; they are therefore defined separately, in the *ASG-Manager Products Dictionary/Repository User's Guide*. Not more than one of each of these clauses can be declared. If a common clause has a subordinate clause or keyword, the subordinate clause identifier or subordinate keyword must not be truncated to an extent where it becomes ambiguous with any other clause identifier or other keyword available in the data definition syntax for this member type.

14. The common clauses can be declared in any order. If present, they must follow the ACCESS, DATASETS, and CONTAINS clauses, if these are present.

15. A record containing the database's data definition statement can be inserted into the data dictionary's source dataset by a suitable command (see the *ASG-Manager Products Dictionary/Repository User's Guide*), and an encoded record can subsequently be generated and inserted into the data entries dataset.

    When the encoded record is generated, a data entries record of a special internal type, a DL/I-DATASET member, is created for each ddname that appears in the database's data definition. The DL/I-DATASET internal member is given a uses table entry for each segment that constitutes the dataset defined by the member. The DL/I-DATASET internal member can be referred to by other members; for example, it could be used in the INPUTS clause of a PROGRAM data definition. DL/I-DATASET members can also be interrogated (see "Condition Keywords for Which and What Commands" on page 102).

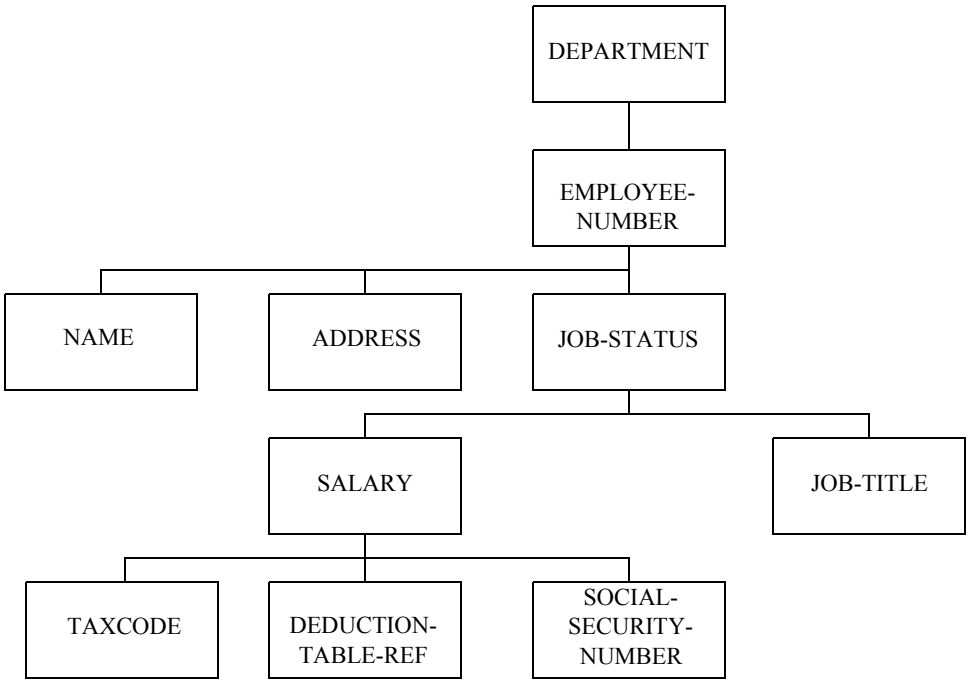
    If' when the encoded record is generated, any segment whose name appears in the database's data definition statement has no data entries record, a dummy data entries record is created for that member as a dummy segment record.

16. When an encoded database member is deleted, any DL/I-DATASET member created for it which is not referred to by other members is also deleted, together with any references that the DL/l-DATASET member made to segments. Any DL/I-DATASET member that is referred to by other members is a dummy member rather than being deleted.

Example

Figure 9 represents a possible hierarchical structure of segments constituting a personnel database called EMPLOYEE-DETAILS. A definition of a HSAM database implementing a structure could be as follows. In this example, meaningful segment names have been retained. The abbreviated 8-character names required by DL/I can be defined as DL/I aliases in the ALIAS clauses of the members that constitute the database.

Figure 9. Segments Constituting a Personnel Database, EMPLOYEE-DETAILS




```
ADD EMPLOYEE-DETAILS
IMS-DATABASE HSAM
ACCESS PASSWORD
DATASETS INPUT EMPLIN RECORD 1024
         OUTPUT EMPLOUT RECORD 1024
         DEVICE 3330 MODEL 1
CONTAINS DEPARTMENT.
         EMPLOYEE-NUMBER PARENT DEPARTMENT,
         NAME PARENT EMPLOYEE-NUMBER.
         ADDRESS PARENT EMPLOYEE-NUMBER,
         JOB-STATUS PARENT EMPLOYEE-NUMBER.
         SALARY PARENT JOB-STATUS.
         TAXCODE PARENT SALARY.
         DEDUCTION-TABLE-REF PARENT SALARY.
         SOCIAL-SECURITY-NUMBER PARENT SALARY.
         JOB-TITLE PARENT JOB-STATUS
;
```

Specification of the Data Definition Statement for a HISAM Type DL/I Database

Syntax

```
{ IMS-DATABASE  }  HISAM [SIMPLE]
{ DL/I-DATABASE }
{ DL/1-DATABASE }
{ DLI-DATABASE  }
{ DL1-DATABASE  }

   [ACCESS [VSAM]]

  [DATASETS PRIME ddname [BLOCK count [RECORD length]]          ]
  [         [OVERFLOW ddname [BLOCK count [RECORD length]]]     ]
  [          DEVICE device                                      ]

  [CONTAINS physical-segment-name                                ]
  [          [,physical-segment-name PARENT physical-segment-name]...]

[common clauses]

{; }
{. }
```

where:

ddname is 1 to 7 alphanumeric characters, being the logical name used in the job control to identify the physical file.

count is an unsigned, non-zero integer, being the number of logical records per physical block.

length is an unsigned non-zero integer, being the maximum length (in bytes) of a logical record. If VSAM is the operating system access method, length must be an even value.

device is one of the keywords or numbers from the list:

DRUM	2311	3310	3350
CELL	2314	3330	3375
2301	2319	3340	3380
2305	2321	3344	FBA

physical-segment-name is the name of a PHYSICAL segment.

common clauses are any of these clauses (as defined in the *ASG-Manager Products Dictionary/Repository User's Guide*):

ACCESS-AUTHORITY	FREQUENCY
ADMINISTRATIVE-DATA	NOTE
ALIAS	OBSOLETE-DATE
CATALOG	QUERY
COMMENT	SECURITY-CLASSIFICATION

DESCRIPTION SEE

EFFECTIVE-DATE

Remarks

1. The HISAM keyword must immediately follow the member type identifier to indicate that a HISAM database is being defined.

2. The keyword SIMPLE specifies that the database being defined is a SIMPLE HISAM database. If present, it must immediately follow the keyword HISAM.

3. The ACCESS clause can be omitted, but if it is present it must immediately follow the HISAM SIMPLE keyword(s).

4. The DATASETS clause defines a dataset group within this database. It must be present if the definition of the database is to be complete.

5. Within the DATASETS clause, the PRIME clause must always be specified. It defines the prime dataset of the dataset group.

6. Within the DATASETS clause, the OVERFLOWS clause must always be specified for a HISAM database, but is invalid for a SIMPLE HISAM database. It defines the overflow file of the database.

7. The ddname in the PRIME clause and the ddname in the OVERFLOW clause, which specify the logical file names of the respective datasets, must each be unique in the data dictionary.

8. If a BLOCK subordinate clause is present in either the OVERFLOW clause or the PRIME clause, a BLOCK subordinate clause must be present in both; in which case, if an associated RECORD subordinate clause is present in either, a RECORD clause must be present in both.

9. The RECORD length specified for the OVERFLOW clause must be equal to or greater than the RECORD length specified for the PRIME clause, if both are specified.

10. The RECORD length specified for a SIMPLE HISAM database must be equal to the length of the contained segment.

11. The control interval size is specified by the product of the BLOCK count and the RECORD length, must not exceed 4096. On encoding, a warning message is output if the product is not a multiple of 512.

12. The DEVICE clause specifies the physical storage device for the dataset files.

13. The CONTAINS clauses list the segments that reside in the database. For the definition of the database to be complete, the CONTAINS clauses must be present and must immediately follow a DATASETS clause.

14. A SIMPLE HISAM database can only contain one segment. For a HISAM database, 1 to 255 different segments can be specified. The segments must be specified in hierarchal sequence; that is, from top to bottom and left to right.

15. The PARENT clauses identify the physical parents of the segments whose names are listed in the CONTAINS clauses. A PARENT clause must not be present for the root segment (the first `physical-segment-name`) but must follow each of the other names listed in the CONTAINS clause.

16. Common clauses can be present in any type of data definition statement; they are therefore defined separately in the *ASG-Manager Products Dictionary/Repository User's Guide*. Not more than one of each of these clauses can be declared. If a common clause has a subordinate clause or keyword, the subordinate clause identifier or subordinate keyword must not be truncated to an extent where it becomes ambiguous with any other clause identifier or other keyword available in the data definition syntax for this member type.

17. The common clauses can be declared in any order. If present, they must follow the ACCESS, DATASETS, and CONTAINS clauses, if these are present.

18. A record containing the database's data definition statement can be inserted into the data dictionary's source dataset by a suitable command (see the *ASG-Manager Products Dictionary/Repository User's Guide*), and an encoded record can subsequently be generated and inserted into the data entries dataset.

    When the encoded record is generated, a data entries record of a special internal type, a DL/I-DATASET member, is created for each ddname that appears in the database's data definition. The DL/I-DATASET internal member is given a uses table entry for each segment that constitutes the dataset defined by the member. The DL/I-DATASET internal member can be referred to by other members, for example, it could be used in the INPUTS clause of a PROGRAM data definition. DL/I-DATASET members can also be interrogated (see "DL/I Member-type Keywords" on page 99).

    If, when the encoded record is generated, any segment whose name appears in the database's data definition statement has no data entries record, a dummy data entries record is created for that member as a dummy segment record.

    When an encoded database member is deleted, any DL/I-DATASET member created for it which is not referred to by other members is also deleted, together with any references that the DL/I-DATASET member made to segments. Any DL/I-DATASET member that is referred to by other members is made into a dummy member rather than being deleted.

Examples

These examples of data definition statements for HISAM databases relate to the hierarchical structure of segments listed in "Specification of the Data Definition Statement for a HSAM Type DL/I Database" on page 57. In these examples meaningful segment names have been retained. The abbreviated 8-character names required by DL/I can be defined as DL/I aliases in the ALIAS clauses of the members that constitute the database. The database could be defined thus:

```
ADD EMPLOYEE-DETAILS:
IMS-DATABASE HISAM
ACCESS VSAM PASSWORD
DATASETS PRIME EMPLP BLOCK 8 RECORD 256
         DEVICE 3330
CONTAINS DEPARTMENT.
         EMPLOYEE-NUMBER PARENT DEPARTMENT,
         NAME PARENT EMPLOYEE-NUMBER.
         ADDRESS PARENT EMPLOYEE-NUMBER.
         JOB-STATUS PARENT EMPLOYEE-NUMBER,
         SALARY PARENT JOB-STATUS.
         TAXCODE PARENT SALARY.
         DEDUCTION-TABLE-REF PARENT SALARY.
         SOCIAL-SECURITY-NUMBER PARENT SALARY.
         JOB-TITLE PARENT JOB-STATUS
;
```

Specification of the Data Definition Statement for a HDAM Type DL/I Database

Syntax

```
{ IMS-DATABASE   }  HDAM
{ DL/I-DATABASE  }
{ DL/1-DATABASE  }
{ DLI-DATABASE   }
{ DL1-DATABASE   }

  [ACCESS [VSAM]                                                        ]
  [                                                                     ]
  [   { RANDOMISING-MODULE   module-name [ANCHOR-POINTS number]         ]
  [   { RANDOMIZING-MODULE               [RELATIVE-BLOCK-MAXIMUM relative-block]
  [                                      [INSERTION-BYTES-MAXIMUM bytes] ]

  [DATASETS PRIME ddname [BLOCK size DEVICE device [SCAN cylinders]     ]
  [   [FREQUENCY-FREE-BLOCKS frequency] [PERCENTAGE-FREE-SPACE percent] ]

  [CONTAINS physical-segment-name                                       ]
  [         [,physical-segment-name PARENT physical-segment-name]...]

[common clauses]

{; }
{. }
```

where:

`module-name` is the name of a MODULE.

`number` is an unsigned integer in the range 1 to 255, being the number of root anchor points required in each control interval or block.

`relative-block` is an unsigned integer in the range 1 to 16777215, being the maximum block number to be produced by the randomizing module.

`bytes` is an unsigned integer in the range 1 to 16777215, being the maximum number of bytes to be inserted into the root addressable area.

`ddname` is 1- to 8-alphanumeric characters, being the logical name used in the job control to identify the physical file.

`size` is an unsigned integer other than 0, being the number of bytes required per physical block or control interval.

`device` is one of the keywords or numbers from the list:

DRUM	2311	3310	3350 3390
CELL	2314	3330	3370
2301	2319	3340	3375
2305	2321	3344	3380

`cylinders` is an unsigned integer in the range 0 to 32767.

`frequency` is an unsigned integer in the range 2 to 100, or is 0.

`percent` is an unsigned integer in the range 0 to 99.

`physical-segment-name` is the name of a PHYSICAL segment.

`common clauses` are any of these clauses (as defined in the *ASG-Manager Products Dictionary/Repository User's Guide*):

ACCESS-AUTHORITY	FREQUENCY
ADMINISTRATIVE-DATA	NOTE
ALIAS	OBSOLETE-DATE
CATALOG	QUERY
COMMENT	SECURITY-CLASSIFICATION
DESCRIPTION	SEE
EFFECTIVE-DATE	

Remarks

1. The HDAM keyword must immediately follow the member type identifier to indicate that a HDAM database is being defined.

2. The ACCESS clause can be omitted, but if it is present, it must immediately follow the HDAM keyword.

3. The RANDOMIZING-MODULE (or RANDOMISING-MODULE) clause specifies the user-supplied randomizing module that is used to store and access the segments in this database.

4. The optional clauses ANCHOR-POINTS, RELATIVE-BLOCK-MAXIMUM, and INSERTION-BYTES-MAXIMUM specify the maximum values for the operands that are required when accessing the root addressable area of the HDAM database.

5. The DATASETS clause defines a dataset group within this database. It must be present if the definition of the database is to be complete. The database can be divided into up to 10 dataset groups.

6. Within the DATASETS clause, the PRIME clause must always be specified. It defines the prime file of the dataset.

7. The ddname in each PRIME clause must be unique in the data dictionary.

8. BLOCK size must not be greater than 4096. If the size is not a multiple of 512, on encoding it is rounded up to the next multiple of 512.

9. The DEVICE clause specifies the physical storage device for the dataset file.

10. The SCAN clause specifies the number of cylinders to be scanned when searching for available storage space. If the SCAN clause is omitted, a default of 3 cylinders is assumed.

11. The FREQUENCY-FREE-BLOCKS clause specifies that, where frequency = *n*, every *n*th control interval or block in this dataset group is to be left as free space during database load or reorganization.

12. The PERCENTAGE-FREE-SPACE clause specifies the minimum percentage of each control interval or block that is to be left as free space in this data file during database load or reorganization.

13. The CONTAINS clauses list the segments that reside in the database. For the definition of the database to be complete, the CONTAINS clauses must be present, and each CONTAINS clause must immediately follow a DATASETS clause.

14. You can specify 1 to 255 different segments in total for the database. The segments must be specified in hierarchal sequence; that is, from top to bottom and left to right.

15. The first `physical-segment-name` listed in the first CONTAINS clause must be the name of the root segment.

16. The PARENT clauses identify the physical parents of the segments whose names are listed in the CONTAINS clauses. A PARENT clause must not be present for the root segment (the first `physical-segment-name` of the first dataset group) but must follow each of the other names listed in the CONTAINS clauses.

17. Common clauses can be present in any type of data definition statement; they are therefore defined separately, in the *ASG-Manager Products Dictionary/Repository User's Guide*. Not more than one of each of these clauses can be declared. If a common clause has a subordinate clause or keyword, the subordinate clause identifier or subordinate keyword must not be truncated to an extent where it becomes ambiguous with any other clause identifier or other keyword available in the data definition syntax for this member type.

18. The common clauses can be declared in any order. If present, they must follow any ACCESS, DATASETS, and CONTAINS clauses.

19. A record containing the database's data definition statement can be inserted into the data dictionary's source dataset by a suitable command (see the *ASG-Manager Products Dictionary/Repository User's Guide*), and an encoded record can subsequently be generated and inserted into the data entries dataset. When the encoded record is generated, a data entries record of a special internal type, a DL/I-DATASET member, is created for each ddname that appears in the database's data definition. The DL/I-DATASET internal member is given a uses table entry for each segment that constitutes the dataset defined by the member. The DL/I-DATASET internal member can be referred to by other members, for example, it could be used in the INPUTS clause of a PROGRAM data definition. DL/I-DATASET members can also be interrogated (see "DL/I Member-type Keywords" on page 99). If, when the encoded record is generated, any segment or module whose name appears in the database's data definition statement has no data entries record, a dummy data entries record for that member is created as a dummy segment record or a dummy module record respectively.

20. When an encoded database member is deleted, any DL/I-DATASET member created for it that is not referred to by other members is also deleted, together with any references that the DL/I-DATASET member made to segments. Any DL/I-DATASET member that is referred to by other members is made into a dummy member rather than being deleted.

Examples

The following example of data definition statements for HDAM databases relate to the hierarchical structure of segments illustrated in "Specification of the Data Definition Statement for a HSAM Type DL/I Database" on page 57. In this example, meaningful segment names have been retained. The abbreviated 8-character names required by DL/I can be defined as DL/I aliases in the ALIAS clauses of the members that constitute the database.

The database could be defined thus:

```
ADD EMPLOYEE-DETAILS;
IMS-DATABASE HDAM
ACCESS VSAM PASSWORD RANDOMISING-MODULE RANDMOD
                      ANCHOR-POINTS 10
                      RELATIVE-BLOCK-MAXIMUM 25600
                      INSERTION-BYTES-MAXIMUM 512
DATASET PRIME EMPL BUFFER 2048
                   DEVICE 3330 MODEL 11
                   SCAN 5
                   FREQUENCY-FREE-BLOCKS 10
                   PERCENTAGE-FREE-SPACE 10
CONTAINS DEPARTMENT,
         EMPLOYEE-NUMBER PARENT DEPARTMENT,
         NAME PARENT EMPLOYEE-NUMBER,
         ADDRESS PARENT EMPLOYEE-NUMBER,
         JOB-STATUS PARENT EMPLOYEE-NUMBER,
         SALARY PARENT JOB-STATUS,
         TAXCODE PARENT SALARY,
         DEDUCTION-TABLE-REF PARENT SALARY,
         SOCIAL-SECURITY-NUMBER PARENT SALARY,
         JOB-TITLE PARENT JOB-STATUS
;
```

Specification of the Data Definition Statement for a HIDAM Type DL/I Database

Syntax

```
{ IMS-DATABASE   } HIDAM
{ DL/I-DATABASE  }
{ DL/1-DATABASE  }
{ DLI-DATABASE   }
{ DL1-DATABASE   }

  [ACCESS [VSAM]   [INDEX [VSAM] [DATABASE database-name] [SEGMENT segment-name]]
                   [SEQUENCE-KEY sequence-key-name]]

  [DATASETS INDEX ddname [BLOCK count [RECORD length]] DEVICE device]

  [DATASETS PRIME ddname [BLOCK size] DEVICE device [SCAN cylinders]
      [FREQUENCY-FREE-BLOCKS frequency] [PERCENTAGE-FREE-SPACE percent]]

  [CONTAINS physical-segment-name
            [,physical-segment-name PARENT physical-segment-name]...]

[common clauses]

{; }
{. }
```

where:

database-name is 1 to 7 alphanumeric characters, being the IMS name of the primary index database associated with this HIDAM database.

segment-name is 1 to 8 alphanumeric characters, being the name of a segment of the primary index that is associated with this HIDAM database.

sequence-key-name is 1 to 8 alphanumeric characters, being the sequence key name of the primary index database associated with this HIDAM database.

ddname is 1 to 7 alphanumeric characters, being the logical name used in the job control to identify the physical file.

count is an unsigned non-zero integer, being the number of logical records per physical block.

length is an unsigned non-zero integer, being the maximum length (in bytes) of a logical record. If VSAM is the operating system access method, length must be an even value.

size is an unsigned non-zero integer, being the number of bytes required per physical block or control interval.

device is one of the keywords or numbers from the list:

DRUM	2311	3310	3350	3390
CELL	2314	3330	3370	
2301	2319	3340	3375	
2305	2321	3344	3380	

cylinders is an unsigned integer in the range 0 to 32767.

frequency is an unsigned integer in the range 2 to 99, or is 0.

percent is an unsigned integer in the range 0 to 99.

physical-segment-name is the name of a PHYSICAL segment.

`common clauses` are any of these clauses (as defined in the *ASG-Manager Products Dictionary/Repository User's Guide*):

ACCESS-AUTHORITY	FREQUENCY
ADMINISTRATIVE-DATA	NOTE
ALIAS	OBSOLETE-DATE
CATALOG	QUERY
COMMENT	SECURITY-CLASSIFICATION
DESCRIPTION	SEE
EFFECTIVE-DATE	

Remarks

1. In Manager Products, a primary index database is not handled as a separate data dictionary member, but is considered to be part of its corresponding HIDAM database. Consequently, the definition of the primary index database is included in the definition of the HIDAM database.

2. The name of the primary index database and the names of its segment and sequence key can be specified:

   - In the PRODUCE command, when DBD control statements for the primary index database are generated by the Source Language Generation facility. These are generated automatically after DBD control statements for the HIDAM database are generated. (See "Generating DL/I DBD Control Statements" on page 131.)

   - In the ACCESS clause of the HIDAM database definition. If different names are specified for the same entity in the PRODUCE command and the ACCESS clause, the name in the PRODUCE command is applied.

3. Names specified in the ACCESS clause do not result in the generation of dummy members.

4. If neither the PRODUCE command nor the ACCESS clause contains a name for the primary index database, the name of the HIDAM database with a suffix I is used as the primary index database name when DBD control statements are generated by the Source Language Generation facility.

5. If neither the PRODUCE command nor the ACCESS clause contains a name for the segment of the primary index database, the name of the root segment of the HIDAM database with a suffix I is used as the name of the segment of the primary index database when DBD control statements are generated by the Source Language Generation facility.

6. If neither the PRODUCE command nor the ACCESS clause contains a name for the sequence key of the primary index database, the name of the sequence key of the HIDAM root segment with a suffix I is used as the name of the sequence key of the primary index database when Database Description (DBD) control statements are generated by the Source Language Generation facility.

7. The HIDAM keyword must immediately follow the member type identifier to indicate that a HIDAM database is being defined.

8. The ACCESS clause can be omitted; but, if it is present, it must immediately follow the HIDAM keyword.

9. The INDEX subordinate clause in the ACCESS clause specifies the operating system access method for, and/or the names to be applied to, the primary index database (see remark 1 on page 71 through remark 6 on page 72). The DATABASE, SEGMENT, and SEQUENCE-KEY clauses may, if present, be in any order within the INDEX clause, but must not precede the VSAM keyword, if that is present.

10. Each DATASETS clause defines the data files within the primary index database and within the HIDAM database. These clauses must be present if the definition of the databases is to be complete. For each of these databases only one data file can be defined.

11. The DATASETS clause that defines the data file for the primary index database has no associated CONTAINS clause. The subordinate INDEX ddname clause defines the prime data file for the primary index database. The ddname must be unique in the data dictionary.

12. For the primary index database, the control interval size specified by the product of the BLOCK count and RECORD length must not exceed 4096.

13. Each DEVICE clause specifies the physical storage device for the data file defined by its containing DATASETS clause.

14. The DATASETS clause for the HIDAM database must include a PRIMARY ddname clause. It defines the prime data file for the HIDAM database. The ddname must be unique in the data dictionary.

15. If a BLOCK is specified in the PRIME clause, `size` must not be greater than 4096. If `size` is not a multiple of 512, on encoding it is rounded up to the next multiple of 512.

16. The SCAN clause specifies the number of cylinders to be scanned when searching for available storage space. If the SCAN clause is omitted, a default of 3 cylinders is assumed.

17. FREQUENCY-FREE-BLOCKS specifies that, where frequency =*n*, every *n*th control interval or block in this dataset group is to be left as free space during database load or reorganization.

18. PERCENTAGE-FREE-SPACE specifies the minimum percentage of each control interval or block that is to be left as free space in this dataset group during database load or reorganization.

19. The DATASETS clause for the HIDAM database is immediately followed by a CONTAINS clause listing the segments that constitute the data file. For the definition of the database to be complete, the CONTAINS clause must be present, and must immediately follow the DATASETS clause.

20. You can specify 1 to 255 different segments for the HIDAM database. The segments must be specified in hierarchal sequence; that is, from the top to the bottom and left to right.

21. The first `physical-segment-name` listed in the first CONTAINS clause must be the name of the root segment.

22. The PARENT clauses identify the physical parents of the segments whose names are listed in the CONTAINS clauses. A PARENT clause must not be present for the root segment, but must follow each of the other names listed in the CONTAINS clauses.

23. Common clauses can be present in any type of data definition statement; they are therefore defined separately in the *ASG-Manager Products Dictionary/Repository User's Guide*. Not more than one of each of these clauses can be declared. If a common clause has a subordinate clause or keyword, the subordinate clause identifier or subordinate keyword must not be truncated to an extent where it becomes ambiguous with any other clause identifier or other keyword available in the data definition syntax for this member type.

24. The common clauses can be declared in any order. If present, they must follow the ACCESS, DATASETS, and CONTAINS clauses, if these are present.

25. A record containing the database's data definition statement can be inserted into the data dictionary's source dataset by a suitable command (see the *ASG-Manager Products Dictionary/Repository User's Guide*), and an encoded record can subsequently be generated and inserted into the data entries dataset.

    If, when the encoded record is generated, a data entries record of a special internal type, a DL/I-DATASET member, is created for each ddname that appears in the database's data definition. The DL/I-DATASET internal member is given a uses table entry for each segment that constitutes the dataset defined by the member. The DL/I-DATASET internal member can be referred to by other members, for example, it could be used in the INPUTS clause of a PROGRAM data definition. DL/I-DATASET members can also be interrogated (see "DL/I Member-type Keywords" on page 99).

    If, when the encoded record is generated, any segment whose name appears in the database's data definition statement has no data entries record, a dummy data entries record is created for that member as a dummy segment record.

26. When an encoded database member is deleted, any DL/I-DATASET member created for it that is not referred to by other members is also deleted, together with any references that the DL/I-DATASET member made to segments. Any DL/I-DATASET member that is referred to by other members is made into a dummy member rather than being deleted.

Example

This example of data definition statements for HIDAM database relate to the hierarchical structure of segments illustrated in "Specification of the Data Definition Statement for a HSAM Type DL/I Database" on page 57. In this example, meaningful segment names have been retained. The abbreviated 8-character names required by IMS (DL/I) can be defined as IMS aliases in the ALIAS clauses of the members that constitute the database.

This example illustrates the specification of the VSAM access method for both the HIDAM database and its primary index database. The keywords DOS-COMPATIBLE and PASSWORD that are included are applicable to the primary index database. The first DATASETS clause defines the dataset group for the primary index database. The segments constituting the HIDAM database are all contained in one primary dataset group (defined by the second DATASETS clause with its associated CONTAINS clause).

```
ADD EMPLOYEE-DETAILS;
DL/I-DATABASE HIDAM
DATASETS INDEX EMPLI BLOCK 4
                      DEVICE 3330
DATASETS PRIME EMPL BLOCK 2048
                    DEVICE 3330
                    SCAN 5
                    FREQUENCY-FREE-BLOCKS 10
                    PERCENTAGE-FREE-SPACE 10
CONTAINS DEPARTMENT,
         EMPLOYEE-NUMBER PARENT DEPARTMENT,
         NAME PARENT EMPLOYEE-NUMBER,
         ADDRESS PARENT EMPLOYEE-NUMBER,
         JOB-STATUS PARENT EMPLOYEE-NUMBER,
         SALARY PARENT JOB-STATUS,
         TAXCODE PARENT SALARY,
         DEDUCTION-TABLE-REF PARENT SALARY,
         SOCIAL-SECURITY-NUMBER PARENT SALARY,
         JOB-TITLE PARENT JOB-STATUS
;
```

Specification of the Data Definition Statement for a LOGICAL Type DL/I Database

Syntax

```
{ IMS-DATABASE  }  LOGICAL
{ DL/I-DATABASE }
{ DL/1-DATABASE }
{ DLI-DATABASE  }
{ DL1-DATABASE  }
  [CONTAINS segment-name                          ]
  [         [,segment-name PARENT segment-name]   ]
[common clauses]

{; }
{. }
```

where:

`segment-name` is the name of a logical or physical segment.

`common clauses` are any of these clauses (as defined in the *ASG-Manager Products Dictionary/Repository User's Guide*):

ACCESS-AUTHORITY	FREQUENCY
ADMINISTRATIVE-DATA	NOTE
ALIAS	OBSOLETE-DATE
CATALOG	QUERY
COMMENT	SECURITY-CLASSIFICATION
DESCRIPTION	SEE
EFFECTIVE-DATE	

Remarks

1. The keyword LOGICAL must immediately follow the member type identifier to indicate that a LOGICAL database is being defined.

2. The CONTAINS clause must be present if the definition of the database is to be complete. It lists the segments that reside in the LOGICAL database.

3. One to 255 different segments can be specified in total for the LOGICAL database. They may be either logical segments and/or physical segments.

4. If a logical segment is specified in the CONTAINS clause of the LOGICAL database, then when DBD control statements are generated, a SEGM is generated statement with the NAME operand equal to the name of the logical segment, and the SOURCE operand(s) equal to the name(s) of the physical segment(s) specified in the logical segment definition.

5. If a physical segment is specified in the CONTAINS clause of the LOGICAL database, then when DBD control statements are generated, a SEGM statement is generated with both the NAME operand and the SOURCE operand equal to the name of the physical segment.

6. The segments must be specified in hierarchical sequence; that is, from top to bottom and left to right.

7. The first `segment-name` listed in the CONTAINS clause must be the name of the root segment.

8. The PARENT clauses identify the segments that represent the physical parents of the segments whose names are listed in the CONTAINS clause. A PARENT clause must not be present for the root segment, but must follow each of the other names listed in the CONTAINS clause.

9. The root segment specified must represent a segment that is the root segment in the physical database in which it resides.

10. The hierarchy of dependent segments must be the same as the hierarchy of segments that they represent, as defined for the physical database in which the segments reside.

11. Logical segments that depend on the same parent segment may not represent the same physical segment.

12. Logical concatenated segments can be specified, to obtain access to destination parents in logical relationships.

13. Specifying a logical concatenated segment also enables logical relationships to be crossed; that is, access to the segments in the physical hierarchical path of the destination parent (as specified in the definition of the physical database in which that destination parent resides) can be obtained either in the downward or upward direction. This is enabled by specifying the segments, which may be either the physical segments themselves or the logical segments representing the physical segments, as dependents of the logical concatenated segment in the logical database. That is, the physical or logical segments representing the physical child and the physical parent as specified in its physical database, can be specified as physical dependents of the logical concatenated segment. (This does not apply if the physical child segment is paired with the logical child in the concatenated segment.) The hierarchy of the segments in the logical database must still be the same as the hierarchy of the segments that they represent in the physical database, except that if the hierarchical path in the upward direction is specified, the relative order of the segments is reversed.

14. Although the dependent segments of a concatenated segment may be intermixed, their left to right order, as defined in their respective physical databases, must be maintained. This applies also to the dependents of non-concatenated segments.

15. A physical target segment or a logical segment representing a target segment must be the root segment of the logical database if the target segment is to be accessed through a secondary index.

16. If the logical database contains either of these types of segment:

    - A directly contained physical segment
    - A logical segment that specifies a physical segment, but does not also specify a physical database in its data definition

    then when the Source Language Generation facility is used to produce DBD control statements, the corresponding physical database is found in one of these ways, and its name is output in the SEGM statement:

    - If the physical segment resides in only one physical database, then the name of that physical database is output in the SEGM statement.
    - If the physical segment resides in more than one physical database and also represents the root segment of the logical of the logical database, then the name of the first physical database that DataManager encounters in the physical segment's used-by table is output in the SEGM statement.
    - If the physical segment resides in more than one physical database and does not represent the root segment in the logical database then a physical database is selected in one of these ways:
        - — If the physical database named in the preceding SEGM statement appears anywhere in the used-by table of the physical segment currently being processed, then the name of this physical database is output in the SEGM statement for the current segment.
        - — If the previous SEGM statement was for a concatenated segment, Manager Products first searches for the logical child's physical database in the used-by table of the segment currently being processed, and if found, the name of this physical database is output in the SEGM statement.
        - — If the logical child's physical database cannot be found in the used-by table, Manager Products searches for the destination parent's physical database, and, if found, outputs its name in the SEGM statement.
        - — If the physical database(s) name(s) were output in the previous SEGM statement cannot be found in the used-by table of the physical segment currently being processed, then the name of the first physical database encountered in the used-by table is output in the SEGM statement.

17. Common clauses are defined in the *ASG-Manager Products Dictionary/Repository User's Guide*. Not more than one of each of these clauses can be declared. If a common clause has a subordinate clause or keyword, the subordinate clause identifier or subordinate keyword must not be truncated to an extent where it becomes ambiguous with any other clause identifier or other keyword available in the data definition syntax for this member type.

18. The common clauses can be declared in any order. If present, they must follow the CONTAINS clause, if that clause is present.

19. A record containing the database's data definition statement can be inserted into the data dictionary's source dataset by a suitable command (see the *ASG-Manager Products Dictionary/Repository User's Guide*), and an encoded record can subsequently be generated and inserted into the data entries dataset. If, when the encoded record is generated, any segment whose name appears in the database's data definition has no data entries record, a dummy data entries record is created for that member, as a dummy segment record.

Example

The following example of a LOGICAL database member definition relates to the hierarchical structure of segments illustrated in "Specification of the Data Definition Statement for a HSAM Type DL/I Database" on page 57. In these examples meaningful segment names have been retained. The abbreviated 8-character names required by IMS (DL/I) can be defined as IMS aliases in the ALIAS clauses of the members that constitute the database.

```
ADD EMPLOYEE-DETAILS;
IMS-DATABASE LOGICAL
CONTAINS DEPARTMENT,
         EMPLOYEE-NUMBER PARENT DEPARTMENT,
         NAME PARENT EMPLOYEE-NUMBER,
         ADDRESS PARENT EMPLOYEE-NUMBER,
         JOB-STATUS PARENT EMPLOYEE-NUMBER,
         SALARY PARENT JOB-STATUS,
         TAXCODE PARENT SALARY,
         DEDUCTION-TABLE-REF PARENT SALARY,
         SOCIAL-SECURITY-NUMBER PARENT SALARY,
         JOB-TITLE PARENT JOB-STATUS
;
```

Specification of the Data Definition Statement for a SECONDARY-INDEX Type DL/I Database

Syntax

```
{ IMS-DATABASE   } { INDEX           }
{ DL/I-DATABASE  } { SECONDARY-INDEX }
{ DL/1-DATABASE  }
{ DLI-DATABASE   }
{ DL1-DATABASE   }

  [ACCESS [VSAM]]

  [DATASETS PRIME ddname [BLOCK count [RECORD length]]                 ]
           [OVERFLOW ddname [BLOCK count [RECORD length]]]             ]
            DEVICE device                                              ]

  [CONTAINS index-pointer-segment-name]

[common clauses]

{; }
{. }
```

where:

ddname is 1- to 7-alphanumeric characters, being the logical name used in the job control to identify the physical file.

count is an unsigned non-zero integer, being the number of logical records per physical block.

length is an unsigned non-zero integer, being the maximum length (in bytes) of a logical record.

device is one of the keywords or numbers from the list:

DRUM	2314	3330	3370
CELL	2319	3340	3375
2301	2321	3344	3380
2305	3310	3350	FBA
2311			

index-pointer-segment is an INDEX-POINTER-SEGMENT member.

common clauses are any of these clauses (as defined in the *ASG-Manager Products Dictionary/Repository User's Guide*):

ACCESS-AUTHORITY	FREQUENCY
ADMINISTRATIVE-DATA	NOTE
ALIAS	OBSOLETE-DATE
CATALOG	QUERY
COMMENT	SECURITY-CLASSIFICATION
DESCRIPTION	SEE
EFFECTIVE-DATE	

Remarks

1. One of the keywords INDEX or SECONDARY-INDEX must immediately follow the member type identifier to indicate that a secondary index type database is being defined.

2. If the ACCESS clause is present, it must immediately follow the INDEX or SECONDARY-INDEX keyword.

3. The operating system access method is VSAM. This can be explicitly stated by a VSAM keyword in the ACCESS clause, or, if the keyword is not present, is assumed.

4. The DATASETS clause defines data files that constitute the secondary index database. It must be present if the definition of the database is to be complete.

5. The PRIME clause specifies the prime data file of the database.

6. The OVERFLOW clause specifies the overflow data file of the database. This clause must be specified if the index pointer segments contain non-unique keys.

7. The ddname in the PRIME clause and the ddname in the OVERFLOW clause, which specify the logical file names of the respective data files, must each be unique in the data dictionary.

8. If an OVERFLOW clause and a PRIME clause are both present, then if a BLOCK subordinate clause is present in either, a BLOCK subordinate clause must be present in both; in which case, if an associated RECORD subordinate clause is present in either, a RECORD clause must be present in both.

9. The RECORD length specified for the OVERFLOW clause must be equal to or greater than the RECORD length specified for the PRIME clause, if both are specified.

10. The control interval size, specified either by the BUFFER size or by the product of the BLOCK count and the RECORD length, must not exceed 4096.

    If the control interval size is specified, then on encoding, a warning message is output if the product is not a multiple of 512.

11. The DEVICE clause specifies the physical storage device for the data files.

12. The contains clause specifies the index pointer segment that is contained in the secondary index database. For the definition of the database to be complete, the CONTAINS clause must be present, and must immediately follow the DATASETS clause.

13. Common clauses, listed under Syntax above, can be present in any type of data definition statement; they are therefore defined separately, in the *ASG-Manager Products Dictionary/Repository User's Guide*. Not more than one of each of these clauses can be declared. If a common clause has a subordinate clause identifier or a subordinate keyword, the subordinate clause or keyword must not be truncated to an extent where it becomes ambiguous with any other clause identifier or other keyword available in the data definition syntax for this member type.

14. The common clauses can be declared in any order. If present, they must follow the ACCESS, DATASETS, and CONTAINS clauses, if these are present.

15. A record containing the database's data definition statement can be inserted into the data dictionary's source dataset by a suitable command (see the *ASG-Manager Products Dictionary/Repository User's Guide*), and an encoded record can subsequently be generated and inserted into the data entries dataset.

    When the encoded record is generated, a data entries record of a special internal type, a DL/I-DATASET member, is created for each ddname that appears in the database's data definition. The DL/I-DATASET internal member is given a uses table entry for each segment that constitutes the dataset defined by the member. The DL/I-DATASET internal member can be referred to by other members, for example, it could be used in the INPUTS clause of PROGRAM data definition. DL/I-DATASET members can also be interrogated (see "Condition Keywords for Which and What Commands" on page 102).

    If, when the encoded record is generated, any database or segment or module whose name appears in the database's data definition statement has no data entries record, a dummy data entries record is created for that member, as a dummy database record or a dummy segment record or a dummy module record respectively.

16. When an encoded database member is deleted, any DL/I-DATASET member created for it that is not referred to by other members is also deleted, together with any references that the DL/I-DATASET member made to segments. Any DL/I-DATASET member that is referred to by other members is made into a dummy member rather than being deleted.

Example

This is an example of a DataManager data definition statement for a secondary index database:

```
ADD EMPIND;
DL/I-DATABASE SECONDARY-INDEX
DATASETS PRIME EMPIP BLOCK 4 RECORD 256
         OVERFLOW EMPIO BLOCK 8 RECORD 256
         DEVICE 3340
```

DataManager Data Definition Statements for DL/I Program Communication Blocks

Syntax

```
{ PROGRAM-COMMUNICATION-BLOCK }    STRUCTURE
{ PCB                         }

{ NAME pcb-name                                                                          }
{ processing-options-1                                                                   }
{                                                                                        }
{ [ { SINGLE-POSITIONING } ]  [DATABASE database-name]  [KEYLENGTH keylength]            }
{   [ { MULTI-POSITIONING } ]                                                            }
{                                                                                        }
{ [ SEGMENT segment-name [processing-options-2]                                        ] }
{   [ [SECONDARY-SEQUENCE [ON index-pointer-segment]]                                  ] }
{   [ [SENSITIVE-FIELDS { sensitive-field } [ , { sensitive-field } ]...  ]          ] ... }
{   [                   { filler-bytes    }   [ { filler-bytes    } ]      ]          }

[common clauses]

{ ; }
{ . }
```

where:

pcb-name is the name of another PROGRAM-COMMUNICATION-BLOCK member (the reference PCB) of which the data definition (excluding any common clauses) is to be regarded as being also a data definition of this member.

processing-options-1 is:

```
{ BY   }{ LOAD  [[,] ASCENDING] [[,] EXCLUSIVE]                                  }[[,] PATH]
{ WITH }{ GET   { [[,] ONLY]                                                     }
        {       { [[,] REPLACE] [[,] DELETE] [[,] INSERT]                        }
        {       { [[,] ASCENDING]  [[,] EXCLUSIVE]                               }
        { INSERT  [[,] ASCENDING] [[,] EXCLUSIVE]                                }
        { UPDATE  [[,] EXCLUSIVE]                                                }
```

database-name is the name of a DATABASE member.

keylength is an integer in the range 0 to 32767.

segment-name is the name of a SEGMENT member.

processing-options-2 is:

```
{ BY   }{GET    [[,] REPLACE] [[,] DELETE] [[,] INSERT]      }
{ WITH }{INSERT                                             }
        {UPDATE                                             }
[[,] EXCLUSIVE [[,] PATH]
```

index-pointer-segment is the name of an INDEX-POINTER SEGMENT member.

sensitive-field is:

```
{group-name              }   ENTERED-AS       [version]
{item-name               }   HELD-AS
{sequence-key-name       }   REPORTED-AS
{concatenated-key-name   }   DEFAULTED-AS
[SUBFIELDS] [EXIT-ROUTINE {module-name  }]
                          {program-name }
                          {system-name  }
[{ REPLACE   }]   [KNOWN-AS logical-name]
 {NOREPLACE  }
 {NO-REPLACE }
```

where:

group-name is the name of a GROUP member.

item-name is the name of an ITEM member.

sequence-key-name is the name of a SEQUENCE KEY member.

concatenated-key-name is the name of a CONCATENATED KEY member.

version is an unsigned integer in the range of 1 to 15, being a number specifying which version of the relevant item is relevant to this sensitive field. The *version* is within the stated form (ENTERED-AS, HELD-AS, or REPORTED-AS). If *version* is omitted or if the stated version does not exist in the stated form, the lowest numbered existing version in that form is assumed to be relevant.

module-name is the name of a MODULE member.

program-name is the name of a PROGRAM member.

system-name is the name of a system member.

`local-name` is a name, conforming to the rules for member names stated in the *ASG-Manager Products Dictionary/Repository User's Guide*, that can be used instead of the name or alias of the sensitive field, when PSB control statements or record layouts or source language data descriptions are generated from this data definition by the DataManager Source Language Generation facility. The `local-name` is not separately recorded in the data dictionary (i.e., no dummy data entries record and no index record is created for `local-name` when the data definition in which it appears is encoded) so `local-name` cannot be interrogated and can be the same as another name, an alias, or a catalog classification in the data dictionary. The `local-name` is the name by which the member forming the sensitive field is known only within the PCB defined by this data definition.

`filler-bytes` is an unsigned integer.

`common clauses` are any of these clauses (as defined in the *ASG-Manager Products Dictionary/Repository User's Guide*):

ACCESS-AUTHORITY	FREQUENCY
ADMINISTRATIVE-DATA	NOTE
ALIAS	OBSOLETE-DATE
CATALOG	QUERY
COMMENT	SECURITY-CLASSIFICATION
DESCRIPTION	SEE
EFFECTIVE-DATE	

Remarks

1. The member type identifiers PROGRAM-COMMUNICATION-BLOCK and PCB are synonymous.

2. A PCB member must be defined for each logical data structure used by any application for which PCB control statements or PCB masks are to be generated.

3. The STRUCTURE keyword must immediately follow the member type identifier to indicate that the application view of a logical data structure is being defined. All segments in the logical data structure must belong to the same database.

4. The NAME clause specifies that the data definition of the reference PCB, `pcbname`, is to be regarded as being also a data definition of this member; with the exception that the pcb-name's common clauses are not applied to this member.

5. Unless the NAME clause is specified, the first subordinate clause within the STRUCTURE clause must be the *processing-options-1* clause. This clause specifies processing options for the segments that constitute the logical data structure. Those segments are specified by the SEGMENT subordinate clauses, within each of which overriding processing options applicable to the particular segment can be specified by *processing-options-2*. For each segment for which this is not specified, *processing-options-1* applies.

6. The *processing-options-1* clause defines the functions that can be performed on the logical data structure from the application view (except where overridden for individual segments by *processing-options-2*). These can be:

   - Database loading
   - Database reading only
   - Database reading and limited updating
   - Adding information to an existing database
   - Database reading and all updating functions

7. For database loading, the LOAD keyword is specified. The LOAD function is not valid for a logical data structure that belongs either to a SECONDARY-INDEX database or to a LOGICAL database. LOAD is also invalid if the secondary processing sequence is to be used to access the logical data structure. If *processing-options-1* specifies LOAD for a logical data structure belonging to a HISAM or HIDAM database, then all other PCB members affecting the same database within an application view must also specify LOAD.

8. For database reading only, without enqueueing to check the availability of segments, GET is specified.

9. For database reading only, with enqueueing to check the availability of segments, GET is specified.

10. For database reading (with enqueueing) and limited updating, GET is specified followed by whichever of the keywords REPLACE, DELETE, and/or INSERT are relevant, in any order; but not more than 3 of the keywords REPLACE, DELETE, INSERT, ASCENDING, EXCLUSIVE, and PATH can follow GET. These rules apply:

    - INSERT is invalid if the logical data structure belongs to a HSAM database or a SECONDARY-INDEX database.
    - DELETE and REPLACE are invalid if the logical data structure belongs to a HSAM database.

11. For adding new occurrences of a segment to a database, INSERT is specified. INSERT is invalid if the logical data structure belongs to a HSAM database or a SECONDARY-INDEX database.

12. For database reading and all updating functions, UPDATE is specified. (UPDATE is thus the equivalent of GET, REPLACE, DELETE, and INSERT.)

13. The ASCENDING keyword, if present, specifies that the segments are processed in ascending sequence only. These rules apply:
    - ASCENDING is not valid with GET ONLY or UPDATE.
    - If LOAD is specified, ASCENDING is valid for a logical data structure that belongs to a HIDAM database or to a HDAM database, but is invalid for all other logical data structures.
    - If the logical data structure belongs to a HIDAM database, and LOAD is specified, ASCENDING is assumed whether the keyword is present or not.
14. The EXCLUSIVE keyword, if present, specifies that online programs can have exclusive use of the logical data structure. EXCLUSIVE is not valid with GET ONLY.
15. The PATH keyword, if present, specifies that the command mode for path calls is used to process the logical data structure. It can be used by DL/I to determine the maximum length of the input/output area.
16. The keywords ASCENDING, EXCLUSIVE, and PATH can, if present, be in any order.
17. If either of the keywords SINGLE-POSITIONING or MULTI-POSITIONING is present, it must immediately follow the `processing-options-1` clause. It specifies the type of positioning required for the logical data structure. If neither of these keywords is present, SINGLE-POSITIONING is assumed. MULTI-POSITIONING is invalid if the logical data structure belongs to a HSAM database.
18. If the STRUCTURE clause is specified, the DATABASE subordinate clause must be specified if the logical data structure resides in a database, whose segments are contained by more than one database. Otherwise, the DATABASE clause is optional.
19. The KEYLENGTH clause specifies the maximum concatenated key length for any path of sensitive segments that is used by the application which uses the PCB.

    Note:

    If KEYLENGTH is not specified, the maximum concatenated key length will be calculated when the PSB is generated. This calculation may cause significant input/output activity; to avoid this, ASG recommends that you specify KEYLENGTH.

20. All segments in the logical data structure must belong to the same database.
21. A SEGMENT clause must be specified for each segment in the logical data structure to which the application is sensitive.
22. The segment clauses for a logical data structure can be in any order. When PCB control statements are produced by the Source Language Generation facility, the segments specified are organized into the order specified in the database data definition statement. When this reorganization takes place, DataManager includes in the new order any segments that are in the hierarchal path to the sensitive statements, but for which no SEGMENT clauses have been specified.

23. In each SEGMENT clause, *segment-name* must immediately follow the SEGMENT keyword, to identify the sensitive segment to be processed.

24. If the LOAD processing option is specified for the logical data structure, then SEGMENT clause must not be entered for virtual logical child segments.

25. From 1 to 255 SEGMENT clauses can be defined for a logical data structure. Only one SEGMENT clause may be specified for each segment.

26. *processing-options-2*, if present, must immediately follow the segment name. It specifies the functions that can be performed on the segment from the application view. If the processing options specified by *processing-options-1* can apply to the segment, this can be omitted. If *processing-options-1* specified LOAD, *processing-options-2* must be omitted.

27. GET, INSERT, and UPDATE, and the optional keywords that can be associated with them, have the same meanings and restrictions as are specified for *processing-options-1* in remark 9 on page 85 through remark 16 on page 86, but applying to the one segment only. GET ONLY and ASCENDING cannot be specified in *processing-options-2*.

28. The keyword SECONDARY-SEQUENCE specifies that the logical data structure is processed through a secondary processing sequence, of which this segment is the root segment. The keyword, if present, must immediately follow *processing-options-2*, if specified; otherwise, it must, if present, immediately follow *segment-name*.

29. If SECONDARY-SEQUENCE is specified, *segment-name* must identify an index target segment, or a logical segment representing an index target segment. In a physical database, the segment must be the root segment. When processing a secondary sequence for a physical database, no logical child segments may be included.

30. The SECONDARY-SEQUENCE keyword may only be entered once for any one logical data structure.

31. The ON index-pointer-segment clause specifies the index pointer segment that indexes the index target segment. If it is omitted, the name of the relevant index pointer segment is obtained from the used-by table of the index target segment, when required for generation of DBDGEN control statements.

32. The SENSITIVE-FIELDS clause is subordinate to the SEGMENT clause. It is used by the Source Language Generation facility:

    - During the generation of PSB control statements, to generate SENFLD and VIRFLD statements that specify the fields to which the application is sensitive.
    - To generate record layouts or COBOL, PL/I, or Assembler data description statements for segment input/output areas when sensitive fields are to be processed.
    - During the generation of DBD control statements, to indicate that DBD HELD control statements are to be generated for the segment's sensitive fields only (rather than for all of the fields contained by the segment).

33. Up to a maximum of 255 sensitive fields can be declared for each segment, within a maximum of 10,000 for the PCB member.

34. The declaration of a sensitive field includes any associated keywords and clauses shown, as well as the sensitive field name. These declarations are listed in the SENSITIVE-FIELDS clause, each sensitive-field name except the first in the list being preceded by a comma and optionally additionally by spaces.

35. When the Source Language Generation facility is used to generate PSB control statements, then:

    - If the sensitive field is contained directly or indirectly by `segment-name`, it is regarded as a true sensitive field, and a SENFLD statement is generated.
    - If the sensitive field is not contained directly or indirectly by `segment-name`, it is regarded as a virtual field, and a VIRFLD statement is generated.

36. SUBFIELDS specifies that when the Source Language Generation facility is used to generate PSB control statements, SENFLD/VIRFLD statements are to be generated for each of the constituent fields of the sensitive field, as well as for the sensitive field itself. If the sensitive field is a group member and:

    - Is contained directly or indirectly by `segment-name`, then SENFLD statements are generated for each of its directly or indirectly contained group or item members
    - Is not contained directly or indirectly by `segment-name`, then:
        - VIRFLD statements are generated for each of its directly or indirectly contained group or item members that are not directly or indirectly contained by `segment-name`.
        - SENFLD statements are generated for each of its directly or indirectly contained group or item members that are directly or indirectly contained by `segment-name`.

    If the sensitive field is a sequence key member or a concatenated key member and:

    - Is defined by `segment-name`, then SENFLD statements are generated for each of its directly or indirectly contained group or item members
    - Is not defined by `segment-name`, then the SUBFIELDS keyword is ignored

37. If a VALUE operand is required on any generated VIRFLD statement (for a sensitive field or a constituent field of a sensitive field) when the Source Language Generation facility is used to generate Program Specification Block (PSB) Control Statements, then the relevant ITEM's definition should include a CONTENTS IS clause.

38. The form (ENTERED-AS, HELD-AS, REPORTED-AS, or DEFAULTED-AS) and version are relevant only when the member named in a sensitive field declaration is an ITEM. They can be included in a sensitive field declaration if:

    - The DL/I TYPE conversion facility is to be used. In this case, the new form and version to which the sensitive field is to be converted can be specified. If none is specified, the form and version of the sensitive field as it is defined in *segment-name* are assumed.
    - The field is a virtual field. In this case, the required form and version of the relevant member can be specified. If no form is specified, default assumptions are made as to the relevant form, in the order NELD-AS, DEFAULTED-AS, ENTERED-AS, and REPORTED-AS. The form first encountered in this order is taken as the defaulted form, and version is applied within this form as stated under syntax.

    The specified or defaulted form and version can be overruled by a USE or USING clause in the PRODUCE command; see "Generating DL/I PSB Control Statements" on page 137.

39. The EXIT-ROUTINE clause states the name of the member that defines a user-written field exit routine that is to be given control whenever the sensitive field is accessed.

40. Sensitive fields can be repeated provided that a KNOWN-M clause is specified for each repetition, so that unique names can be generated when COBOL, PL/I, or Assembler data description statements are generated.

41. REPLACE specifies that this field can be altered on a replace call. NOREPLACE or NO-REPLACE specifies that this field cannot be altered on a replace call.

42. If none of the keywords REPLACE, NOREPLACE or NO-REPLACE is specified, then if either of the processing options UPDATE or REPLACE has been specified, the keyword REPLACE is assumed for the sensitive field.

43. The sensitive field keywords REPLACE, NOREPLACE, and NO-REPLACE are ignored if neither of the processing options UPDATE or REPLACE is specified.

44. If the first sensitive field in a segment input/output area is not to start in the first byte position and/or if sensitive fields are not to be contiguous within the segment input/output area, filler-byte declarations must be included wherever appropriate in the list of sensitive field declarations to enable the Source Language Generation facility to calculate the start position of each field in the segment input/output area.

45. The SENSITIVE-FIELD clause is invalid if the segment is a logical child segment or a logical concatenated segment, and the processing option applicable is INSERT, LOAD, or UPDATE.

46. It is the user's responsibility to declare all the sensitive fields required by DL/I; for example, sequence key fields and segment search fields, as their start positions cannot be anticipated by DataManager.

47. Common clauses, listed under Syntax above, can be present in any type of data definition statement; they are therefore defined separately in the *ASG-Manager Products Dictionary/Repository User's Guide*. Not more than one of each of these clauses can be declared. If a common clause has a subordinate clause or keyword, the subordinate clause identifier or subordinate keyword must not be truncated to an extent where it becomes ambiguous with any other clause identifier or other keyword available in the data definition syntax for this member type.

48. The common clauses can be in any order. If present, they must follow the STRUCTURE clause.

49. A record containing the PCB's data definition statement can be inserted into the data dictionary's source data set by a suitable command (see the *ASG-Manager Products Dictionary/Repository User's Guide*), and an encoded record can subsequently be generated and inserted into the data entries data set. If, when the encoded record is generated, any PCB or database or segment or sensitive field whose name appears in the PCB's data definition statement has no data entries record, DataManager creates a dummy data entries record for that member, as a dummy PCB record, dummy database record, dummy segment or dummy item record, respectively.

Examples

See "Application View" on page 13. The member SKILLEMP-PCB is for a logical data structure that resides in a LOGICAL database. The application to which this PCB member relates is sensitive to the 3 segments SKILL, NAME and EXPR, and processes them all by the GET ONLY option.

The member AUTOREG-PCB in "Application View" on page 13 is for a logical data structure of two segments, NAMEID and CITY, that reside in a HDAM database indexed by a secondary index. The segments are processed by the GET option. SECONDARY-SEQUENCE is specified to indicate that this logical data structure is processed using a secondary sequence with the index target segment NAMEID as the root segment.

The example below shows a PCB member for a logical data structure residing in the HDAM database SKILLINV, illustrated in Figure 2 on page 6:

```
ADD SKILLINV-PCB;
PCB STRUCTURE
    BY GET, INSERT
    SEGMENT SKILLMAST
    SEGMENT SKILLNAM
    SEGMENT EXPRMAST BY INSERT PATH
    SEGMENT EDCUMAST BY INSERT
;
```

This logical data structure as a whole has processing options of GET and INSERT specified. The segment EXPRMAST has overriding options of INSERT PATH specified. For the segment EDUCMAST, the overriding option INSERT allows this segment to be in the path of segments to be inserted.

For a SENSITIVE-FIELDS example, using the segment ASY-PACK shown in Figure 3 on page 9, a PCB member could be defined thus:

```
ADD ASY-PACK-PCB;
PCB STRUCTURE
    BY GET ONLY
    SEGMENT ASY-LINE
    SEGMENT ASY-PACK
    SENSITIVE-FIELD PACK-NO, 10, QTY-REQD
;
```

DataManager System, Program, and Module Data Definition Statements for a DL/I Environment

Outline of the SYSTEM, PROGRAM, and MODULE Data Definition Statements for a DL/I Environment

The data definition statements for DataManager SYSTEM, PROGRAM, and MODULE members acting on conventional files are described in the *ASG-Manager Products Dictionary/Repository User's Guide*. For the DL/I interface, a further clause, the PROCESSES clause, is included in the formats of those statements. This section describes that clause. For a full specification of the SYSTEM, PROGRAM, and MODULE data definition statements in a DL/I environment, therefore, this section must be read in conjunction with the relevant parts of the *ASG-Manager Products Dictionary/Repository User's Guide*.

The PROCESSES clause is available also in some other DataManager interface facilities. Its purpose is to specify an application's processing of its data within a specific environment. The clause has a number of alternative environment-dependent formats. In a DL/I environment, the PROCESSES clause defines an application's view of the DL/I databases accessed by the application.

The PROCESSES clause specifies the details of these DL/I features that an application SYSTEM, PROGRAM, or MODULE may utilize:

- The PROGRAM-COMMUNICATION-BLOCK or PCB members accessed by the application
- Segment-search-arguments

The PCB members named in the PROCESSES clause are used by the Source Language Generation facility when producing Program Specification Block (PSB) control statements (see "Application View" on page 18, "DataManager Data Definition Statements for DL/I Program Communication Blocks" on page 82, and "Generating DL/I PSB Control Statements" on page 137).

The segment search argument details may be used by the Source Language Generation facility when producing Database Description (DBD) control statements (see "Application View" on page 18 and "Generating DL/I DBD Control Statements" on page 131).

Specification of the PROCESSES Clause

Syntax

```
PROCESSES { DL/I }
          { DL/1 }
          { DLI  }
          { DL1  }

{ CONTAINS pcb-name[,pcb-name]... [SSAS clause] }
{ SSAS clause                                   }
```

where:

pcb-name is the name of a Program Communication Block (PCB) member.

SSAS *clause* is a clause having this format:

```
{ SEGMENT-SEARCH-ARGUMENTS }
{ SSAS                     }
 SEGMENT segment-name USED-IN clause
                     [USED-IN clause]...
 [SEGMENT segment-name USED-IN clause
                      [USED-IN clause]...]...
```

where *segment-name* is the name of a segment member.

USED-IN clause is a clause having this format:

```
USED-IN ssa-name
[COMMAND-CODES [ { FIRST-OCCURRENCE } ]
                 { LAST-OCCURRENCE  }
               [[,] { IO-MOVE     }
                    { NOREPLACE   }
                    { NO-REPLACE  }
               [[,] ENQUEUE 'class']
               [[,] NULL [poisitions]]]
[QUALIFIED-ON search-field operator]
```

where:

ssa-name is the segment-search-argument name, being a name that is valid in the programming language (COBOL, PL/I, or Assembler) relevant to the member that contains the PROCESSES clause in which this USED-IN clause appears.

class is an alphabetic character in the range A to J.

positions is an unsigned integer.

search-field is a name as stated in remark 16.

operator is of this format:

```
{ EQ  }
{ =   }
{ NE  }
{ GT  }
{ >   }
{ GE  }
{ LT  }
{ <   }
{ LE  }
```

where:

EQ or = means equal to.

NE means not equal to.

GT or > means greater than.

GE means greater than or equal to.

LT or < means less than.

LE means less than or equal to.

Remarks

1. The keyword DL/I (or one of its permitted variants) or IMS must immediately follow the PROCESSES keyword to indicate that a DL/I application view is being defined. The keyword IMS is synonymous with DL/I and its variants. It is included in the syntax for compatibility with the OS version of the interface.

2. If the CONTAINS subordinate clause is present, it must immediately follow the DL/I or IMS keyword.

3. If the Source Language Generation facility is to be used to produce Database Description (DBD) control statements for the database to which the segment belongs, at which time it is to generate the segment's search fields only (as opposed to generating all of the fields contained by the segment), a USED-IN clause must be specified to indicate which of the segment's fields are its search fields. (See "Application View" on page 13 and "Generating DL/I DBD Control Statements" on page 131.)

4. If the condition stated in remark 3 on page 93 does not apply, the USED-IN clause is omitted.

5. The USED-IN keyword must be followed immediately by ssa-name which must be unique in the PROCESSES clause.

6. The COMMAND-CODES clause is declared if the SEGMENT-SEARCH-ARGUMENT is to contain one or more command codes to provide functional variations applicable to either the call function or the segment qualification.

7. For retrieval calls, the command code FIRST-OCCURRENCE allows backing-up within a database record (starting with the first occurrence of this segment type under its parent, or with the first occurrence of this segment type after a position established earlier in the hierarchy) in order to satisfy the call.

8. For insert calls, the command code FIRST-OCCURRENCE is used for segments having a non-unique sequence field or no sequence field, and an insert rule of HERE, to specify that occurrences of this segment are to be inserted as the first segment on the twin chain.

9. For retrieval calls, the command code LAST-OCCURRENCE specifies that the last occurrence of this segment under its parent that satisfies the qualification statement is to be retrieved; or if there is no qualification statement, then the last occurrence of this segment under its parent is to be retrieved.

10. For insert calls, the command code LAST-OCCURRENCE is used for segments having a non-unique sequence field or no sequence field and an insert rule of HERE, to specify that occurrences of this segment are to be inserted as the last segment on the twin chain.

11. The IO-MOVE command code is valid only for path calls: in the relevant PCB member, PATH must be included in the segment's *processing-options-2*, or, if these are omitted, in the STRUCTURE clause's processing-options-l. For retrieval calls, the command code specifies that this segment is to be moved to the application program's input/output area. For insert calls, it designates the first segment that is to be inserted from the input/output area.

12. The NOREPLACE or NO-REPLACE command code specifies that for a replace call following a path retrieval call, this segment will not have been changed, and is therefore not to be replaced.

13. The command code ENQUEUE class specifies that this segment is to be enqueued for single update, where class is the class identifier used on the dequeue call to dequeue all resources enqueued by the user with that class.

14. The command code NULL [positions] enables a fixed number of bytes to be set aside for command codes, which may be set on or off by the application. The number of null bytes to be generated is specified by positions. If positions is omitted, one byte is assumed.

15. The QUALIFIED-ON clause defines information that DL/I uses to test the value of this segment's key or data fields within the database, to determine whether the segment meets the user's specifications.

16. The QUALIFIED-ON keyword, if present, must be followed immediately by search-field, which can identify a field of any of these types:

    - A GROUP or ITEM member that is contained directly or indirectly by this segment; including:

      — For a logical child segment, the destination parent's concatenated key

      — For a logical segment or a logical concatenated segment, the physical segment(s) represented by this segment

      If a member is indirectly contained by the segment, and is defined as an array in the data definition of its containing group, it must not be specified as search-field.

    - A field specified as sequence-key-name or concatenated-key-name in the data definition of:

      — This segment

      — The physical segment(s) represented by this segment, if this segment is a logical segment or a logical concatenated segment

    (See further in remark 17 on page 95.)

    - If this segment is an index target segment or a logical segment representing an index target segment, and is not a logical concatenated segment or a dependant of a logical concatenated segment, then a field defined as an index-search-field-name in the data definition of a related index pointer segment.

    - If this segment is an index pointer segment, the field defined as key-name in this segment's data definition. In this case, the field specified by search-field includes any susbequence fields specified in the segment's data definition.

17. If search-field is a sequence key field in the data definition of a virtual logical child segment, then the field includes all sequence key fields that follow it in that data definition.

18. The operator specifies the manner in which the contents of the search-field are to be tested against the comparative value.

19. When the member containing the PROCESSES clause is encoded, if any member whose name appears in that member's data definition has no data entries record, DataManager creates a dummy data entries record for the latter member, in accordance with these rules:

    - If the name appears in a CONTAINS clause that immediately follows PROCESSES IMS or PROCESSES DL/I (or a variant), DataManager creates a dummy PCB member. (This member is described in the informatory message as a DUMMY STRUCTURE PCB, for compatibility with the OS version of the interface, in which other kinds of PCB members can also exist.)
    - If the name appears in a CONTAINS clause that does not immediately follow PROCESSES IMS or PROCESSES DL/I (or a variant), DataManager creates a dummy module member.
    - If the name appears anywhere in the QUALIFIED-ON clause, DataManager generates a dummy item member.
    - If the name appears in any other clause, the dummy is created as defined in the specification of the SYSTEM, PROGRAM, or MODULE member in the *ASG-Manager Products Dictionary/Repository User's Guide*.

Examples

Refer to the example in "Application View" on page 13. This example shows a PROCESSES clause declaring two PCB members, each for a different database, and the segment search arguments required for that application.

For the first segment, SKILL, there is a USED-IN clause that defines a COMMAND-CODE and a QUALIFIED-ON clause for a search field. For the segment EXPR there is a USED-IN clause which defines no COMMAND-CODE but does have a QUALIFIED-ON clause for a search field.

For the third segment, NAMEID, there is again a USED-IN clause that defines a COMMAND-CODE and a QUALIFIED-ON clause for a search field. The segment CITY has a USED-IN clause specified, but has no COMMAND-CODE or QUALIFIED-ON clause.

This example shows a PROCESSES clause for an application requiring one PCB member, SKILLINV-PCB (see the example at the end of "DataManager Data Definition Statements for DL/I Program Communication Blocks" on page 82):

```
PROCESSES DL/I
CONTAIMS SKILLINV-PCB
SEGMENT-SEARCH-ARGUMENTS
    SEGMENT SKILMAST USED-IN SKILMAST-SSA
            QUALIFIED-ON SKILLCODE EQ
    SEGMENT SKILLNAME USED-IN SKILLNAM-SSA
            QUALIFIED-ON SURNAME EQ
    SEGMENT EXPRMAST USED-IN EXPRMAST-SSA
            COMMAND-CODE IO-MOVE
    SEGMENT EDCUMAST USED-IN EDCUMAST-SSA
```

Segment-search arguments are specified for 4 segments. The segments SKILMAST and SKILLNAME each have a USED-IN clause defining a QUALIFIED-ON clause for a search field. For the segment EXPRMAST, the USED-IN clause defines the COMMAND-CODE IO-MOVE to indicate that this segment is the first in a path of segments to be inserted. No COMMAND-CODE or QUALIFIED-ON clauses are specified for the segment EDUCMAST.

4 Extensions to DataManager Commands for DL/I Databases

Introduction

DataManager provides powerful facilities for documenting, interrogating, and processing the data definitions of the various types of DL/I databases and their components. These facilities are provided by means of:

- Additional member-type keywords in those commands that permit member-type selection (see "DL/I Member-type Keywords" on page 99)
- Additional condition keywords in the WHICH and WHAT commands (see "Condition Keywords for Which and What Commands" on page 102)

DL/I Member-type Keywords

The syntax of these DataManager commands (defined in the *ASG-Manager Products Dictionary/Repository User's Guide*):

- BULK ENCODE
- BULK PRINT
- BULK REPORT
- GLOSSARY
- LIST
- PERFORM
- WHICH

includes a number of member-type selection keywords that enable the processing to be confined to members of the selected type or types.

The member-type selection keywords include the keyword DATABASES. This keyword selects all members at the database level of the member-type hierarchy; if more than one Data Base Management System interface is included in the implementation of DataManager, then database members defined under any of the implemented interfaces are selected.

If the DL/I Interface is included in the implementation, additional keywords are made available to permit the selection to be confined to:

- All DL/I databases
- A specific category or specific categories of DL/I databases
- All DL/I segments
- A specific category or specific categories of segments
- Any of the internal member types described in "Special DataManager Member Types" on page 17 (except for BULK ENCODE and BULK PRINT)

These are the additional member-type selection keywords:

- DL/I-DATABASES
- DL/1-DATABASES
- DLI-DATABASES
- DL1-DATABASES
- GSAM-DATABASES
- HSAM-DATABASES
- SHSAM-DATABASES
- HISAM-DATABASES
- SHISAM-DATABASES
- HDAM-DATABASES
- HIDAM-DATABASES
- PHYSICAL-DATABASES
- LOGICAL-DATABASES
- SECONDARY-INDEX-DATABASES
- SEGMENTS
- PHYSICAL-SEGMENTS
- LOGICAL-SEGMENTS
- INDEX-POINTER-SEGMENTS
- PROGRAM-COMMUNICATION-BLOCKS
- PCBS
- SEQUENCE-KEYS
- DL/I-DATASETS
- DL/1-DATASETS
- DLI-DATASETS
- DL1-DATASETS
- INDEX-SEARCH-FIELDS
- SYSTEM-RELATED-FIELDS
- CONCATENATED-KEYS
- CONCATENATED-KEY-NAMES

These are not relevant for BULK ENCODE or BULK PRINT because members of these types have no source records.

These keywords are also available in the Controller's commands to save the contents of a data dictionary and to analyze a data dictionary's disk space usage. (These are documented in the *ASG-Manager Products Controller's Manual*.)

It is thus possible to obtain complete documentation of DL/I databases at the database or at any component level; to interrogate on database type and on any component type; and to select by database type or component type for manipulation by BULK ENCODE or through PERFORM commands.

Condition Keywords for Which and What Commands

Introduction

The WHICH command enables the user to interrogate the data dictionary as to which members, of selected types (see "DL/I Member-type Keywords" on page 99), satisfy selected conditions. Among the conditions that can be stated are that the members named in the response should USE a member named in the command, or that they should CONSTITUTE the member named in the command. These conditions can be restricted by a VIA clause, or by alternative verb keywords, to references to or from other members via a particular clause of a data definition. Similar conditions can be stated in the WHAT command, but without the restriction of the interrogation to selected categories of members.

The DL/I Interface provides further keywords for the condition clause.

The tables in the following sections give information on these keywords:

- The Member Type Interrogation table in "Member Type Interrogations" on page 106 explains which VIA keywords are appropriate for use with a particular DL/I member type, to interrogate various aspects of its definition.
- The Interrogation Syntax table in "Interrogation Syntax" on page 112 lists in alphabetical order the keywords that can be used in a VIA clause, together with the member types with which they can be used, and the responses that will be obtained.
- The Alternative Verb Keyword table in "Alternative Verb Keywords" on page 130 offers alternative verb keywords that can be used instead of same USES and CONSTITUTES constructions.

The Interrogation Syntax and Alternative Verb Keyword tables together give the possible values for the selection, member-type, alternative-verb-keyword, and via-keyword variables in a WHICH command of the form:

```
WHICH selection { { USES        } member-name VIA via-keyword }  ;
                  { CONSTITUTES }                             }
                  { alternative-verb-keyword member-name      }
```

For example, to find out which process members use a particular segment in the segment search argument, the VIA keyword SSAS is used. The entry for SSAS in the Interrogation Syntax table shows that the format of the required command would be:

```
WHICH { MODULES  }   USE { index-pointer-segment-name }   VIA SSAS;
      { PROGRAMS }       { logical-segment-name       }
      { SYSTEMS  }       { physical-segment-name      }
```

There is no alternative verb keyword available for this interrogation.

The member types listed for selection and member-name, the alternative verb keywords, and the keywords for use in the VIA clause are additional to those available for the generalized version of the WHICH command. The exceptions to this are the BOUND, CONTAINS, IF, and NAME keywords, and the alternative verb keyword CONTAINS; these are included in the tables to demonstrate their use with DL/I-specific member types.

If any of the keywords are also available for interrogating a DataManager definition of another Data Base Management System, and the user's implementation of DataManager includes an interface to that system, responses to interrogations can also include members that are defined for that other Database Management System.

Throughout the following sections, any of the alternative forms DL/1-DATABASES, DLI-DATABASES, and DL1-DATABASES are accepted for the keyword DL/I-DATABASES. Similarly, the alternative forms DL/1-DATASETS, DLI-DATASETS, and DL1-DATASETS are accepted for the keyword DL/I-DATASETS.

Examples

The keywords for use in the VIA clause allow every clause of a member definition to be interrogated. The examples that follow show how the keywords can be used to interrogate the DataManager definitions of some important DL/I concepts.

Generated Fields Interrogation

The GENERATES clause of physical segment or index pointer segment data definitions can be interrogated using the keyword GENERATES in the VIA clause, or by using the alternative verb keywords GENERATES or GENERATED-BY. For example, these commands could be used to obtain a list of all the fields that are directly specified in the GENERATES clause of the segments residing in a particular database:

```
KEEP WHICH PHYSICAL-SEGMENTS DIRECTLY CONSTITUTE
     physical-database-name;
PERFORM "ALSO KEEP WHICH ITEMS, GROUPS, SEQUENCE-KEYS"
        "DIRECTLY CONSTITUTE * VIA GENERATES;"
        KEPT-DATA CLEAR-KEPT-DATA;
LIST KEPT-DATA ALPHABETICALLY;
```

If an alternative verb keyword were used, the PERFORM command might read:

```
PERFORM "ALSO KEEP WHICH ITEMS, GROUPS, SEQUENCE-KEYS"
        "DIRECTLY GENERATED-BY *;"
        KEPT-DATA CLEAR-KEPT-DATA;
```

Hierarchical Path Interrogation

Hierarchical path interrogation is performed by using the keywords PARENT or FATHER in the VIA clause, or by using the alternative verb keywords FATHERS or FATHERED-BY. For example, using the example illustrated by the syntax in "Specification of the Data Definition Statement for a HISAM Type DL/I Database" on page 62, the response to the command:

```
WHICH SEGMENTS USE JOB-TITLE VIA PARENT;
```

would consist of the segments DEPARTMENT, EMPLOYEE-NUMBER, and JOB-STATUS, which are direct or indirect parents of segment JOB-TITLE.

The command:

```
WHICH SEGMENTS DIRECTLY CONSTITUTE DIRECTORY JOB-STATUS VIA FATHER;
```

would cause the segments SALARY and JOB-TITLE, which are direct dependents of segment JOB-STATUS, to be output.

Using the alternative verb keywords, the first interrogation could be:

```
WHICH SEGMENTS FATHER JOB-TITLE;
```

and the second:

```
WHICH SEGMENTS DIRECTLY FATHERED-BY JOB-STATUS
```

Logical Relationship Interrogation

The TO keyword interrogates the relationship between logical child segments and their destination parent segments, as specified in the RELATED-AS clause of the logical child segment definition. For example, the command:

```
WHICH PHYSICAL-SEGMENTS DIRECTLY CONSTITUTE ASY-LINE VIA TO;
```

when used with the syntax in "Specification of the Data Definition Statement for a Segment that Resides in a Physical Database" on page 24 would respond with the segment PRODPART.

Secondary Index Relationship Interrogation

The relationships between index pointer segments and index target segments can be ascertained by using the TARGET keyword.

Using the example illustrated in Figure 3 on page 9, the command:

```
WHICH INDEX-POINTER-SEGMENTS DIRECTLY USE NAMEID VIA TARGET;
```

would respond with the segment COLORSEG.

The relationship between index pointer segments and source segments can be interrogated using the SOURCE keyword. Thus, the interrogation:

```
WHICH INDEX-POINTER-SEGMENTS DIRECTLY USE AUTOMBLE VIA SOURCE;
```

would respond with the segment COLORSEG.

Segment Search Argument Interrogation

The SSAS keyword can be used to find out which segments are used by a particular process member via its SEGMENT-SEARCH-ARGUMENTS clause. This could be achieved by the command:

```
WHICH SEGMENTS DIRECTLY CONSTITUTE process-member-name VIA SSAS;
```

Using the PROCESSES clause example in "Specification of the PROCESSES Clause" on page 92, the response would consist of the segments SKILMAST, SKILLNAM, EXPRMAST, and EDUCMAST.

The QUALIFIED-ON keyword is used to find out the relationships between process member types and the fields specified in the QUALIFIED-ON subordinate clause of the SEGMENT-SEARCH-ARGUMENTS clause. Again, using the example in "Specification of the PROCESSES Clause" on page 92, the response to the command:

```
WHICH MEMBERS DIRECTLY CONSTITUTE process-member-name VIA QUALIFIED-ON;
```

would include the members SKLLCODE, SURNAME, and INITIAL.

Sensitive Segment And Sensitive Field Interrogation

The relationships between structure type Program Communication Blocks (PCBS) and the sensitive segments and fields specified in them can be interrogated using the SEGMENT or SENSITIVE-FIELDS keywords respectively. For example, using the second example of a structure type PCB in "DataManager Data Definition Statements for DL/I Program Communication Blocks" on page 82, the command:

```
WHICH SEGMENTS DIRECTLY CONSTITUTE ASY-PACK-PCB VIA SEGMENT;
```

would respond with the segments ASY-LINE and ASY-PACK.

The command:

```
WHICH MEMBERS DIRECTLY CONSTITUTE ASY-PACK-PCB VIA SENSITIVE-FIELDS;
```

using the same example, would respond with the members PACK-NO, PROD-NO, and QTY-REQD.

Sequence Key Interrogation

The SEQUENCE-KEY clause can be used to interrogate the relationships between sequence key fields and segments in which they are specified.

These commands could be used to ascertain the sequence key fields of the logical database SKILLEMP illustrated in Figure 2 on page 6:

```
KEEP WHICH PHYSICAL-SEGMENTS CONTAINED-BY SKILLEMP;
PERFORM 'WHICH ITEMS, GROUPS, SEQUENCE-KEYS DIRECTLY'
        'CONSTITUTE * VIA SEQUENCE-KEY;'
        KEPT-DATA CLEAR-KEPT-DATA;
```

This pair of commands would respond with the members SURNAME, PAYRNUMB, SKLLCODE, QUALCODE, and EMPLOYEE-NO.

Variable Length Array Interrogation

The BOUND keyword can be used to interrogate the relationship between variable length arrays and physical segments. For example, the command:

```
WHICH PHYSICAL-SEGMENTS USE NUMBER-OF-LINES VIA BOUND;
```

would respond with the names of the physical segments whose CONTAINS clauses refer directly or indirectly to a variable length array, the number of occurrences of which is based on the value of the item NUMBER-OF-LINES.

Member Type Interrogations

The purpose of this table is to summarize, for each DL/I-specific member type, the VIA keywords that may be used to interrogate various clauses of the member definition.

In the first column of the table, the member types are listed in the order of databases, segments, Program Communication Blocks (PCBs), and process members. The second column lists the keywords that are available for interrogating clauses in members of a particular type. The third column explains, for each keyword, the relationship between the member type and the clause, or subordinate clause, of the member type data definition that the keyword interrogates.

Member Type Interrogation

Member Type	Keyword for Use in VIA Clause	Relationship Interrogated by Keyword
HDAM-DATABASES HIDAM-DATABASES	CONTAINS	Relationship between HDAM or HIDAM databases and segment members that are contained in the database.
	DL/I- DATASETS	Relationship between HDAM or HIDAM databases and the data file members specified in the DATASETS clause of the database data definition statement.

Member Type Interrogation

Member Type	Keyword for Use in VIA Clause	Relationship Interrogated by Keyword
	FATHERS / PARENTS	Hierarchical parent and child relationship between segments whose names are listed in the CONTAINS clause of the database data definition statement.
	RANDOMIZING MODULES / RANDOMIZING MODULES	Relationship between HDAM and HIDAM databases and module members specified in the RANDOMIZING-MODULES clause of the database data definition statement.
HISAM-DATABASES HSAM-DATABASES	CONTAINS	Relationship between HSAM or HISAM databases and segment members that are contained in the database.
	DL/I- DATASETS	Relationship between HSAM or HISAM databases and the data file members that constitute the database (that is, the data file members specified in the DATASETS clause of the database data definition statement).
	FATHERS / PARENTS	Hierarchical parent and child relationship between segments whose names are listed in the CONTAINS clause of the database data definition statement.
LOGICAL-DATABASES	CONTAINS	Relationship between LOGICAL databases and segment members contained directly or indirectly in the database.
	FATHERS / PARENTS	Hierarchical parent and child relationship between segments whose names are listed in the CONTAINS clause of the database data definition statement.
SECONDARY-INDEX-DATABASES	CONTAINS	Relationship between secondary index databases and the index pointer segment contained in the database.
	DL/I-DATASETS	Relationship between secondary index databases and the dataset members that constitute the database (that is, the datasets specified in the DATASETS clause of the database data definition statement).
INDEX-POINTER-SEGMENT	BOUND	Relationship between index pointer segments and variable length arrays specified directly or indirectly as groups or items in the CONTAINS clause of the segment data definition statement.

Member Type Interrogation

Member Type	Keyword for Use in VIA Clause	Relationship Interrogated by Keyword
	CONTAINS	Relationship between index pointer segments and group and/or item members specified in the CONTAINS clause of the segment data definition statement.
	DUPLICATE-DATA-FIELDS	Relationship between index pointer segments and items, groups and/or system related fields specified in the DUPLICATE-DATA-FIELDS clause of the index pointer segment data definition statement.
	GENERATES	Relationship between index pointer segments and members of any of the types that may be specified in the GENERATES clause of the index pointer segment data definition statement.
	IF	Relationship between index pointer segments and item and/or group members specified in IF subordinate clauses in the CONTAINS clause of the index pointer segment data definition statement.
	{ IN / OF }	Relationship between index pointer segments and members that may be specified in the IN/OF subordinate clause of the GENERATES clause of the index pointer segment data definition statement.
	MAINTENANCE-EXITS	Relationship between index pointer segments and module members specified in the MAINTENANCE-EXITS clause of the index pointer segment data definition statement.
	ON	Relationship between index pointer segments and the member specified in the ON subordinate clause of the RELATED-TO clause of the index pointer segment data definition statement [that is, the index search field (XDFLD)].
	SEARCH-KEY-FIELDS	Relationship between index pointer SEGMENT segments and group and/or item members specified as search key fields in the SEARCH-KEY-FIELDS clause of the segment data definition statement.
	SEQUENCE-KEYS	Relationship between index pointer segments and the member specified as the sequence key in the SEQUENCE-KEY clause of the segment data definition statement.

Member Type Interrogation

Member Type	Keyword for Use in VIA Clause	Relationship Interrogated by Keyword
	SOURCE	Relationship between index pointer segments and the index source segment specified in the SOURCE clause of the segment data definition statement.
	SUBSEQUENCE-FIELDS	Relationship between index pointer segments and items, groups, and/or system related fields specified in the SUBSEQUENCE-FIELDS clause of the segment data definition statement.
	TARGET	Relationship between index pointer segments and the index target segment specified in the RELATED-TO clause of the index pointer segment data definition statement.
LOGICAL-SEGMENT	CONTAINS	Relationship between logical segments and physical segments contained by the segment.
	{ DATABASES IN-DATABASES }	Relationship between logical segments and the physical database in which the physical segment contained by the logical segment resides (specified by the IN subordinate clause of the logical segment data definition statement).
PHYSICAL-SEGMENT	BOUND	Relationship between physical segments and variable length arrays specified directly or indirectly as groups or items in the CONTAINS clause of the segment data definition statement.
	CONCATENATED-KEY-CONSTITUENTS (source segments only)	Relationship between physical segments and fields specified in the CONCATENATED-KEY-FIELDS clause of the physical segment data definition statement. Only those fields specified before the AS CK*xxxxx* subordinate clause are included.
	CONCATENATED-KEY-FIELDS (source segments only)	Relationship between physical segments and system related fields specified in the AS CX*xxxxx* subordinate clause of the CONCATENATED-KEY-FIELDS clause in the segment data definition statement.
	CONCATENATED-KEY-NAMES (logical child segments only)	Relationship between physical segments and the member specified as a concatenated key name in the CONCATENATED-KEY-NAME clause of the segment data definition statement.
	CONTAINS	Relationship between physical segments and group and/or item members contained directly or indirectly in the segment.

Member Type Interrogation

Member Type	Keyword for Use in VIA Clause	Relationship Interrogated by Keyword
	EDIT-COMPRESSION-EXITS	Relationship between physical segments and module members specified in the EDIT-COMPRESSION-EXITS clause of the segment data definition statement.
	GENERATES	Relationship between physical segments and fields that can be specified in the GENERATES clause of the segment data definition.
	IF	Relationship between physical segments and item and/or group members specified in IF subordinate clauses of the CONTAINS clause of the segment data definition statement.
	{ IN / OF }	Relationship between physical segment and members of any of the types that are specified in the IN/OF subordinate clause of the GENERATES clause of the segment data definition statement.
	RENAMES (logical child segments only)	Relationship between logical child segments and items, groups and sequence key members specified in the RENAMES clause of the logical child segment data definition statement.
	SEQUENCE-KEY-CONSTITUENTS (logical child segments only)	Relationship between logical child segments and fields specified in the SEQUENCE-KEY clause of the segment data definition statement, when the AS subordinate clause of the SEQUENCE-KEY clause has also been specified. Only the entries preceding the AS subordinate clause are included in the response.
	SEQUENCE-KEY	Relationship between physical segments and the item or group member specified in the SEQUENCE-KEY clause of the segment data definition statement, when the AS subordinate clause has not been specified. Also, when the AS clause has been specified, the relationship between the physical segment and the sequence key name specified in the AS clause.
	TO (logical child segments only)	Relationship between logical child segments and the destination parent segment specified in the TO subordinate clause of the RELATED-AS clause of the physical segment data definition statement.
	WITH (paired logical child segments only)	Relationship between the logical child segment and the segment with which it is paired, as specified in the WITH subordinate clause of the segment data definition statement.

Member Type Interrogation

Member Type	Keyword for Use in VIA Clause	Relationship Interrogated by Keyword
PCB (GSAM type)	DATABASES IN-DATABASES	Relationship between PCBs and the GSAM database named in the PCB data definition statement.
	EXIT-ROUTINES	Relationships between PCB and module, program, or system members specified in the EXIT-ROUTINES of the PCB data definition statement.
	NAME	Relationship between PCBs and the PCB specified in the NAME clause of the PCB data definition statement.
	SECONDARY-SEQUENCE-ON	Relationship between PCBs and the index pointer segment specified in the ON subordinate clause of the SECONDARY-SEQUENCE clause of the PCB data definition statement.
	SEGMENT	Relationship between PCBs and sensitive segments specified in the SEGMENT clause of the PCB data definition statement.
	SENSITIVE-FIELDS	Relationship between PCB and fields specified as sensitive by means of the SENSITIVE-FIELDS clause of the PCB data definition statement.
MODULE PROGRAM SYSTEM	CONTAINS	Relationship between module, program or system members and the PCB members which they use; that is, the PCB members specified in the CONTAINS subordinate clause of the PROCESSES-clause of the module, program or system data definition statement.
	QUALIFIED-ON	Relationship between module, program or system members and the segment search fields specified in the QUALIFIED-ON subordinate clause of the PROCESSES clause of the module, program or system data definition statement.
	SSAS	Relationship between module, program or system members and the segment members specified in the SSAS subordinate clause of the PROCESSES clause of the module, program or system data definition statement.

Interrogation Syntax

This table provides the user with the information required to construct an interrogation of the form:

```
WHICH selection { { USES        } member-name VIA via-keyword } ;
                { { CONSTITUTES }                              }
                { alternative-verb-keyword member-name         }
```

The first column lists, in alphabetical order, the keywords that can be used in a VIA clause. The second column lists all the meaningful member types that could be specified for the selection variable in a USES interrogation. The third column lists the member types from which the member named in the member-name variable should be selected if a meaningful response is to be obtained. The fourth and fifth columns give, respectively, information similar to that in the second and third columns, except that the member types given are those that are meaningful in a CONSTITUTES interrogation.

The final column explains the response that will be obtained from either a USES or CONSTITUTES interrogation, and includes any notes concerning the use of the keyword. The responses detailed here are those that appear when the interrogation has been qualified by the keyword DIRECTLY. If DIRECTLY is not specified, both direct relationships and indirect relationships established by CONTAINS clauses are reported on.

In addition to the member types listed in the second and fourth columns, the general selection keywords MEMBERS, KEPT-DATA, and INDEX-NAMES may be used, although meaningful responses will be obtained only when these categories include members of the types listed in the second and fourth columns.

The values that may be supplied for the alternative-verb-keyword variable are described in "Alternative Verb Keywords" on page 130.

Interrogation Syntax

Keyword for use in the VIA clause	USES interrogations: Meaningful member-type selection keywords	USES interrogations: Meaningful member types for member-name	CONSTITUTES interrogations: Meaningful member-type selection keywords	CONSTITUTES interrogations: Meaningful member types for member-name	Explanation/Notes
BOUND	INDEX-POINTER-SEGMENTS PHYSICAL-SEGMENTS SEGMENTS	ITEM	ITEMS	INDEX-POINTER-SEGMENT PHYSICAL-SEGMENT	USES: Obtains the name of each index pointer segment and/or physical segment that contains a variable length array whose number of occurrences is specified by the value of the item named in the member-name part of the interrogation. CONSTITUTES: Obtains the name of each item that is used as an array bound for a variable length array contained in the index pointer segment or physical segment named in the member-name part of the interrogation.
CONCATENATED-KEY-CONSTITUENTS	PHYSICAL-SEGMENTS SEGMENTS	ITEM GROUP	ITEMS GROUPS	PHYSICAL-SEGMENT (source segments only)	This keyword applies only to physical segments that are defined as index source segments. USES: Obtains the name of each physical segment that specifies, in the CONCATENATED-KEY-FIELDS clause of its data definition, the group or item or sequence key named in the member-name part of the interrogation. The member named by member-name must appear in the part of the clause preceding the AS CK*xxxxx* subordinate clause. CONSTITUTES: Obtains the name of each item and/or group and/or sequence key that is specified in the CONCATENATED-KEY-FIELDS clause of the physical segment named in the member-name part of the interrogation. Only items, groups, and sequence keys specified prior to each AS CK*xxxxx* subordinate clause are included in the response.

	USES interrogations		CONSTITUTES interrogations		
Keyword for use in the VIA clause	**Meaningful member-type selection keywords**	**Meaningful member types for member-name**	**Meaningful member-type selection keywords**	**Meaningful member types for member-name**	**Explanation/Notes**
CONCATENATED-KEY- FIELDS	PHYSICAL-SEGMENTS SEGMENTS	SYSTEM-RELATED-FIELD	SYSTEM-RELATED-FIELDS	PHYSICAL-SEGMENT (source segments only)	This keyword only applies to physical segments that are defined as index source segments. USES: Obtains the name of the physical segment that specifies, in the AS CK*xxxxx* subordinate clause of its data definition, the system related field named in the member-name part of the interrogation. This field must be of the form CK*xxxxx* (see "Specification of the Data Definition Statement for a Segment that Resides in a Physical Database" on page 24). CONSTITUTES: Obtains the name of each system related field that is specified in the AS CK*xxxxx* subordinate clause of the data definition of the physical segment named in the member-name part of the interrogation.
CONCATENATED-KEY- NAMES	PHYSICAL-SEGMENTS SEGMENTS	CONCATENATED-KEY-NAME	CONCATENATED-KEY-NAME	PHYSICAL-SEGMENT	USES: Obtains the name of the physical segment or index-pointer-segment that specifies in the CONCATENATED-KEY-NAME clause of its data definition, the concatenated key name named in the member-name part of the interrogation. CONSTITUTES: Obtains the name of the concatenated key that is specified in the CONCATENATED-KEY-NAME clause of the data definition of the physical segment, or index pointer segment named in the member-name part of the interrogation.

Interrogation Syntax

Keyword for use in the VIA clause	USES interrogations: Meaningful member-type selection keywords	USES interrogations: Meaningful member types for member-name	CONSTITUTES interrogations: Meaningful member-type selection keywords	CONSTITUTES interrogations: Meaningful member types for member-name	Explanation/Notes
CONTAINS	HDAM-DATABASE HIDAM-DATABASES HISAM-DATABASES HSAM-DATABASES PHYSICAL-DATABASES SHISAM-DATABASES SHSAM-DATABASES DL/I-DATABASES DATABASES	PHYSICAL-SEGMENTS	PHYSICAL-SEGMENTS SEGMENTS	HDAM-DATABASE HIDAM-DATABASE HISAM-DATABASE HSAM-DATABASE SHISAM-DATABASE SHSAM-DATABASE	USES: Obtains the name of each database that specifies, in the CONTAINS clause of its data definition, the physical segment named in the member-name part of the interrogation. CONSTITUTES: Obtains the name of each physical segment, specified in the CONTAINS clause of the data definition of the database named in the member-name part of the interrogation. Instead of using the CONTAINS keyword in the VIA clause, the verb keywords CONTAINS and CONTAINED-BY could be used (see "Interrogation Syntax" on page 112).
CONTAINS	LOGICAL-DATABASES DL/I-DATABASES DATABASES	LOGICAL-SEGMENT PHYSICAL-SEGMENT	LOGICAL-SEGMENT PHYSICAL-SEGMENTS SEGMENTS	LOGICAL-DATABASE	USES: Obtains the name of each logical database that specifies, in the CONTAINS clause of its data definition, the logical or physical segment named in the member-name part of the interrogation. CONSTITUTES: Obtains the name of each logical and/or physical segment, that is specified in the CONTAINS clause of the data definition of the database named in the member-name part of the interrogation. Instead of using the CONTAINS keyword in the VIA clause, the verb keywords CONTAINS and CONTAINED-BY could be used (see "Interrogation Syntax" on page 112).

	USES interrogations		CONSTITUTES interrogations		
Keyword for use in the VIA clause	**Meaningful member-type selection keywords**	**Meaningful member types for member-name**	**Meaningful member-type selection keywords**	**Meaningful member types for member-name**	**Explanation/Notes**
CONTAINS	SECONDARY-INDEX-DATABASES DL/I-DATABASES DATABASES	INDEX-POINTER-SEGMENT	INDEX-POINTER-SEGMENTS SEGMENTS	SECONDARY-INDEX-DATABASE DATABASE	USES: Obtains the name of the secondary index database that specifies, in the CONTAINS clause of its data definition, the index pointer segment named in the member-name part of the interrogation. CONSTITUTES: Obtains the name of each index pointer segment, that is specified in the CONTAINS clause of the data definition of the secondary index database named in the member-name part of the interrogation. Instead of using the CONTAINS keyword in the VIA clause, the verb keywords CONTAINS and CONTAINED-BY could be used (see "Interrogation Syntax" on page 112).
CONTAINS	INDEX-POINTER-SEGMENTS PHYSICAL-SEGMENTS SEGMENTS	ITEM GROUP	ITEMS GROUPS	INDEX-POINTER-SEGMENT PHYSICAL-SEGMENT	USES: Obtains the name of each physical segment and/or index pointer segment that specifies, in the CONTAINS clause of its data definition, the group or item named in the member-name part of the interrogation. CONSTITUTES: Obtains the name of each group and/or item, that is specified in the CONTAINS clause of the data definition of the index pointer segment, named in the member-name part of the interrogation. Instead of using the CONTAINS keyword in the VIA clause, the verb keywords CONTAINS and CONTAINED-BY could be used (see "Interrogation Syntax" on page 112).

Interrogation Syntax

Keyword for use in the VIA clause	USES interrogations: Meaningful member-type selection keywords	USES interrogations: Meaningful member types for member-name	CONSTITUTES interrogations: Meaningful member-type selection keywords	CONSTITUTES interrogations: Meaningful member types for member-name	Explanation/Notes
CONTAINS	LOGICAL-SEGMENTS SEGMENTS	PHYSICAL-SEGMENTS	PHYSICAL-SEGMENTS SEGMENTS	LOGICAL-SEGMENT	USES: Obtains the name of each logical segment that specifies, in the CONTAINS clause of its data definition, the physical segment named in the member-name part of the interrogation. CONSTITUTES: Obtains the name of each physical segment, that is specified in the CONTAINS clause of the data definition of the logical segment named in the member-name part of the interrogation. Instead of using the CONTAINS keyword in the VIA clause, the verb keywords CONTAINS and CONTAINED-BY could be used (see "Interrogation Syntax" on page 112).
CONTAINS	MODULES PROGRAMS SYSTEMS	PCB PROGRAM-COMMUNICATION-BLOCKS	PCBS PROGRAM-COMMUNICATION-BLOCKS	MODULE PROGRAM SYSTEM	USES: Obtains the name of each module and/or program and/or system that specifies, in the CONTAINS subordinate clause of the PROCESSES clause of its data definition, the Program Communication Block (PCB) named in the member-name part of the interrogation. CONSTITUTES: Obtains the name of each Program Communication Block (PCB), that is specified in the CONTAINS subordinate clause of the PROCESSES clause of the data definition of the module, program or system named in the member-name part of the interrogation. Instead of using the CONTAINS keyword in the VIA clause, the verb keywords CONTAINS and CONTAINED-BY could be used (see "Interrogation Syntax" on page 112).

	USES interrogations		CONSTITUTES interrogations		
Keyword for use in the VIA clause	**Meaningful member-type selection keywords**	**Meaningful member types for member-name**	**Meaningful member-type selection keywords**	**Meaningful member types for member-name**	**Explanation/Notes**
DATABASES	LOGICAL-SEGMENTS SEGMENTS	HDAM-DATABASE HIDAM-DATABASE	HDAM-DATABASES HIDAM-DATABASES PHYSICAL-DATABASES DL/I-DATABASES DATABASES	LOGICAL-SEGMENT	USES: Obtains the name of each logical segment that specifies, in the IN clause of its data definition, the HDAM or HIDAM or HISAM database named in the member-name part of the interrogation. CONSTITUTES: Obtains the name of the database, that is specified in the IN clause of the data definition of the logical segment named in the member-name part of the interrogation.
DL/I-DATASETS	HDAM-DATABASE HIDAM-DATABASES HSAM-DATABASES HISAM-DATABASES SHSAM-DATABASES SHISAM-DATABASES PHYSICAL-DATABASES SECONDARY-INDEX- DATABASES DL/I-DATABASES DATABASES	DL/I-DATASET	DL/I-DATASETS	HDAM-DATABASE HIDAM-DATABASE HISAM-DATABASE SHSAM-DATABASE SHISAM-DATABASE SECONDARY-INDEX-DATABASE	USER: Obtains the name of the database that specifies, in the DATASETS clause of its data definition, the DL/I-DATASET member named in the member-name part of the interrogation. CONSTITUTES: Obtains the name of each DL/I-DATASET, that is specified in the DATASETS clause of the data definition statement of the database named in the member-name part of the interrogation.

Interrogation Syntax

Keyword for use in the VIA clause	USES interrogations: Meaningful member-type selection keywords	USES interrogations: Meaningful member types for member-name	CONSTITUTES interrogations: Meaningful member-type selection keywords	CONSTITUTES interrogations: Meaningful member types for member-name	Explanation/Notes
DUPLICATE-DATA-FIELDS	INDEX-POINTER-SEGMENTS SEGMENTS	ITEM GROUP SYSTEM-RELATED-FIELD	ITEMS GROUPS SYSTEM-RELATED FIELDS	INDEX-POINTER-SEGMENT	USES: Obtains the name of each index pointer segment that specifies, in the DUPLICATE-DATA-FIELDS clause of its data definition, the item, group, or system related field named in the member-name part of the interrogation. In a USES interrogation, any system-related field specified for member-name must be of the form CK*xxxxx* (see "Specification of the Data Definition Statement for a SEGMENT that Resides in a Secondary Index Database" on page 46). CONSTITUTES: Obtains the name of each item and/or group and/or system related field, that is specified in the DUPLICATE-DATA-FIELDS clause of the data definition of the index pointer segment named in the member-name part of the interrogation.
EDIT-COMPRESSION-EXITS	PHYSICAL-SEGMENTS SEGMENTS	MODULE	MODULES	PHYSICAL-SEGMENT	USES: Obtains the name of each physical segment that specifies, in the EDIT-COMPRESSION-EXIT clause of its data definition, the module named in the member-name part of the interrogation. CONSTITUTES: Obtains the name of the module, that is specified in the EDIT-COMPRESSION-EXIT clause of the data definition of the physical segment named in the member-name part of the interrogation.

	USES interrogations		CONSTITUTES interrogations		
Keyword for use in the VIA clause	**Meaningful member-type selection keywords**	**Meaningful member types for member-name**	**Meaningful member-type selection keywords**	**Meaningful member types for member-name**	**Explanation/Notes**
EXIX-ROUTINES	PCBS PROGRAM-COMMUNICATION-BLOCKS	MODULE PROGRAM SYSTEM	MODULES PROGRAMS SYSTEMS	PCB PROGRAM-COMMUNICATION-BLOCK	USES: Obtains the name of each PCB that specifies in the EXIT-ROUTINES clause of its data definition statement the module, program, or system named in the member-name part of the interrogation. CONSTITUTES: Obtains the name of each module and/or program and/or system that is specified in the EXIT-ROUTINES clause of the data definition of the PCB named in the member-name part of the interrogation.
FATHERS	LOGICAL-SEGMENTS PHYSICAL-SEGMENTS SEGMENTS	LOGICAL-SEGMENT PHYSICAL-SEGMENT	LOGICAL-SEGMENTS PHYSICAL-SEGMENTS SEGMENTS	LOGICAL-SEGMENT PHYSICAL-SEGMENT	USES: Obtains the name of each segment that is a parent of the segment specified in the member-name part of the interrogation. CONSTITUTES: Obtain the name of each segment that is a dependent of the segment specified in the member-name part of the interrogation. If a segment that resides in more than one database is interrogated using this keyword, DataManager processes only the first database that it obtains from the segment's used-by table. The keyword FATHERS is synonymous with the keyword PARENTS. Instead of using the keyword FATHERS (or PARENTS) in the VIA clause, the verb keywords FATHERS and FATHERED-BY can be used.

Interrogation Syntax

Keyword for use in the VIA clause	USES interrogations: Meaningful member-type selection keywords	USES interrogations: Meaningful member types for member-name	CONSTITUTES interrogations: Meaningful member-type selection keywords	CONSTITUTES interrogations: Meaningful member types for member-name	Explanation/Notes
GENERATES	INDEX-POINTER-SEGMENTS SEGMENTS	ITEM GROUP INDEX-SEARCH-FIELD SYSTEM-RELATED-FIELD SEQUENCE-KEY	ITEMS GROUPS INDEX-SEARCH-FIELDS SYSTEM-RELATED-FIELDS SEQUENCE-KEYS	INDEX-POINTER-SEGMENT	USES: Obtains the name of each index pointer segment that specifies, in the GENERATES clause of its data definition, the member named in the member-name part of the interrogation. CONSTITUTES: Obtains the names of any of the members, that are specified in the GENERATES clause of the data definition named in the member-name part of the interrogation. Instead of using the keyword GENERATES in the VIA clause, the verb keywords GENERATES and GENERATED-BY can be used (see "Interrogation Syntax" on page 112).
GENERATES	PHYSICAL-SEGMENTS SEGMENTS	ITEM GROUP SEQUENCE-KEY CONCATENATED-KEY	ITEMS GROUPS SEQUENCE-KEYS CONCATENATED-KEYS CONCATENATED-KEY-NAMES	PHYSICAL-SEGMENT	USES: Obtains the name of each index physical segment that specifies, in the GENERATES clause of its data definition, the item, group, sequence key or concatenated key named in the member-name part of the interrogation. CONSTITUTES: Obtains the names of any of the members, that are specified in the GENERATES clause of the data definition of the physical segment named in the member-name part of the interrogation. Instead of using the keyword GENERATES in the VIA clause, the verb keywords GENERATES and GENERATED-BY can be used (see "Interrogation Syntax" on page 112).

Interrogation Syntax

	USES interrogations		CONSTITUTES interrogations		
Keyword for use in the VIA clause	**Meaningful member-type selection keywords**	**Meaningful member types for member-name**	**Meaningful member-type selection keywords**	**Meaningful member types for member-name**	**Explanation/Notes**
IF	INDEX-POINTER-SEGMENTS PHYSICAL-SEGMENTS SEGMENTS	ITEM GROUP	ITEMS GROUPS	INDEX-POINTER-SEGMENT PHYSICAL-SEGMENT	USES: Obtains the name of each GSAM database and/or index pointer and/or physical segment that specifies, in the IF subordinate clause of the CONTAINS clause of its data definition, the item or group named in the member-name part of the interrogation. CONSTITUTES: Obtains the name of each item and/or group, that is specified in the IF subordinate clause of the CONTAINS clause of the data definition of the GSAM database, index pointer segment or physical segment named in the member-name part of the interrogation.
IN	INDEX-POINTER-SEGMENTS SEGMENTS	GROUP SYSTEM-RELATED-FIELD INDEX-SEARCH-FIELD SEQUENCE-KEY	GROUPS SYSTEM-RELATED-FIELDS INDEX-SEARCH-FIELDS SEQUENCE-KEYS	INDEX-POINTER-SEGMENT	USES: Obtains the name of each index pointer segment that specifies, in the IN/OF subordinate clause of the GENERATES clause of its data definition, the member named in the member-name part of the interrogation. CONSTITUTES: Obtains the name of any of the members, that is specified in the IN/OF subordinate clause of the data definition of the index pointer segment named in the member-name part of the interrogation.
IN-DATABASES					See DATABASES.
MAINTENANCE-EXITS	INDEX-POINTER-SEGMENTS SEGMENTS	MODULE	MODULES	INDEX-POINTER-SEGMENT	USES: Obtains the name of each index pointer segment that specifies, in the MAINTENANCE-EXIT clause of its data definition, the module named in the member-name part of the interrogation. CONSTITUTES: Obtains the name of the module, that is specified in the MAINTENANCE-EXIT clause of the data definition of the index pointer segment named in the member-name part of the interrogation.

Interrogation Syntax

Keyword for use in the VIA clause	USES interrogations: Meaningful member-type selection keywords	USES interrogations: Meaningful member types for member-name	CONSTITUTES interrogations: Meaningful member-type selection keywords	CONSTITUTES interrogations: Meaningful member types for member-name	Explanation/Notes
NAME	PCBS PROGRAM-COMMUNICATION-BLOCKS	PCB PROGRAM-COMMUNICATION-BLOCK	PCBS PROGRAM-COMMUNICATION-BLOCKS	PCB PROGRAM-COMMUNICATION-BLOCK	USES: Obtains the name of each Program Communication Block (PCB) that specifies, in the NAME clause of its data definition, the PCB named in the member-name part of the interrogation. CONSTITUTES: Obtains the name of the PCB, that is specified in the NAME clause of the data definition of the PCB named in the member-name part of the interrogation.
OF					See IN.
ON	INDEX-POINTER-SEGMENTS SEGMENTS	INDEX-SEARCH-FIELD	INDEX-SEARCH-FIELDS	INDEX-POINTER-SEGMENTS	USES: Obtains the name of the index pointer segment that specifies, in the ON clause of its data definition, the index search field named in the member-name part of the interrogation. CONSTITUTES: Obtains the name of the index search field, that is specified in the ON clause of the data definition of the index pointer segment named in the member- name part of the interrogation.
PARENTS					See FATHERS.
QUALIFIED-ON	MODULES PROGRAMS SYSTEMS	ITEM GROUP INDEX-SEARCH-FIELD SEQUENCE-KEY CONCATENATED-KEY	ITEMS GROUPS INDEX-SEARCH-FIELDS SEQUENCE-KEYS CONCATENATED-KEYS CONCATENATED-KEY- NAMES	MODULE PROGRAM SYSTEM	USES: Obtains the name of each module and/or program and/or system that specifies, in the QUALIFIED-ON subordinate clause of the PROCESSES clause of its data definition, the field named in the member-name part of the interrogation. CONSTITUTES: Obtains the names of members, that are specified in the QUALIFIED-ON subordinate clause of the PROCESSES clause of the data definition of the module, program, or system named in the member-name part of the interrogation.

	USES interrogations		CONSTITUTES interrogations		
Keyword for use in the VIA clause	**Meaningful member-type selection keywords**	**Meaningful member types for member-name**	**Meaningful member-type selection keywords**	**Meaningful member types for member-name**	**Explanation/Notes**
RANDOMIZING-MODULES RANDOMIZING-MODULES	HDAM-DATABASES PHYSICAL-DATABASES DL/I-DATABASES DATABASES	MODULE	MODULES	HDAM-DATABASES	USES: Obtains the name of each HDAM database that specifies, in the RANDOMISING-MODULES clause of its data definition, the module named in the member-name part of the interrogation. CONSTITUTES: Obtains the name of the module that is specified in the RANDOMISING-MODULES clause of its data definition of the HDAM database named in the member-name part of the interrogation.
RENAMES	PHYSICAL-SEGMENTS SEGMENTS	ITEM GROUP	ITEMS GROUPS	PHYSICAL-SEGMENT (logical child segments only)	This keyword is only applicable to logical child segments. USES: Obtains the name of each logical child segment that specifies, in the RENAMES clause of its data definition, the item, group, or sequence key named in the member-name part of the interrogation. CONSTITUTES: Obtains the names of items and/or groups and/or sequence keys, that are specified in the RENAMES clause of the data definition of the logical child segment named in the member-name part of the interrogation.
SEARCH-KEY-FIELDS	INDEX-POINTER-SEGMENTS SEGMENTS	ITEM GROUP	ITEMS GROUPS	INDEX-POINTER-SEGMENT	USES: Obtains the name of each index pointer segment that specifies, in the SEARCH-KEY-FIELDS clause of its data definition, the item or group named in the member-name part of the interrogation. CONSTITUTES: Obtains the names of items and/or groups, that are specified in the SEARCH-KEY-FIELDS clause of the data definition of the index pointer segment named in the member-name part of the interrogation.

Interrogation Syntax

Keyword for use in the VIA clause	USES interrogations: Meaningful member-type selection keywords	USES interrogations: Meaningful member types for member-name	CONSTITUTES interrogations: Meaningful member-type selection keywords	CONSTITUTES interrogations: Meaningful member types for member-name	Explanation/Notes
SECONDARY-SEQUENCE-ON	PROGRAM-COMMUNICATION-BLOCKS PCBS	INDEX-POINTER-SEGMENT	INDEX-POINTER-SEGMENTS SEGMENTS	PCB	This keyword is only applicable to structure type PCBs. USES: Obtains the name of each structure type PCB that specifies, in the ON subordinate clause of the SECONDARY- SEQUENCE clause of its data definition, the index pointer segment named in the member-name part of the interrogation. CONSTITUTES: Obtains the name of the index pointer segment that is specified in the ON subordinate clause of the SECONDARY-SEQUENCE clause of the data definition of the structure type PCB named in the member-name part of the interrogation.
SEGMENT	PROGRAM-COMMUNICATION-BLOCKS PCBS	INDEX-POINTER-SEGMENT LOGICAL-SEGMENT PHYSICAL-SEGMENT	INDEX-POINTER-SEGMENTS LOGICAL-SEGMENTS PHYSICAL-SEGMENTS SEGMENTS	PCB	This keyword is only applicable to structure type PCBs. USES: Obtains the name of each structure type PCB that specifies, in the SEGMENT clause of its data definition, the segment named in the member-name part of the interrogation. CONSTITUTES: Obtains the name of each segment that is specified in the SEGMENT clause of the data definition of the structure type PCB named in the member-name part of the interrogation.

	USES interrogations		CONSTITUTES interrogations		
Keyword for use in the VIA clause	**Meaningful member-type selection keywords**	**Meaningful member types for member-name**	**Meaningful member-type selection keywords**	**Meaningful member types for member-name**	**Explanation/Notes**
SENSITIVE-FIELDS	PROGRAM-COMMUNICATION-BLOCKS PCBS	ITEM GROUP SEQUENCE-KEY CONCATENATED-KEY	ITEMS GROUPS SEQUENCE-KEYS CONCATENATED-KEYS CONCATENATED-KEY-NAMES	PCB	This keyword is only applicable to structure type PCBs. USES: Obtains the name of each structure type PCB that specifies, in the SENSITIVE-FIELDS clause of its data definition, the member named in the member-name part of the interrogation. CONSTITUTES: Obtains the name of members that are specified in the SENSITIVE-FIELDS clause of the structure type PCB named in the member-name part of the interrogation.
SEQUENCE-KEY-CONSTITUENTS	PHYSICAL-SEGMENTS SEGMENTS	ITEM GROUP	ITEMS GROUPS	PHYSICAL-SEGMENT (virtual logical child segments only)	This keyword is only applicable to logical child segments, and only when an AS subordinate clause has been specified in the SEQUENCE-KEYS clause. The response includes only those members that precede the AS clause, not those specified in it. USES: Obtains the name of each logical child segment that specifies, in the SEQUENCE-KEY clause of its data definition, the item or group named in the member-name part of the interrogation. CONSTITUTES: Obtains the name of each item and/or group that is specified in the SEQUENCE-KEY clause of the data definition of the logical child segment specified in the member-name part of the interrogation.

Interrogation Syntax

Keyword for use in the VIA clause	USES interrogations: Meaningful member-type selection keywords	USES interrogations: Meaningful member types for member-name	CONSTITUTES interrogations: Meaningful member-type selection keywords	CONSTITUTES interrogations: Meaningful member types for member-name	Explanation/Notes
SEQUENCE-KEYS	PHYSICAL-SEGMENTS SEGMENTS	ITEM GROUP SEQUENCE-KEY	ITEMS GROUPS SEQUENCE-KEYS	PHYSICAL-SEGMENTS	USES: Obtains the name of each physical segment that specifies, in the SEQUENCE-KEY clause of its data definition: The item or group named in the member-name part of the interrogation, if an AS subordinate clause is not present, or The sequence key internal member named in the member-name part of the interrogation, if an AS subordinate clause is present. CONSTITUTES: Obtains the name of one of these: Each item or group specified, where no AS subordinate clause is present, or Each sequence key internal member specified, when an AS subordinate clause is present where these have been specified in the SEQUENCE-KEY clause of the data definition of the physical segment named in the member-name part of the interrogation.
SEQUENCE-KEYS	INDEX-POINTER-SEGMENTS SEGMENTS	SEQUENCE-KEYS	SEQUENCE-KEYS	INDEX-POINTER-SEGMENTS	USES: Obtains the name of the index pointer segment that specifies, in the SEQUENCE-KEY clause of its data definition, the sequence-key named in the member-name part of the interrogation. CONSTITUTES: Obtains the name of the sequence key that is specified in the SEQUENCE-KEY clause of the data definition of the index pointer segment named in the member-name part of the interrogation.

	USES interrogations		CONSTITUTES interrogations		
Keyword for use in the VIA clause	**Meaningful member-type selection keywords**	**Meaningful member types for member-name**	**Meaningful member-type selection keywords**	**Meaningful member types for member-name**	**Explanation/Notes**
SOURCE	INDEX-POINTER-SEGMENTS SEGMENTS	PHYSICAL-SEGMENT (source segment only)	PHYSICAL-SEGMENTS SEGMENTS	INDEX-POINTER-SEGMENT	USES: Obtains the name of each index pointer segment that specifies, in the SOURCE clause of its data definition, the index source segment named in the member-name part of the interrogation. CONSTITUTES: Obtains the name of the index source segment, that is specified in the SOURCE clause of the data definition of the index pointer segment named in the member-name part of the interrogation
SSAS	MODULES PROGRAMS SYSTEMS	INDEX-POINTER-SEGMENT LOGICAL-SEGMENT PHYSICAL-SEGMENT	INDEX-POINTER-SEGMENTS LOGICAL-SEGMENTS PHYSICAL-SEGMENTS SEGMENTS	MODULE PROGRAM SYSTEM	USES: Obtains the name of each module and/or program and/or system that specifies, in the SEGMENT-SEARCH- ARGUMENTS clause of the PROCESSES clause of its data definition, the segment named in the member-name part of the interrogation. CONSTITUTES: Obtains the name of each logical and/or physical and/or index pointer segment, that is specified in the SEGMENT-SEARCH-ARGUMENTS subordinate clause of the PROCESSES clause of the data definition of the module, system, or program named in the member-name part of the interrogation.
SUBSEQUENCE-FIELDS	INDEX-POINTER-SEGMENTS	ITEM GROUP SYSTEM-RELATED-FIELD	ITEMS GROUPS SYSTEM-RELATED-FIELDS	INDEX-POINTER-SEGMENT	USES: Obtains the name of each index pointer segment that specifies, in the SUBSEQUENCE-FIELDS clause of its data definition, the item, group, or system related field named in the member-name part of the interrogation. CONSTITUTES: Obtains the name of each item and/or group and/or system related field, that is specified in the SUBSEQUENCE-FIELDS clause of the data definition of the index pointer segment named in the member-name part of the interrogation.

Interrogation Syntax

Keyword for use in the VIA clause	USES interrogations: Meaningful member-type selection keywords	USES interrogations: Meaningful member types for member-name	CONSTITUTES interrogations: Meaningful member-type selection keywords	CONSTITUTES interrogations: Meaningful member types for member-name	Explanation/Notes
TARGET	INDEX-POINTER-SEGMENTS SEGMENTS	PHYSICAL-SEGMENTS (target segments only)	PHYSICAL-SEGMENTS SEGMENTS	INDEX-POINTER-SEGMENT	USES: Obtains the name of each index pointer segment that specifies, in the RELATED-TO clause of its data definition, the index target segment named in the member-name part of the interrogation. CONSTITUTES: Obtains the name of the index target segment, that is specified in the RELATED-TO clause of the data definition of the index pointer segment named in the member-name part of the interrogation.
TO	PHYSICAL-SEGMENTS SEGMENTS	PHYSICAL-SEGMENT (destination parent segments only)	PHYSICAL-SEGMENTS SEGMENTS	PHYSICAL-SEGMENT (logical child segments only)	USES: Obtains the name of each logical child segment that specifies, in the TO subordinate clause of its data definition, the destination parent segment named in the member-name part of the interrogation. CONSTITUTES: Obtains the name of the destination parent segment, that is specified in the TO subordinate clause of the data definition of the logical child segment named in the member-name part of the interrogation.
WITH	PHYSICAL-SEGMENTS SEGMENTS	PHYSICAL-SEGMENT (logical child segments only)	PHYSICAL-SEGMENTS SEGMENTS	PHYSICAL-SEGMENT (logical child segments only)	USES: Obtains the name of the logical child segment that specifies, in the WITH subordinate clause of the RELATED-AS clause of its data definition, the logical child segment named in the member-name part of the interrogation (that is, the logical child segment with which it is paired). CONSTITUTES: Obtains the name of the logical child segment, that is specified in the WITH subordinate clause of the RELATED-AS clause, in the data definition of the logical child segment named in the member-name part of the interrogation.

Alternative Verb Keywords

A number of verb keywords are available for use as alternatives to certain USES and CONSTITUTES interrogations. When these keywords are used, there is no need for a VIA clause to be supplied.

For example, the interrogation:

```
WHICH selection FATHERS member-name;
```

is equivalent to:

```
WHICH selected USES member-name VIA { PARENT  }
                                    { FATHERS }
```

The equivalents are shown in this table:

Alternative Verb Keyword	Equivalent USES/CONSTITUTES Interrogation
CONTAINS member-name	USES member-name VIA CONTAINS
CONTAINED-BY member-name	CONSTITUTES member-name VIA CONTAINS
FATHERS member-name	USES member-name VIA { PARENT | FATHERS }
FATHERED-BY member-name	CONSTITUTES member-name VIA { PARENT | FATHERS }
GENERATES member-name	USES member-name VIA GENERATED
GENERATED-BY member-name	CONSTITUTES member-name VIA GENERATES

5 DL/I Source Language Generation from DataManager

Introduction

The Source Language Generation facility can produce DL/I statements of these types:

- DL/I Database Description control statements (which can subsequently be used as input for an DL/I DBD generation)
- DL/I Program Specification Block control statements (which can subsequently be used as input for a DL/I PSB generation)
- Record layouts and/or COBOL, PL/I, or Assembler data description statements for users' segment input/output areas (defined through PCB members)
- Record layouts and/or COBOL, PL/I, or Assembler data description statements for Program Communication Block (PCB) masks
- Record layouts and/or COBOL, PL/I, or Assembler data description statements for segment search arguments

Generation of these statements is achieved by use of the PRODUCE command, described in the *ASG-Manager Products Source Language Generation* publication. The variations of the PRODUCE command required for the generations listed above are described in this chapter.

The PRODUCE command can also be used to generate MARK IV File Definition forms from encoded DL/I-DATABASE and SEGMENT members. The use of the PRODUCE command for this purpose is documented in the *ASG-DataManager MARK IV Interface* publication.

Generating DL/I DBD Control Statements

The installation macro DGDBD allows you to tailor generated DL/I DBD control statements to your own requirements. This macro is described in the Appendix, "Macros for Tailoring the DataManager DL/I Interface," on page 159.

This is the syntax of the PRODUCE command for generating DL/I Database Description (DBD) control statements:

Syntax

```
PRODUCE { IMS   } { DATABASE-DESCRIPTIONS } [FOR { PL/I } ]
        { DL/I  } { DBDGEN                }      { PL/1 }
        { DL/1  }                                { PLI  }
        { DLI   }                                { PL1  }
        { DL1   }

[ { SEARCH-FIELDS    } ] [NO-ASSEMBLY-PRINT]
  { DIRECT-FIELDS    }
  { GENERATES-FIELDS }
  { ALL-FIELDS       }

[PRIMARY-INDEX [DATABASE database-name] [SEGMENT segment-name]
       [SEQUENCE-KEY sequence-key-name] [AS library-name]]
FROM member-name [AS library-name]
[,member-name [AS library-name]]...

[control-options] { ; }
                  { . }
```

where:

database-name, *segment-name*, and *sequence-key-name* are valid DL/I names; database-name must not include a zero character.

member-name is the name of an encoded member that is a DL/I-DATABASE.

library-name is the name to be given to the generated library member in the output file. It must not be more than 16 characters, of which the first character must be alphabetic or one of the characters #, local currency symbol with the internal code hexadecimal 5B, or %.

control-options is a series of optional clauses that are defined in the *ASG-Manager Products Source Language Generation* publication; except that:

- The USE or USING clause defined there is excluded
- Only the KNOWN-AS option is valid in the GIVING clause
- Only the KNOWN-AS and ALIAS options are valid in the OMITTING clause
- If both the keywords NO-GENERATION and NO-PRINT are present in the command, no processing takes place

Remarks

1. The first three elements of the command must be the first three shown in the format. They must be in the order shown.

2. Specify a FOR clause when you want DBD FIELD control statements to include the two additional bytes required by PL/I for variable length fields.

3. None of the keywords SEARCH-FIELDS, DIRECT-FIELDS, GENERATES FIELDS, or ALL-FIELDS, or the PRIMARY-INDEX clause (and hence remark 4 on page 133 through remark 12 on page 134 and remark 18 on page 135 through remark 20 on page 136 describing these keywords) are relevant when processing a LOGICAL database.

4. If any of the keywords SEARCH-FIELDS, DIRECT-FIELDS, GENERATES-FIELDS, ALL-FIELDS, or NO-ASSEMBLY-PRINT, or the PRIMARY-INDEX clause are present in the command, they must precede the FROM clause.

5. If none of the keywords SEARCH-FIELDS, DIRECT-FIELDS, GENERATES- FIELDS, or ALL-FIELDS is specified in the command, SEARCH-FIELDS is assumed.

6. If any of the keywords SEARCH-FIELDS, DIRECT-FIELDS, GENERATES-FIELDS, or ALL-FIELDS is specified, DBD FIELD control statements are automatically generated for these types of field:

   - Sequence key fields
   - Index-search-fields (XDFLDS), if index target segments are being processed
   - System-related fields whose names are prefixed by a slash (/), if index source segments are being processed
   - Any field that is directly specified in the GENERATES clause of the segment being processed

7. If one of the keywords SEARCH-FIELDS, DIRECT-FIELDS, or ALL-FIELDS is specified, that is if GENERATES-FIELDS is not specified, DBD FIELD control statements are automatically generated for the following types of fields also:

   When processing a physical segment:

   - Segment search fields that are directly or indirectly contained by the segment. These fields are specified in the QUALIFIED-ON clause of the PROCESSES clause of SYSTEM, PROGRAM, or MODULE members that refer to the segment.
   - Sensitive fields that are directly or indirectly contained by the segment. These fields are specified in the SENSITIVE-FIELDS clause of PCB members.

   When processing an index pointer segment:

   - Any field that is used as a segment search field, or a sensitive field, or which is directly specified in the GENERATES clause of the segment being processed, but only if these fields constitute the user data part of the index pointer segment.

   When processing an index source segment:

   - Any field that is required for secondary indexing, that is, any field that directly occurs in the SEARCH, SUBSEQUENCE, or DUPLICATE-DATA lists of any index pointer segment that uses the index source segment being processed.

8. SEARCH-FIELDS specifies that DBD FIELD control statements are to be generated only for the fields described in remark 5 on page 133 and remark 6 on page 133.

9. DIRECT-FIELDS specifies that DBD FIELD control statements are to be generated for the fields described in remark 5 on page 133 and remark 6 on page 133, and for fields that are directly specified in the CONTAINS clause of the segment being processed.

10. GENERATES-FIELDS specifies that DBD FIELD control statements are only to be generated for the fields described in remark 5 on page 133 and for the fields that are directly specified in the GENERATES clause of the segment being processed.

    Thus, the GENERATES-FIELDS keyword suppresses the automatic generation by DataManager of fields that are specified as segment search fields, sensitive fields or fields used for secondary indexing, as described in remark 6 on page 133.

11. If GENERATES-FIELDS is specified, then when an index pointer segment is processed, DBD FIELD control statements are generated for all fields specified in the GENERATES clause regardless of whether they are part of the user data, or the SEARCH, SUBSEQUENCE, or DUPLICATE-DATA parts of the index pointer segment, or part of the target segment's concatenated key, (if this is included in the index pointer segment).

12. ALL-FIELDS specifies that DBD FIELD control statements are to be generated for:

    - All the fields that constitute the segment when a physical segment is being processed.
    - The sequence key field and all of the fields that constitute the user data part of the segment when an index pointer segment is being processed.

13. When processing arrays, DataManager generates a DBD FIELD control statement for the first occurrence of the array.

14. SEGM control statements are generated, in the correct hierarchical sequence, for each segment whose name is listed in the CONTAINS clause of the database's data definition.

15. For segments that participate in any logical or secondary indexing relationship, the operands for the SEGM control statements and their respective LCHILD control statements are obtained from the data definitions both of the segments being processed and of the segments to which these are related.

16. The operands for the DBD and DATASET control statements are obtained from the database's data definition. The DBDNAME applied to the generated DBD control statements is the database name.

17. For a HIDAM database:

    - The DBD control statements generated, if valid when complete, are immediately followed by the DBD control statements for its primary index database, which are generated automatically.
    - The names to be applied to the primary index database, its index pointer segment and the segment's sequence key field, can be specified in the PRIMARY-INDEX clause of the PRODUCE command.
    - If any of these names is not specified in the command, but is specified in the ACCESS clause of the HIDAM database definition, then the name specified in the latter clause is applied.
    - If different names are specified for the same entity in the PRODUCE command and the ACCESS clause, the name specified in the PRODUCE command is applied.
    - Where neither the PRODUCE command nor the ACCESS clause specifies the relevant name:
        - The name applied to the primary index database is the name of the HIDAM database suffixed with I.
        - The name of the index pointer segment is the name of the HIDAM root segment suffixed with I.
        - The name applied to the sequence key field of the index pointer segment is the name of the sequence key field for the HIDAM root segment suffixed with I.

          If any of these names becomes too long when suffixed with I, it is shortened by dropping the middle character.
    - The DBD control statements for the primary index database are written to the output file as a separate member. The library name of this member can be specified by the AS library-name subordinate clause of the PRIMARY-INDEX clause. If this clause is omitted, the library name applied is the library name of the HIDAM DBD control statements suffixed with I. If this name becomes too long when suffixed with I, it is not truncated, (see remark 22 on page 136), and generation of the member containing the control statements does not take place.

      In order to avoid this situation, a valid library name can be specified in the AS library-name subordinate clause of the PRIMARY-INDEX clause, or the MEMLEN parameter of the DGDBD tailoring macro can be used to extend the permissible length of library names (see "The Macros DGDBD and DGPSB" on page 160).

18. If NO-ASSEMBLY-PRINT is stated in the command, an Assembler PRINT NOGEN statement is generated, to eliminate listing of the DBD control statements when they are assembled.

19. The PRIMARY-INDEX clause can be present in the command only if one (and only one) of the member-names in the FROM clause is the name of a HIDAM database. If more than one of the member-names in the FROM clause are the names of HIDAM databases, and a PRIMARY-INDEX clause is present in the command, no generation is performed in respect of any HIDAM database name other than the first.

20. The PRIMARY-INDEX clause specifies, in respect of a HIDAM database named in the FROM clause, user names that are to be applied to:
    - The corresponding primary index database
    - The index pointer segment of the primary index database
    - The sequence key field of the index pointer segment
    - The library member name of the Database Description control statements for the primary index database

21. AS clauses are relevant only if DBD control statements are being written to an output dataset.

22. Each AS clause present in the command relates only to the member name that immediately precedes it. It declares a name under which the generated DBD control statements are to be cataloged in the output source library dataset.

23. For each member-name for which no AS clause is specified, library-name is defaulted to member-name if member-name conforms to the length restriction on library-name. The length restriction on library-name is a maximum of eight characters (unless tailored, see MEMLEN). If member-name is longer than the permitted maximum length for library-name, no generation takes place in respect of that member-name, a message is output, and processing continues with the next member-name or command.

24. Library-names, whether declared or defaulted, are not subjected to any name editing, nor to any ALIAS or WITH-ALIAS clauses (see the *ASG-Manager Products Source Language Generation* publication) that may be present in the command.

25. If ONTO filename is not specified in the PRODUCE command, a default file name of GENLIB is used (unless another name is specified by the DDNAME parameter of the macro DGDBD; see "The Macros DGDBD and DGPSB" on page 160).

26. The USE or USING clause is not applicable in the PRODUCE command for generation of DBD control statements, as the form and version of GROUP and ITEM members are obtained from the containing SEGMENT data definitions.

27. Other control-options clauses are as specified in the *ASG-Manager Products Source Language Generation* publication, except that the GIVING clause may only specify KNOWN-AS, and tile OMITTING clause may only specify OMITTING KNOWN-AS and/or ALIAS.

28. If GIVING KNOWN-AS is specified, generated data names are based wherever possible on local-names from:
    - Containing members' KNOWN-AS clauses
    - Logical child segments' RENAMES clauses

    instead of on the members' names or aliases. (The equivalent DGDBD macro keyword usage is KNOWNAS=YES.)

Generating DL/I PSB Control Statements

The installation macro DGPSB allows generated DL/I PSB control statements to be tailored to the installation's own requirements. This macro is described in Appendix, "Macros for Tailoring the DataManager DL/I Interface," on page 159.

This is the syntax of the PRODUCE command for generating DL/I Program Specification Block (PSB) control statements:

Syntax

```
PRODUCE { IMS   } { PROGRAM-SPECIFICATIONS }
        { DL/I  } { PSBGEN                 }
        { DL/1  }
        { DLI   }
        { DL1   }
[FOR { PL/I } ]   [NO-ASSEMBLY-PRINT]
     { PL/1 }
     { PLI  }
     { PL1  }

FROM member-name [AS library-name]
   [,member-name [AS library-name]]...
[control-options] { ; }
                  { . }
```

where:

member-name is the name of an encoded member that is a SYSTEM, a PROGRAM, or a MODULE.

library-name is the name to be given to the generated library member in the output file. It must not be more than 16 characters, of which the first character must be alphabetic or one of the characters #, local currency symbol with the internal code hexadecimal 5B, or %.

control-options is a series of optional clauses that are defined in the *ASG-Manager Products Source Language Generation* publication; except that:

- Only the INITIAL-VALUES and/or KNOWN-AS options are valid in the GIVING clause
- Only the ALIAS and/or INITIAL-VALUES and/or KNOWN-AS options are valid in the OMITTING clause
- If both the keywords NO-GENERATION and NO-PRINT are present in the command, no processing takes place

Remarks

1. The first elements in the command must be the first three shown in the format. They must be in the order shown.

2. Specify a FOR clause when you want DBD FIELD control statements to include the two additional bytes required by PL/I for variable length fields.

3. The optional keyword NO-ASSEMBLY-PRINT must, if present, precede the FROM clause.

4. The PSBNAME applied to the generated PSB control statements is the SYSTEM, PROGRAM, or MODULE member name.

5. The language operands for the PSBGEN control statement is obtained from the LANGUAGE clause of the SYSTEM, PROGRAM, or MODULE member being processed, provided that the character string in that clause is any of these:

ALC	ASSEMBLER	ASSEMBLY	BAL	COBOL
PLI	PL1	PL/I	PL/1	RPG

   If the character string is not one of these, or if the LANGUAGE clause is not present, then COBOL is assumed. The remaining types of control statements are generated from the PCB members listed in the CONTAINS subordinate clause of the PROCESSES clause in the data definition of the SYSTEM, PROGRAM, or MODULE member.

6. PCB control statements are generated in the same sequence as that in which PCB members have been defined in the PROCESSES clause.

7. For the PCB for a logical-data-structure, if KEYLENGTH has not been specified in the PCB definition then the value of the KEYLENGTH operand is calculated by DataManager as the length of the largest concatenated key for all data-sensitive segments specified in the relevant member.

8. The PROCSEQ operand is generated by DataManager if one of the SEGMENT clauses specified for the PCB member contains the keyword SECONDARY-SEQUENCE.

9. SENSEG control statements are generated in the correct hierarchical sequence for:

   - Each SEGMENT clause specified in a logical-data-structure PCB member
   - Each segment along the hierarchical paths to those segments

   subject to a maximum of 500 segments.

10. Following each SENSEG statement generated, if sensitive fields are defined for that segment in the PCB data definition, DataManager generates:

    - A SENFLD statement for each sensitive field specified that is directly or indirectly contained by the segment
    - A VIRFLD statement for each sensitive field specified that is directly or indirectly contained by the segment
    - A SENFLD statement for each constituent member of a sensitive field that is indirectly contained by the segment, if SUBFIELDS has been specified for the sensitive field in the PCB member definition
    - A VIRFLD statement for each constituent member of a sensitive field that is indirectly contained by the segment, if SUBFIELDS has been specified for the sensitive field in the PCB member definition

    The statements are generated in the order in which the sensitive fields are specified, and the start position for each sensitive field is calculated from the lengths of any preceding sensitive fields together with any preceding filler-bytes specified.

11. If it is required to generate a VALUE operand on any VIRFLD statement, then:

    - The relevant item member's definition must include a CONTENTS IS clause.
    - Either the control-option GIVING INITIAL-VALUES must be specified in the PRODUCE command, or the values of INITVAL in the DGPSB macro must be YES.

12. All names generated are subject to any editing specified in the command.

13. AS clauses are relevant only if PSB control statements are being written to an output dataset.

14. Each AS clause present in the command relates only to the `member-name` that immediately precedes it. It declares a name under which the generated PBC control statements are to be cataloged in the output source library dataset.

15. For each member-name for which no AS clause is specified, `library-name` is defaulted to `member-name` if `member-name` conforms to the length restriction on `library-name`. The length restriction on `library-name` is a maximum of 8 characters (unless tailored, see MEMLEN). If `member-name` is longer than the permitted maximum length for `library-name`, no generation takes place in respect of that `member-name`, a message is output, and processing continues with the next `member-name` or command.

16. `Library-name`s, whether declared or defaulted, are not subjected to any name editing, nor to any ALIAS or WITH-ALIAS clauses (see the *ASG-Manager Products Source Language Generation* publication) that may be present in the command.

17. If ONTO `file-name` is not specified in the PRODUCE command, DataManager uses a default file name of GENLIB (unless another name is specified by the DDNAME parameter of the macro DGPSB; see Appendix, "Macros for Tailoring the DataManager DL/I Interface," on page 159).

18. Except as stated in remark 19 on page 140, the USE or USING control-options clause is not applicable in the PRODUCE command for generation of PSB control statements, as the form and version of any group or item sensitive field are obtained from the containing SEGMENT data definition.

19. The USE or USING clause is relevant only:

    - If it is required to use the DL/I TYPE conversion facility on sensitive fields, when the form and version to which the fields are to be converted can be specified in this clause
    - When processing virtual fields

20. Other control-options clauses are as specified in the *ASG-Manager Products Source Language Generation* publication, except that:

    - The GIVING clause may only include INITIAL-VALIES and/or KNOWN-AS.
    - The OMITTING clause may only include the options ALIAS, INITIAL-VALUES, and KNOWN-AS.

21. If GIVING KNOWN-AS is specified, generated data names are based whenever possible on local-names from:

    - KNOWN-AS clauses specified for the sensitive fields in the PCB member definition
    - Containing members' KNOWN-AS clauses, when processing the members that constitute a sensitive field. (The members that constitute the sensitive field are only processed if SUBFIELDS has been specified for the sensitive field in the PCB member definition.)

    instead of on the members' names or aliases. (The equivalent DGPSB macro keyword usage is KNOWNAS=YES.)

    It should be noted that the generated data names are not based on the KNOWN-AS clauses that are directly specified in the SEGMENT definition's CONTAINS clause.

Generation of COBOL, PL/I, or Assembler Data Description Statements for Segment Input/Output Areas

The PRODUCE Command

The format of the PRODUCE command to generate COBOL, PL/I, or Assembler data description statements (and/or record layouts) for segment input/output areas is as described in the *ASG-Manager Products Source Language Generation* publication.

The member-name in the FROM clause must be the name of an encoded SEGMENT member, and the USE or USING clause is not applicable (because the form and version of contained GROUP and ITEM members are determined from the segment data definition). If the USE or USING clause is present in the command because it is required for members of other types also named in the FROM clause, it is ignored when SEGMENT members are processed.

The PRODUCE command can also generate COBOL, PL/I, or Assembler data description statements for certain types of DL/I fields for which data dictionary members of special internal types exist. See "Miscellaneous DL/I Fields" on page 148. In these cases, the member-name in the FROM clause is the name of the field for which the internal member was created.

Installation Macros

Three installation macros are provided, which allow the names that are to be applied to certain lines of the generated data descriptions to be specified. These are the macros:

- DOSCOB, which is relevant to COBOL language generation
- DGSPLI, which is relevant to PL/I language generation
- DGSBAL, which is relevant to Assembler language generation

These macros are described in the Appendix, "Macros for Tailoring the DataManager DL/I Interface," on page 159.

The data description statements that are generated for the various types of segments are described in "Simple Physical Segments" on page 142 through "Miscellaneous DL/I Fields" on page 148.

A fourth installation macro, DGSREC, applies if record layouts are produced without associated source language data description generation. This macro is also described in the Appendix, "Macros for Tailoring the DataManager DL/I Interface," on page 159.

The installation macros DGCOB, DOPLI, DGBAL, and DGREC, described in the *ASG-Manager Products Source Language Generation* publication, also apply, respectively, when segment input/output area data descriptions are generated in COBOL, in PL/I, in Assembler, or in record layout form without associated source language.

Segment Input/Output Areas: Items Defined as BINARY or BITS

Except as stated below, if a binary item or a bit string item is ALIGNED by virtue of the definition of the containing GROUP or SEGMENT, then:

- A one-byte binary item is rounded up to two bytes in length
- A three-byte binary item is rounded up to four bytes in length
- A five-, six-, or seven-byte binary item is rounded up to eight bytes in length.
- Each bit string item begins on the next available byte boundary

If a binary or bit string item is a sequence key field, or a part of a sequence key field, of

- A destination parent segment, or
- An index pointer segment, or
- An index source segment, or
- A segment in the hierarchical path of a destination parent segment, an index pointer segment or an index source segment

then when it forms part of

- A logical child segment, by virtue of the destination parent's concatenated key, or
- An index pointer segment, by virtue of the index target segment's concatenated key, or
- A system related field, by virtue of the index source segment's concatenated key

the binary or bit string item is not aligned. The length of unaligned binary items is not rounded up unless the value of RNDBIN in the relevant macro DGCOB, DGPLI, DGBAL, or DGREC is YES. Bit string items, if not aligned, do not begin at the next byte boundary unless the RNDBIT parameter in the tailoring macros is set to YES. If the lengths of binary or bit string items are to be consistent in different contexts (for example, in CONTAINS clauses and in concatenated keys) or in different languages (for example, COBOL and BAL) the value of RNDBIN and RNDBIT in these macros must be set to YES.

Simple Physical Segments

For a simple physical segment that participates in no logical or secondary indexing relationships, data description statements are generated in the same manner as for a GROUP member.

Logical Child Segments

The COBOL, PL/I, or Assembler data description statements generated for a logical child segment include the concatenated key of the destination parent.

A line is generated containing the name to be applied to the concatenated key. The name output is the name specified in the CONCATENATED-KEY-NAME clause of the segment definition, if specified; otherwise the name is obtained from the macro DGSCOB, DOSPLI, DGSBAL, or DGSREC, as appropriate. This line is followed by the description of the constituent concatenated keys, each one generated separately down to ITEM level. If there is any intersection data, it is preceded by a line containing the name to be applied to the user data, which is also obtained from the appropriate macro. The two names obtained, whether from the segment definition or from the appropriate macro, are subjected to any editing that is specified in the command.

The following illustrates the structure of COBOL or PL/I data description statements generated for a logical child segment:

```
01  LOGICAL-CHILD-SEGMENT-NAME
   03 CONCATENATED-KEY-NAME
      05 KEYA
      05 KEYB
      05 KEYC
   03 USER-DATA-NAME
      05 FIELDA
      05 FIELDB
      05 FIELDC
```

If the data definition for a logical child segment includes AS sequence-key-name, the generated data description statements do not include sequence-key-name. If required, COBOL, PL/I, or Assembler data description statements for this type of field can be generated separately in their own right, as described in "Miscellaneous DL/I Fields" on page 148.

The application program could include a COPY or %INCLUDE statement for the segment, followed by a COPY or %INCLUDE statement for the sequence-key-name field; then if the program is written in Assembler, the sequence-key-name field can be ORGed back to the starting position of the sequence key field; or if the program is written in PL/I, the sequence-key-name field can be generated as a based structure whose pointer is set to the starting position of the sequence key field.

Destination Parent Segments

Destination parent segments are treated as ordinary physical segments; that is, data description statements are generated in the same manner as for a GROUP member.

Index Target and Index Source Segments

Index target and index source segments are treated as ordinary physical segments; that is, data description statements are generated in the same manner as for a GROUP member.

If COBOL, PL/I, or Assembler data description statements are required for XDFLD fields (that is, index-search-field-name fields that are defined in SEGMENT INDEX-POINTER members) or for system related fields, they can be generated separately in their own right, as described in "Miscellaneous DL/I Fields" on page 148.

Logical Segments and Logical Concatenated Segments

The COBOL, PL/I, or Assembler data description statements for a logical segment are generated from the physical segment represented by the logical segment; except that the name in the first statement is that of the logical segment.

The data description statements generated for a logical concatenated segment are generated from the two physical segments represented by the logical concatenated segment (except that the name in the first statement is that of the logical concatenated segment). The following illustrates the structure of COBOL or PL/I data description statements generated for a logical concatenated segment:

```
01 CONCATENATED-SEGMENT-NAME
  03 LOGICAL-CHILD-SEGMENT-NAME
    05 CONCATENATED-KEY-NAME
      07 KEYA
      07 KEYB
      07 KEYC
    05 USER-DATA-NAME
      07 FIELDA
      07 FIELDB
      07 FIELOC
  03 DESTINATION-PARENT-SEGMENT-NAME
    05 FIELDD
    05 KEYC
    05 FIELDE
```

In this illustration two different lines are generated for KEYC, the key field of the destination parent; however, the fields can be distinguished from one another in the application program by qualifying the appropriate field with either the logical child segment name or the destination parent segment name. In Assembler data description statements, the second and subsequent occurrences of duplicated names are blanked out.

Variable Length Segments

A variable length segment is defined to DataManager by specifying that the segment contains, directly or indirectly, a variable length item member. A segment that directly or indirectly contains a variable length array is not recognized as a variable length segment by DataManager.

If COBOL data description statements are to be generated for a variable length segment, the segment must contain a variable length ITEM member, and this member must be redefined by a variable length array. This is to satisfy the requirements of the VS COBOL compiler, which only recognizes a segment as being of variable length if a variable length array is contained in the segment.

For example, if a COBOL data description were generated from this data definition:

```
CONTAINS
ITEMA ELSE (ITEMB)ITEMC
;
```

the VS COBOL compiler would output a warning message and compilation would continue. However, this definition:

```
CONTAINS
(ITEMB)ITEMC ELSE ITEMA
;
```

would cause the VS COBOL compiler to output an error message and compilation would fail.

The COBOL, PL/I, or Assembler data description statements generated for a variable length segment include a line for the two byte size field. The name to be applied to this line is taken from the macro DGSCOB, DOSPLI, DGSBAL, or DGSREC, as appropriate. The name is subjected to any editing specified in the command.

This illustrates the structure of COBOL or PL/I data description statements generated for a variable length physical segment:

```
01 SEGMENT-NAME
  03 SIZE-FIELD-NAME
  03 FIELDA
  03 FIELDB
```

This illustrates the structure of COBOL or PL/I data description statements generated for a variable length logical concatenated segment:

```
01 CONCATENATED-SEGMENT-NAME
  03 LOGICAL-CHILD-SEGMENT-NAME
    05 SIZE-FIELD-NAME
    05 CONCATENATED-KEY-NAME
      07 KEYA
      07 KEYB
      07 KEYC
    05 USER-DATA-NAME
      07 FIELDA
      07 FIELDB
      07 FIELDC
  03 DESTINATION-PARENT-SEGMENT-NAME
    05 SIZE-FIELD-NAME
    05 FIELDD
    05 KEYC
    05 FIELDE
```

If both parts of a logical concatenated segment are variable length, then the two size fields can be distinguished from one another in the application program, by qualifying the required size field with either the logical child segment name or the destination parent segment name, as appropriate. In Assembler data description statements, the second and subsequent occurrences of duplicated names are blanked out.

Path Calls

Data description statements for a user's input/output area that is to handle segments accessed in a path call can be obtained in this way:

- A separate COBOL, PL/I, or Assembler data description must be generated for each of the data sensitive segments to be processed in the path call. (The starting level number can be specified in the command.)
- The application program must then issue for its input/output area contiguous COPY or %INCLUDE statements for each of the data sensitive segments to be concatenated.

Index Pointer Segments

The Source Language Generation facility produces a complete and comprehensive set of COBOL, PL/I, or Assembler data description statements for index pointer segments.

The macros DGSCOB, DGSPLI, DGSBAL, and DGSREC are used widely in the generation of these data description statements. The statements generated include statements containing names, obtained from the appropriate macro, that identify and separate parts of the index pointer segment. These are parts of the segment to which there is no particular requirement to apply a name in the data dictionary data definition, but which the user might possibly wish to process as entities. The approach is adopted to make it easier for the user to process any constituent parts of the index pointer segments.

This example illustrates the structure of COBOL or PL/I data description statements generated for a complex index pointer segment. All constituent members are generated down to ITEM level. All names are subject to any editing specified in the PRODUCE command.

Data Description Statements	See Remark Number:
`01 INDEX-POINTER-SEGMENT-NAME`	1
`  03 KEY-NAME`	2
`  05 INDEX-FIELD-NAME`	2, 3
`  07 FIELD-A`	
`  07 FIELD-B`	
`  07 FIELD-C`	
`  05 SUBSEQUENCE-NAME`	4
`  07 CKA`	5
`  09 KEYA`	
`  09 FIELD-D`	
`  07 SXA`	6
`  07 CKB`	7
`  09 KEYB`	
`  03 DUPLICATE-DATA-NAME`	8
`  05 CKA`	
`  07 KEYA`	
`  07 FIELD-D`	
`  05 CKB`	
`  07 KEYB`	
`  05 CKC`	
`  07 FIELD-E`	
`  03 USER-DATA-NAME`	9
`  05 FIELD-F`	
`  05 FIELD-G`	
`  05 FIELD-H`	

Remarks

1. The first line contains the member-name of the index pointer segment for which data description statements are being generated, and is always generated (except, for COBOL generation, when the value of the GEN keyword of the DOCOB macro is FD).

2. This name is obtained from the member's data definition, and is always generated.

3. This field includes the members defined in the related index source segment's definition to constitute the search field. It represents the search field that can be used in segment-search-arguments when accessing the related index target segment.

4. SUBSEQUENCE-NAME is obtained from the macro DGSCOB, DOSPLI, or DGSBAL, as appropriate. It is generated only if subsequence fields are specified for the index pointer segment. The field includes the system related fields defined in the related index source segment's definition, that are specified in the index pointer segment's definition to constitute the subsequence fields.

5. This is a system related field of the type that is constituted by any part of the source segment's concatenated key. In this illustration its constituent members are a sequence key field followed by a constituent member of the next contiguous sequence key field in the source segment's concatenated key.

6. This is a system related field of the type that prompts DL/I to generate a unique four byte value.

7. This is another system related field of the type that is constituted by any part of the source segment's concatenated key; but this field has only one constituent, a sequence key field.

8. DUPLICATE-DATA-NAME is obtained from the macro DOSCOB, DGSPLI, or DGSBAL as appropriate. It is generated only if duplicate-data fields are specified in the index pointer segment. The field includes the system related fields defined in the related index source segment's definition, that are specified in the index pointer segment's definition to constitute the duplicate-data fields.

9. USER-DATA-NAME is obtained from the macro DGSCOB, DGSPLI, or DGSBAL as appropriate. It is generated only if the index pointer segment contains user data.

With COBOL and PL/I data description statements, any duplicate names that are generated can be distinguished from one another by qualifying them with higher level fields whose names are unique.

When Assembler data description statements are generated, each of the fields constituting the index-field-data, subsequence-data, duplicate-data and the DL/I generated concatenated-key-data are given unique names by DataManager, to allow for the same field appearing more than once in the segment. This is achieved by concatenating each constituent field name to either the INDEX-FIELD-NAME, SUBSEQUENCE-NAME, or DUPLICATE-DATA-NAME, depending on where it appears. If a name becomes too long it is shortened by dropping characters from the middle.

To ensure uniqueness of field names where more than one segment is involved, the user must, if necessary, use separate PRODUCE commands for the different segments, and include editing clauses in the PRODUCE commands.

Miscellaneous DL/I Fields

The DataManager Source Language Generation facility can be used to generate record layouts or COBOL, PL/I, or Assembler data description statements for these types of DL/I fields:

- Sequence-key-name fields, with a line generated for each constituent member down to ITEM level. If a sequence-key-name field has been defined for a virtual logical child segment, only the sequence-key-name field named in the PRODUCE command is generated. If more than one sequence-key-name field is defined for the segment, then each one required must be generated separately; contiguous COPY or %INCLUDE statements can subsequently be issued in the application program to include them concatenated together.
- Index-search-field-name fields (XDFLDs), with a line generated for each constituent member down to ITEM level
- System-related fields, with a line generated for each constituent member down to ITEM level
- Concatenated-key-name fields, with a line generated for each constituent member down to ITEM level

Generation of COBOL, PL/I, or Assembler Data Description Statements for Segment Sensitive Fields Input/Output Areas

The format of the PRODUCE command to generate COBOL, PL/I, or Assembler data description statements (and/or record layouts) for segment sensitive fields input/output areas is as specified in the *ASG-Manager Products Source Language Generation* publication, with the addition of a qualifier clause. This is the format of the qualifier clause, which immediately precedes the command's FROM clause:

`USED-IN pcb-name`

where `pcb-name` is the name of a STRUCTURE type PROGRAM-COMMUNICATION-BLOCK or PCB member.

The member-name in the FROM clause must be the name of a SEGMENT member.

This form of the PRODUCE command first generates a source language (or record layout) data description line for the segment being processed. A line is then generated for each sensitive field specified for that segment in the PCB member named in the USED-IN clause. These lines are generated for the fields in the order in which the fields are specified, with fillers generated wherever filler-bytes are specified in the PCB member.

If no sensitive fields have been specified for the segment in the PCB member definition, then statements or record layouts are generated as they would be normally, as if the USED-IN clause had not been specified.

If GIVING KNOWN-AS is specified, the generated data names are based on local-names from:

- KNOWN-AS clauses specified for the sensitive fields in the PCB member definition
- Containing members' KNOWN-AS clauses, when processing the members that constitute a sensitive field

instead of on the members' names or aliases. (The equivalent DGCOB, DGPLI, DGBAL, or DGREC macro keyword usage is KNOWNAS=YES.)

It should be noted that the generated data names are not based on the KNOWN-AS clauses that are directly specified in the SEGMENT definition's CONTAINS clause.

Example

Using the example segment ASY-PACK shown in Figure 3 on page 9 and the related example PCB member ASY-PACK-PCB shown in the examples section of "DataManager Data Definition Statements for DL/I Program Communication Blocks" on page 82 (example of STRUCTURE type PCB), this command could be issued to generate COBOL data description statements for the segment sensitive fields input/output area:

```
PRODUCE COBOL USED-IN ASY-PACK-PCB FROM ASY-PACK;
```

These would be the generated source language statements:

```
01 ASY-PACK,
  03 PACK-NO ---,
  03 FILLER PIC XX,
  03 PROD-NO ---,
  03 QTY-REQD.
```

Generation of COBOL, PL/I, or Assembler Data Description Statements for PCB Masks

The PRODUCE command can be used to generate COBOL, PL/I or Assembler data description statements and/or record layouts for PCB masks. In order to do this, each PCB mask must be defined to DataManager as a GROUP containing these members:

- An ITEM member with a length of eight bytes and a CHARACTER form-description, to receive the database name returned by DL/I
- An ITEM member with a length of two bytes and a CHARACTER form-description, to receive the segment level number returned by DL/I
- An ITEM member with a length of two bytes and a CHARACTER form-description, to receive the status code returned by DL/I
- An ITEM member with a length of four bytes and a CHARACTER form-description, to contain the list of processing options required by DL/I
- An ITEM member with a length of four bytes and a BINARY form-description, to be used by DL/I for internal linkage
- An ITEM member with a length of eight bytes and a CHARACTER form-description, to contain the segment name returned by DL/I
- An ITEM member with a length of four bytes and a BINARY form-description, to contain the length of the key feedback area
- An ITEM member with a length of four bytes and a BINARY form-description, to receive the figure returned by DL/I for the number of sensitive segment types to which the application program is sensitive
- An ITEM member with a CHARACTER form-description and of sufficient length to receive the concatenated key of the segment returned by DL/I. The length of this item is defined by the value of the length of key feedback field.

Example

This example shows how a PCB mask, named DB-PCB, might be defined to DataManager.

```
ADD DB-PCB;
GROUP
CONTAINS DB-NAME,SEG-LEVEL,STAT-CODE,PROC-OPT,FILLER,
         SEG-NAME,LN-KFB,NU-SENSEG,KEY-FB
;
ADD DB-NAME;
ITEM
HELD-AS CHAR 8
;
ADD SEG-LEVEL;
ITEM
HELD-AS CHAR 2
;
ADD STAT-CODE;
ITEM
HELD-AS CHAR 2
;
ADD PROC-OPT;
ITEM
HELD-AS CHAR 4
;
ADD FILLER;
ITEM
HELD-AS BINARY 9;
;
ADD SEG-NAME;
ITEM
HELD-AS CHAR 8
;
ADD LENG-KFB;
ITEM
HELD-AS BINARY 9
;
ADD NU-SENSEG
ITEM
HELD-AS BINARY 9
;
ADD KEY-FB;
ITEM
HELD-AS CHAR 100
;
```

COBOL data description statements could be generated from this definition by this command:

```
PRODUCE COBOL FROM DB-PCB NOGEN PRINT USING HELD-AS;
```

These statements would be produced:

```
01 DB-PCB
   02   DB-NAME      PIC X(8).
   02   SEG-LEVEL    PIC XX.
   02   STAT-CODE    PIC XX.
   02   PROC-OPT     PIC X(4).
   02   FILLER       PIC S9(9)    COMP.
   02   SEG-NAME     PIC X(8).
   02   LEN-KFB      PIC S9(9)    COMP.
   02   NU-SENSEG    PIC S9(9)    COMP.
   02   KEY-FB       PIC X(100).
```

PL/I data description statements could be generated by this command:

```
PRODUCE PL/I FROM DB-PCB NOGEN PRINT USING HELD-AS;
```

and these statements would be produced:

```
DCL
01 DB-PCB,
   3    DB-NAME    CHAR (8),
   3    SEG-LEVEL  CHAR (2),
   3    STAT-CODE  CHAR (2),
   3    PROC-OPT   CHAR (4),
   3    FILLER     FIXED BIN (31),
   3    SEG-NAME   CHAR (8),
   3    LEN-KFB    FIXED BIN (31),
   3    NU-SENSEG  FIXED BIN (31),
   3    KEY-FB     CHAR (100)
```

Assembler data description statements could be generated by this command:

```
PRODUCE BAL FROM DB-PCB NOGEN PRINT USING
HELD-AS DROPPING "-";
```

and these statements would be produced:

```
DBPCB      DS    0CL136
DBNAME     DS    CL8
SEGLEVEL   DS    CL2
STATCODE   DS    CL2
PROCOPT    DS    CL4
FILLER     DS    FL4
SEGNAME    DS    CL8
LENKFB     DS    FL4
NUSENSEG   DS    FL4
KEYFB      DS    CL100
*                    END OF GROUP DBPCB
;
```

Generation of COBOL, PL/I, or Assembler Data Description Statements for Segment Search Arguments

The definition of segment search arguments used during the generation of DBD control statements is described in "Specification of the PROCESSES Clause" on page 92. The section below describes how to define segment search arguments for the generation of COBOL, PL/I, or Assembler, or record layouts.

The PRODUCE command can be used to generate COBOL, PL/I, or Assembler data description statements and/or record layouts for segment search arguments. In order to do this, each segment search argument for which data description statements are to be generated must be defined to DataManager as a GROUP member, and its component parts must be defined as ITEM members contained by that GROUP.

An unqualified segment search argument should be defined as a GROUP containing:

- An ITEM member with a length of eight bytes, a CHARACTER form-description, and a CONTENTS clause that specifies the name of the segment to be searched.
- An ITEM member with a length of one byte, a CHARACTER form-description, and a CONTENTS clause that specifies the asterisk character (*). This field is necessary only if a command code is included in the segment search argument.
- An ITEM member with a length of one to four bytes and a CHARACTER form description. This field will receive command codes from the application program. Alternatively, the member could have a CONTENTS clause specifying up to four command codes for the segment search argument. This field is not required if no command codes are to be included in the segment search argument.
- An ITEM member with a length of one byte, a CHARACTER form description, and a CONTENTS clause that specifies a space character.

A qualified segment search argument should be defined as a GROUP containing the first three items listed above, plus:

- An ITEM member with a length of one byte, a CHARACTER form description, and a CONTENTS clause that specifies the left parenthesis character to indicate the start of the qualification statement.
- An ITEM member with a length of eight bytes, a CHARACTER form-description, and a CONTENTS clause that specifies the name of the search field.
- An ITEM member with a length of two bytes, a CHARACTER form-description, and a CONTENTS clause that specifies the relational operator.
- An ITEM member with a CHARACTER form-description, and a CONTENTS clause that specifies the value that is to be compared with the values of the fields being searched. The length of this field must be the same as that specified in the DataManager data definition of the segment search field.
- An ITEM member with a length of one byte, and a CHARACTER form-description, with a CONTENTS clause that specifies the right parenthesis character to indicate the end of the qualification statement.

The standard segment search argument format described above and illustrated below may be varied in two ways:

- If the C command code is used to retrieve a segment by its concatenated key, the qualification statement must be replaced by an ITEM member with a CHARACTER form-description and of the appropriate length to receive the concatenated key of the required segment.
- Fields can be included to allow multiple qualification statements to be specified. The fields required would be, for each additional qualification statement:
  - An ITEM member with a length of one byte, a CHARACTER form-description, and a CONTENTS clause that specifies the logical operator
  - An ITEM member with a CONTENTS clause that specifies the name of the search field, as described above
  - An ITEM member with a CONTENTS clause that specifies the relational operator, as described above
  - An ITEM member with a CONTENTS clause that specifies the value to be compared with the values of fields being searched, as described above.

Example

This segment search argument:

Segment Name	*	Command Code	Begin QS	Field Name	R.O.	Value	End QS
TEST-SEG	*	---	(	TESTFLD	EQ	AA	)

could be defined as a GROUP named TEST-SSA containing the ITEMs SSEGNAME, SCCSEP, SCOMCODE, SLPAREN, SFLDNAME, SCOMPOP, SFLDVAL, and SRPAREN, as shown here:

```
ADD TEST-SSA;
GROUP
CONTAINS SSEGNAME, SCCSEP, SCOMCODE, SLPAREN, SFLDNAME,
         SCOMPOP, SFLDVAL, SRPAREN
;
ADD SSEGNAME;
ITEM
HELD-AS CHAR 8
CONTENTS IS "TEST-SEG"
;
ADD SCCSEP;
ITEM
HELD-AS CHAR 1
CONTENTS IS "*"
;
ADD SCOMCODE;
ITEM
HELD-AS CHAR 4
CONTENTS IS "---"
;
ADD SLPAREN;
ITEM
HELD-AS CHAR 1
CONTENTS IS "("
;
ADD SFLDNAME;
ITEM
HELD-AS CHAR 8
CONTENTS IS "TESTFLD"
;
ADD SCOMPOP;
ITEM
HELD-AS CHAR 2
CONTENTS IS "EQ"
;
ADD SFLDVAL;
ITEM
HELD-AS CHAR 2
CONTENTS IS "AA"
;
ADD SRPAREN;
ITEM
HELD-AS CHAR 1
CONTENTS IS ")"
;
```

COBOL data description statements could then be generated from this definition by this command:

```
PRODUCE COBOL FROM TEST-SSA NOGEN PRINT USING HELD-AS
GIVING INITIAL VALUES;
```

These data description statements would be generated:

```
01 TEST-SSA.
   02 SSEGNAME    PIC X(8)
                  VALUE "TEST-SEG".
   02 SCCSEP      PIC X
                  VALUE "*".
   02 SCOMCODE    PIC X(4)
                  VALUE "---".
   02 SLPAREN     PIC X
                  VALUE "(".
   02 SFLDNAME    PIC X(8)
                  VALUE "TESTFLD".
   02 SCOMPOP     PIC XX
                  VALUE "EQ".
   02 SFLDVAL     PIC XX
                  VALUE "AA".
   02 SRPAREN     PIC X
                  VALUE ")".
```

PL/I data description statements could be generated by this command:

```
PRODUCE PL/I FROM TEST-SSA NOGEN PRENT USING HELD-AS GIVING
INITIAL-VALUES;
```

and these statements would be produced:

```
DCL
1 TEST-SSA,
   3 SSEGNAME     CHAR(8)
                  INIT ('TEST-SEG'),
   3 SCCSEP       CHAR(1)
                  INIT ('*'),
   3 SCOMCODE     CHAR(4)
                  INIT ('----'),
   3 SLPAREN      CHAR(1)
                  INIT ('('),
   3 SFLDNAME     CHAR(8)
                  INIT ('TESTFLD'),
   3 SCOMPOP      CHAR(2)
                  INIT ('EQ'),
   3 SFLDVAL      CHAR(2)
                  INIT ('AA'),
   3 SRPAREN      CHAR(1)
                  INIT (')');
```

BAL data description statements could be generated by this command:

```
PRODUCE BAL FROM TEST-SSA NOGEN PRINT USING HELD-AS
DROPPING "-" GIVING INITIAL-VALUES;
```

and these statements would be produced:

```
TESTSSA   DS   DCL27
SSEGNAME  DC   CL8'TEST-SEG'
SCCSEP    DC   CL1'*'
SCOMCODE  DC   CL4'----'
SLPAREN   DC   CL14'('
SFLDNAME  DC   CL8'TESTFLD'
SCOMPOP   DC   CL2'EQ'
SFLDVAL   DC   CL2'AA'
SRPAREN   DC   CL1')'
*                        END OF GROUP TESTSSA
```

Appendix

Macros for Tailoring the DataManager DL/I Interface

Implementation of the DL/I Interface Macros

Several macros (additional to those described in the *ASG-Manager Products Source Language Generation* publication) are available to enable DL/I interface output generated by the PRODUCE command of DataManager to be tailored to conform to a particular installation's standards. These are the macros:

- DGDBD, to enable the output of Database Description (DBD) control statements to be tailored
- DGPSB, to enable the output of Program Specific Blocks (PSB) control statements to be tailored
- DGSCOB, to enable COBOL source language output to be tailored
- DGSPLI, to enable PL/I source language output to be tailored
- DGSBAL, to enable Assembler source language to be tailored
- DGSREC, to enable the output of record layouts to be tailored

These macros are supplied as source modules on the installation tape. The tables in "The Macros DGDBD and DGPSB" on page 160 (for DGDBD and DGPSB) and "The Macros DGSCOB, DGSPLI, DGSBAL, and DGSREC" on page 161 (for DGSCOB, DGSPLI, DGSBAL, and DGSREC) list the keywords of the macros, for which values can be specified when DataManager is installed. For any macro, if the supplied default values of all these keywords are acceptable, no further action need be taken in respect of the macro. If any values are to be changed, the procedure described in the *ASG-Manager Products Installation in DOS Environments* publication must be carried out.

These are the names of the resulting assembled module:

- DIL88 if the macro is DGDBD
- DIL89 if the macro is DGPSB
- DIL99 if the macro is DGSCOB
- DIL98 if the macro is DGSPLI
- DIL97 if the macro is DGSBAL
- DIL96 if the macro is DGSREC

The Macros DGDBD and DGPSB

The macros DGDBD and DGPSB respectively enable the generation of Database Description (DBD) control statements and of Program Specification Block (PSB) control statements to be tailored. This table lists the keywords of these macros for which values can be specified when DataManager is installed:.

The Macros DGDBD and DGPSB: Keywords Specifiable on Installation

Keyword	Specifies	Default Value	Alternative Values
ACHAR	The hexadecimal values of any additional characters that are to be accepted for output in names produced by the Source Generation facility, to enable characters not in the standard source language set to be output (see Note 1)	No default	Any valid hexadecimal value, or a sub-list of such values
ALIAS	Whether IMS specific aliases are to be generated instead of member names	NO	YES (see Note 2)
COLMAIN	Starting character position for statement type	10	Up to 99
COLSUBS	Starting character position for keyword of operand	16	Up to 99
CONCARD	Whether a control card is to be produced	YES	NO (see Note 3)
DDNAME	Default library name	'GENLIB'	'name' (see Note 4)
INITVAL (in DGPSB only)	Whether VALUE operands are to be generated on VIRFLO statements	NO	YES
KNOWNAS	Whether local-names from KNOWN-AS clauses are to be generated instead of member names	NO	YES (see Note 2)
LENMENT	Maximum length of main entry	71	Up to 99
LIBCC	The format of the control card output as the first record of a QSAM FILE (unless overridden by 'control card' on an ONTO clause)	(See the *ASG-Manager Products Source Language Generation* publication)	A delimited character string of 1 to 72 characters including a question mark (?)
MEMLEN	Maximum length of library-name	8	Up to 16
RNDBIN	Rounding of binary items	YES	NO
RNDBIT	Whether bit string fields are to be generated with byte alignment	NO	YES

Notes

1. The standard Source Language Generation facility output character set for Database Description (DBD) and Program Specification Block (PSB) control statements conforms to that defined for COBOL for the data division. This character set can be extended to allow non-standard characters to be output in names, by entering the hexadecimal value of each required character as a value to ACHAR. The user should ensure that any extra characters that are added to the output character set in this way are used only in ways that are permitted by the software with which DataManager is used.

2. If both ALIAS=YES and KNOWNAS=YES apply, then when a data name is generated for a member that has an ALIAS clause and is subject to a containing member's KNOWN-AS clause, the KNOWN-AS local-name takes precedence.

3. When the value CONCARD=NO is used, to suppress the generation of a control card, the production of BKEND cards is also suppressed.

4. 'name' is a valid name which is not more than 32 characters in length. It must be presented within quotes and must be different from all other name values specified or used by default for the same macro.

The Macros DGSCOB, DGSPLI, DGSBAL, and DGSREC

The purpose and applicability of these macros are defined in "Generation of COBOL, PL/I, or Assembler Data Description Statements for Segment Input/Output Areas" on page 140. This table lists the keywords of these macros for which values can be specified when DataManager is installed:.

The Macros DGSCOB, DGSPLI, DGSBAL, and DGSREC: Keywords Specifiable on Installation

Keyword	Specifies	Default Value	Alternative Values
CONKEY	In a logical child segment: the name to be applied to the destination parent's concatenated key. The value specified by CONKEY is only used when no CONCATENATED-KEY-NAME clause has been specified in the SEGMENT member's data definition	'CONCAT-KEY'	'name'
CONSTNT	The name to be applied to the CONSTANT field of an index pointer segment	'CONSTANT'	'name'
DUPDATA	The name to be applied to the duplicate data fields of an index pointer segment	'DUP-DATA-FLD'	'name'
SIZE	The name to be applied to the SIZE field of a variable length segment	'SIZE-FIELD'	'name'

The Macros DGSCOB, DGSPLI, DGSBAL, and DGSREC: Keywords Specifiable on Installation

Keyword	Specifies	Default Value	Alternative Values
SUBSEQ	The name to be applied to the subsequence fields of an index pointer segment	'SIZE-FIELD'	'name'
SUBSEQ	The name to be applied to the subsequence fields of an index pointer segment	'SUBSEQUENCE-FIELD'	'name'
USERDAT	The name to be applied to the user-date field of logical child and index pointer segments	'USER-DATA'	'name'

Note.

1. 'name' is a valid name that is not more than 32 characters in length. It must be presented within quotes and must be different from all other name values specified or used by default for the same macro.

Index

A

alignment 32
arrays
    FIELD control statements for 134

B

BACKWARD-LOGICAL-TWIN pointer 29
BOUND interrogation keyword 109

C

COBOL SYNCHRONIZED keyword 32
common clauses 22, 28, 50, 56, 58, 62, 66, 71, 75, 79, 84
CONCATENATED 29
concatenated 29
concatenated key 29
    index source segment 31
    index target segment 31
    internal member type 29
    name 30
    sensitive segments in PCB 138
CONCATENATED-KEY-CONSTITUENTS interrogation keyword 109
CONCATENATED-KEY-FIELDS clause 31
CONCATENATED-KEY-NAME 29
CONCATENATED-KEY-NAME clause 29
CONCATENATED-KEY-NAMES interrogation keyword 109
CONTAINS interrogation keyword 106
conventions page vi
crossing logical relationships 76

D

data description statement generation for segment I/O areas 87
data set overflow 63
database
    loading (processing option) 85
    primary index 22
    reading processing option 85
    updating processing option 85
database definition
    OUTPUT clause 59
Database Description (DBD) Control Statements 22
    for LOGICAL database 75
    for primary index database 71
DATABASES interrogation keyword 109
DATASET control statements 134
DATASETS clause 58
DBD FIELD Control Statements 52
destination parent segment 29
DEVICE clause 59
DL/I- DATASETS interrogation keyword 106
duplicate data fields across segments 33
DUPLICATE-DATA-FIELDS interrogation keyword 108
DUPLICATE-DATA-FIELDS clause 51

E

edit/compression routine for segment 36
EDIT-COMPRESSION-EXITS interrogation keyword 110
exit, user 36

F

FATHERS interrogation keyword 107
floating point items 50
FORWARD-LOGICAL-TWIN pointer 29
FREQUENCY-FREE-BLOCKS clause 67

G

GENERATES interrogation keyword 108, 110
GENERATES-FIELDS keyword in PRODUCE command 133
GENLIB output file 136
GSAM database
    input data set 58
    interrogation of 122

H

HDAM database 67
- ACCESS clause 67
- ANCHOR-POINTS clause 67
- definition 67
- INSERTION-BYTES-MAXIMUM clause 67
- RANDOMIZING-MODULE clause 67
- RELATIVE-BLOCK-MAXIMUM clause 67
- root segment of 35

HELD-AS form of ITEM 51

I

IN-DATABASES interrogation keyword 122
installation macros 141

L

LCHILD control statements 134
logical concatenated segment 76

M

MAINTENANCE-EXITS interrogation keyword 108
MARK IV file definition forms 131

N

NO-ASSEMBLY-PRINT keyword 138

O

OF interrogation keyword 110
ON interrogation keyword 108
output source library data set 136

P

PARENTS interrogation keyword 107
path calls 145
physical database
- interrogation of 109

primary index database 135
PRODUCE command
- ALL-FIELDS keyword 133
- AS clause 135
- control options 131
- DIRECT-FIELDS keyword 133
- FROM clause 133
- PRIMARY-INDEX clause 135
- SEARCH-FIELDS keyword 133
- USE clause 136
- USING clause 136

Program Communication Block
- interrogation of 117
- SEGMENT clause 138

PSB control statements
- generation 137
- library-names 136
- PROCSEQ operand 138

PSBGEN control statement operands 138

Q

QUALIFIED-ON interrogation keyword 111

R

RENAMES interrogation keyword 110

S

SEARCH-KEY-FIELDS interrogation keyword 108
secondary indexing, fields for 133
SECONDARY- SEQUENCE-ON interrogation keyword 111
SEGM control statements 134
SEGMENT interrogation keyword 111
SENSEG control statements 138
SENSITIVE-FIELDS interrogation keyword 111
SEQUENCE-KEY interrogation keyword 108
SEQUENCE-KEY-CONSTITUENTS interrogation keyword 110
SOURCE interrogation keyword 109
SSAS interrogation keyword 111
SUBSEQUENCE-FIELDS interrogation keyword 109

T

TARGET interrogation keyword 109
TO interrogation keyword 110

U

user data 133