

# **ASG-DataManager™ IMS (DL/I) Interface**

Version 2.5

Publication Number: DMR0200-25-IMS

Publication Date: December 2000

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.

© 1998-2002 Allen Systems Group, Inc. All rights reserved.

All names and products contained herein are the trademarks or registered trademarks of their respective holders.



---

# Contents

---

<b>Preface</b> .....	<b>v</b>
<b>About this Publication</b> .....	<b>vi</b>
<b>Publication Conventions</b> .....	<b>vi</b>
<b>ASG Customer Support</b> .....	<b>ix</b>
Intelligent Support Portal (ISP) .....	ix
Telephone Support .....	ix
<b>ASG Documentation/Product Enhancement Fax Form</b> .....	<b>xi</b>
<b>1 DataManager IMS (DL/I) Interface Facilities</b> .....	<b>1</b>
<b>2 The IMS (DL/I) Environment and DataManager</b> .....	<b>5</b>
<b>Introduction</b> .....	<b>5</b>
Segments .....	5
Databases .....	12
Application View .....	13
<b>Further Information</b> .....	<b>15</b>
Segments .....	15
IMS (DL/I) Data Fields .....	16
IMS (DL/I) Databases .....	16
Special DataManager Member Types .....	17
Application View .....	19
<b>3 Member Types</b> .....	<b>21</b>
<b>Member-type Syntax for IMS (DL/I) Segments</b> .....	<b>22</b>
Physical Segments .....	24
Logical Segments .....	51
Segments that Reside in a Secondary Index Database .....	55
<b>Member-type Syntax For IMS (DL/I) Databases</b> .....	<b>69</b>
Outline of the IMS-DATABASE Member Type .....	69
Member Type of a GSAM Type IMS (DL/I) Database Syntax .....	70
The Member Type for a HSAM Type IMS (DL/I) Database .....	75

The Member Type for a HISAM Type IMS (DL/I) Database . . . . .	79
The Member Type for a HDAM Type IMS (DL/I) Database . . . . .	86
The Member Type for a HIDAM Type IMS (DL/I) Database . . . . .	94
The Member Type for a LOGICAL Type IMS (DL/I) Database . . . . .	106
The Member Type for a SECONDARY-INDEX Type IMS (DL/I) Database . . . . .	111
<b>Member-type Descriptions for IMS (DL/I) Program Communication Blocks . . .</b>	<b>117</b>
PROGRAM-COMMUNICATION-BLOCK . . . . .	117
Example of a GSAM type PCB . . . . .	129
Examples of OUTPUT-MESSAGE Type PCBs . . . . .	129
Examples of STRUCTURE Type PCBs . . . . .	130
<b>The PROCESSES Clause . . . . .</b>	<b>132</b>
Syntax of the PROCESSES Clause . . . . .	132
<b>4 Extensions to DataManager Commands for IMS (DL/I) Databases . . . . .</b>	<b>139</b>
<b>Introduction . . . . .</b>	<b>139</b>
<b>IMS (DL/I) Member-type Keywords . . . . .</b>	<b>139</b>
<b>Condition Keywords for WHICH and WHAT Commands . . . . .</b>	<b>141</b>
Examples . . . . .	142
Member Type Interrogations . . . . .	146
Interrogation Syntax . . . . .	154
Alternative Verb Keywords . . . . .	174
<b>5 IMS (DL/I) Source Language Generation . . . . .</b>	<b>175</b>
<b>Introduction . . . . .</b>	<b>176</b>
<b>Generating IMS (DL/I) DBD Control Statements . . . . .</b>	<b>176</b>
<b>Generating IMS (DL/I) PSB Control Statements . . . . .</b>	<b>183</b>
<b>Generation of COBOL, PL/I, or Assembler Data Description Statements for Segment Input/Output Areas . . . . .</b>	<b>189</b>
The PRODUCE Command . . . . .	189
Installation Macros . . . . .	189
Segment Input/Output Areas: Items Defined as BINARY or BITS . . . . .	190
Simple Physical Segments . . . . .	190
Logical Child Segments . . . . .	190
Destination Parent Segments . . . . .	191
Index Target and Index Source Segments . . . . .	191
Logical Segments and Logical Concatenated Segments . . . . .	192
Variable Length Segments . . . . .	192
Path Calls . . . . .	194
Index Pointer Segments . . . . .	194
Miscellaneous IMS (DL/I) Fields . . . . .	198

Generation of COBOL, PL/I, or Assembler Data Description Statements for Segment Sensitive Fields Input/Output Areas .....	198
Generation of COBOL, PL/I, or Assembler Data Description Statements for PCB Masks .....	200
Generation of COBOL, PL/I, or Assembler Data Description Statements for Segment Search Arguments.....	203
<b>Appendix A</b>	
<b>Macros for Tailoring the IMS Interface.....</b>	<b>209</b>
Implementation of the IMS (DL/I) Interface Macros .....	209
The Macros DGDBD And DGPSB .....	210
The Macros DGSCOB, DGSPLI, DGSBAL, and DGSREC.....	213
<b>Appendix B</b>	
<b>Manager Products and IMS Keywords .....</b>	<b>215</b>
Introduction.....	215
IMS Databases.....	215
<b>Index.....</b>	<b>217</b>



---

## Preface

---

This *ASG-DataManager IMS (DL/I) Interface* describes the OS version of the IMS (DL/I) Interface facility. This facility (additional to the basic set-up, maintenance, and interrogation features) enables the user to fully define IMS (DL/I) databases in the dictionary and to produce IMS (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 OS version of this interface encompasses the Data Language/I (DL/I) facility of the IMS/VS subsystem available under VS (excluding DOS/VS).

The DOS version of the interface is described in a separate manual.

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 IMS \(DL/I\) Interface Facilities," on page 1](#) summarizes the interfaces between DataManager and IMS (DL/I).
- [Chapter 2, "The IMS \(DL/I\) Environment and DataManager,"](#) discusses very briefly the concept of IMS (DL/I) databases and illustrates how an IMS (DL/I) database can be defined to DataManager.
- [Chapter 3, "Member Types,"](#) gives the specifications of the DataManager data definition statements for IMS (DL/I) databases and their constituents.
- [Chapter 4, "Extensions to DataManager Commands for IMS \(DL/I\) Databases,"](#) describes the interrogation and documentation facilities for reporting on IMS (DL/I) databases.
- [Chapter 5, "IMS \(DL/I\) Source Language Generation,"](#) describes the interface between IMS (DL/I) and the DataManager Source Language Generation facility.

## Publication Conventions

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

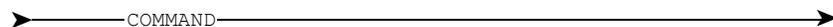
Diagrams are read from left to right along a continuous line (the "main path"). Keywords and variables appear on, above, or below the main path.

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:

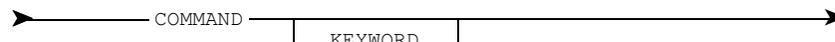


A required keyword appears on the main path:

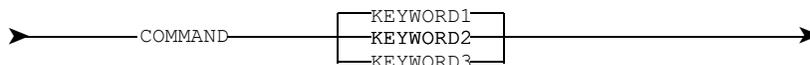
**Convention Represents**



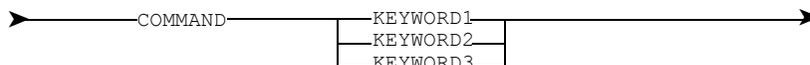
An optional keyword appears below the main path:



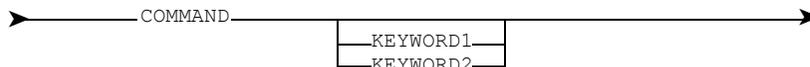
Where there is a choice of required keywords, the keywords appear in a vertical list; one of them is on the main path:



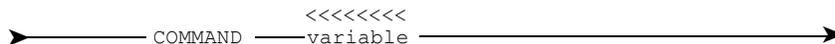
or



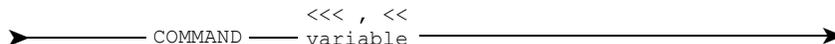
Where there is a choice of optional keywords, the keywords appear in a vertical list, below the main path:



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:



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:



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:



# ASG Customer Support

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.

**ASG Third-party Support.** 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.

## Intelligent Support Portal (ISP)

Online product support is available at: <http://www.asg.com/support/support.asp> via the ASG Intelligent Support Portal (ISP). Your logon information for ISP online support is:

Customer ID = NNNNNNNNNN

Password = XXXXXXXXXXXX

where:

NNNNNNNNNN is your customer ID supplied by ASG Product Distribution.

XXXXXXXXXX is your unique password supplied by ASG Product Distribution.

The *ASG-Intelligent Support Portal User's Guide* provides instructions on how to use the ISP and is located on the ASG Support web page.

## Telephone Support

To expedite response time, 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 as displayed
- A description of the specific steps that immediately preceded the problem
- Verify whether you received an ASG Service Pack or cumulative service tape for this product. It may include information to help you resolve questions regarding installation of this ASG product. The Service Pack instructions are in a text file on the distribution media included with the Service Pack. You can access the latest software corrections and Service Packs via the ISP.
- 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.)

### 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

**Business Hours Support**

<b>Your Location</b>	<b>Phone</b>	<b>Fax</b>	<b>E-mail</b>
<b>United States and Canada</b>	800.354.3578 or 800.775.5675	703.464.4901	support@asg.com
<b>Australia</b>	00.800 3544 3578 or 61.3.9645.8500	61.2.9460.0280	support.au@asg.com
<b>England</b>	00.800 3544 3578 or 44.1727.736305	44.1727.812018	support.uk@asg.com
<b>France</b>	00.800 3544 3578 or 33.141.028590	33.141.028589	support.fr@asg.com
<b>Germany</b>	00.800 3544 3578 or 49.89.45716.200	49.89.45716.400	support.de@asg.com
<b>Italy</b>	00.800 3544 3578 or 39.0290.4500.25		support.it@asg.com
<b>Singapore</b>	00.800 3544 3578 or 65.332.2922	65.337.7228	support.sg@asg.com
<b>South Africa</b>	00.800 3544 3578 or 00.800.201.423		support.sa@asg.com
<b>All other countries:</b>	1.239.435.2201		support@asg.com

**Non-Business Hours - Emergency Support**

<b>Your Location</b>	<b>Phone</b>	<b>Your Location</b>	<b>Phone</b>
<b>United States and Canada</b>	800.354.3578	<b>Netherlands</b>	00.800 3354 3578
<b>Asia</b>	00.800 3544 3578	<b>New Zealand</b>	00.800 3354 3578
<b>Australia</b>	00.800.3354 3578	<b>Singapore</b>	00.800 3354 3578
<b>Denmark</b>	00.800 3544 3578	<b>South Korea</b>	00.800 3354 3578
<b>France</b>	00.800.3354 3578	<b>Sweden</b>	00.800 3354 3578
<b>Germany</b>	00.800.3354 3578	<b>Switzerland</b>	00.800 3354 3578
<b>Hong Kong</b>	00.800 3544 3578	<b>Thailand</b>	00.800 3354 3578
<b>Ireland</b>	00.800 3544 3578	<b>United Kingdom</b>	00.800 3354 3578
<b>Israel</b>	00.800 3544 3578	<b>All other countries</b>	1.239.435.2201 or 1.602.667.2800
<b>Japan</b>	00.800 3544 3578		

If you receive a voice mail message, follow the instructions to report a production-down or critical problem. 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. Please have available the information described previously when the ASG Support representative contacts you.

## ASG Documentation/Product Enhancement Fax Form

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

If you do not have access to the web, FAX your suggestions to product management at (239) 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.

Company Name	Telephone Number	Site ID	Contact name

Product Name/Publication	Product Version # ( <i>required</i> )	Publication Date
<b>Product:</b>		
<b>Publication # (<i>required</i>):</b>		
<b>Tape VOLSER:</b>		

Enhancement Request:



---

# 1

## DataManager IMS (DL/I) Interface Facilities

---

DataManager's IMS (DL/I) Interface provides facilities for all users in an IMS (DL/I) environment. It enables users:

- To define IMS (DL/I) databases and segments to DataManager [in a simpler manner than that available from the use of IMS (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
- To 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
- To 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
- To generate from the data dictionary and to insert into the appropriate source library complete sets of PSB Control Statements to allow a PSB generation process
- To generate record layouts and/or COBOL, PL/I, or Assembler data descriptions for segment input/output areas
- To generate record layouts and/or COBOL, PL/I, or Assembler data descriptions for segment input/output areas for sensitive fields
- To generate record layouts and/or COBOL, PL/I, or Assembler data descriptions for Program Communication Block (PCB) masks
- To generate record layouts and/or COBOL, PL/I, or Assembler data descriptions for segment search arguments (SSAs)

The ability to define IMS (DL/I) databases, segments, and PCBs demands three additional member types in DataManager:

- To define a database, the member type is IMS-DATABASE or DL/I-DATABASE. The member type identifier IMS-DATABASE is synonymous with 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 PCB, the member type is PROGRAM-COMMUNICATION-BLOCK or PCB, which comes between the MODULE member type and the IMS-DATABASE/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 IMS-DATABASE/DL/I-DATABASE member type and the GROUP member type in the DataManager member type hierarchy.

The IMS-DATABASE/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 IMS \(DL/I\) Environment and DataManager," on page 5](#) and are specified in [Chapter 3, "Member Types," 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 IMS \(DL/I\) Environment and DataManager," on page 5](#) and are specified in [Chapter 3, "Member Types," on page 21](#).

To enable the definitions of IMS (DL/I) databases, PCBs, and segments to be processed by DataManager in the same way as other members of the data dictionary, the keywords IMS-DATABASES, 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, DL1-DATABASES, and DLI-DATABASES are accepted for the keyword DL/I-DATABASES.

Also added to these commands are the keywords:

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

to enable interrogation and documentation in respect of members of internal member types. These members are generated by DataManager (see "[Special DataManager Member Types](#)" on page 17). Since 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 IMS (DL/I) database systems. These facilities are specified in [Chapter 4, "Extensions to DataManager Commands for IMS \(DL/I\) Databases,"](#) on page 139.

The ability to generate IMS (DL/I) control statements, data descriptions for segment input/output areas, PCB masks, and segment search arguments require 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 publication *ASG-Manager Products Source Language Generation*.

Enhancements to the Source Language Generation facility that enable it to output IMS (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, "IMS \(DL/I\) Source Language Generation,"](#) on page 175.

For an installation that is implementing an IMS (DL/I) database management system for the first time, ASG strongly recommends the following approach:

- Study, in depth, the concepts and facilities both of IMS and of DataManager.
- Design the IMS 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.
- Similarly ADD program and module data definitions and PCB members for the application views.
- Using the ASG-Manager Products Source Language Generation facility, generate the IMS (DL/I) control statements and the data descriptions for segment input/output areas, PCB masks, and segment search arguments.

Users should find that this approach is easier and offers more in-built automatic checks on accuracy than implementation using IMS (DL/I) facilities alone.

---

# 2

## The IMS (DL/I) Environment and DataManager

---

This chapter includes these sections:

<b>Introduction</b> . . . . .	<b>5</b>
Segments . . . . .	5
Databases. . . . .	12
Application View . . . . .	13
<b>Further Information</b> . . . . .	<b>15</b>
Segments . . . . .	15
IMS (DL/I) Data Fields. . . . .	16
IMS (DL/I) Databases. . . . .	16
Special DataManager Member Types. . . . .	17
Application View . . . . .	19

### Introduction

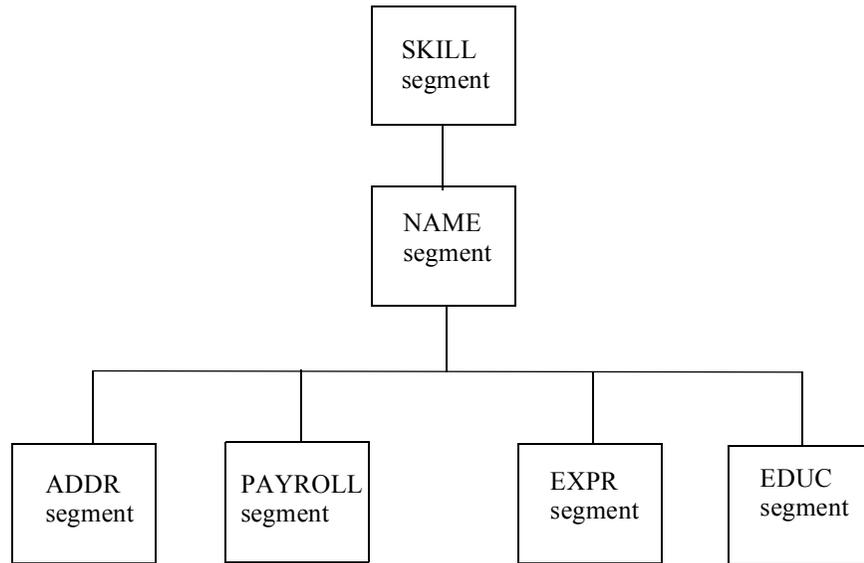
#### Segments

One of the fundamental concepts of IMS 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 an IMS (DL/I) environment is the segment. Regardless of where or how segments are physically stored, an IMS (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.

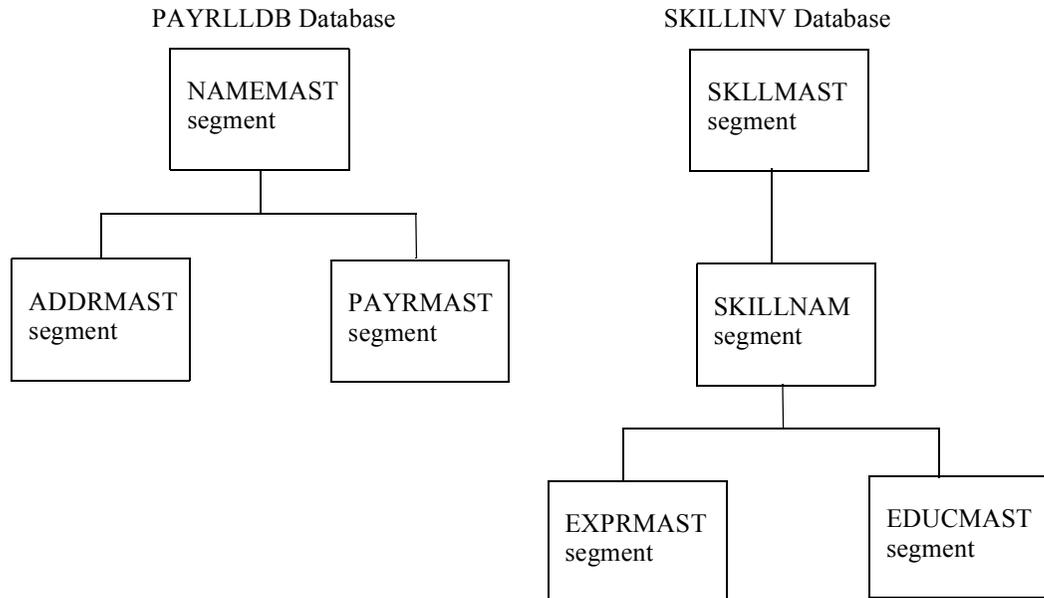
[Figure 1](#) illustrates this concept of a logical data structure for an employee database named SKILLEMP.

**Figure 1 • Logical Structure of an Employee Database, SKILLEMP**



However, the six segments in [Figure 1 on page 6](#) 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 IMS (DL/I) Interface, each of the segments shown in [Figure 1 on page 6](#) 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 the following would be the method of using DataManager data definition statements to define these segments:

- For the segments in [Figure 1 on page 6](#):

```

ADD SKILL;
SEGMENT LOGICAL
CONTAINS 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;
SEGMENT LOGICAL
CONTAINS EDUCMAST
;
```

- For the segments in [Figure 2 on page 7](#):

```
ADD NAMEMAST;
SEGMENT PHYSICAL
RELATED-AS DESTINATION-PARENT
ATTRIBUTES
  CONTAINS INITIAL, SURNAME, SEX
  FREQUENCY 100
  SEQUENCE-KEY SURNAME DUPLICATED
  INSERT-POSITION LAST
;
ADD ADDRMAST;
SEGMENT 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
  FREQUENCY 10
  SEQUENCE-KEY SKLLCODE UNIQUELY
;
ADD SKILLNAM;
SEGMENT PHYSICAL
RELATED-AS UNIDIRECTIONAL-CHILD TO NAMEMAST
  POINTERS SYMBOLIC
ATTRIBUTES
  SEQUENCE-KEY EMPLOYEE-NO UNIQUELY
  CONTAINS EMPLOYEE-NO
;
ADD EXPRMAST;
```

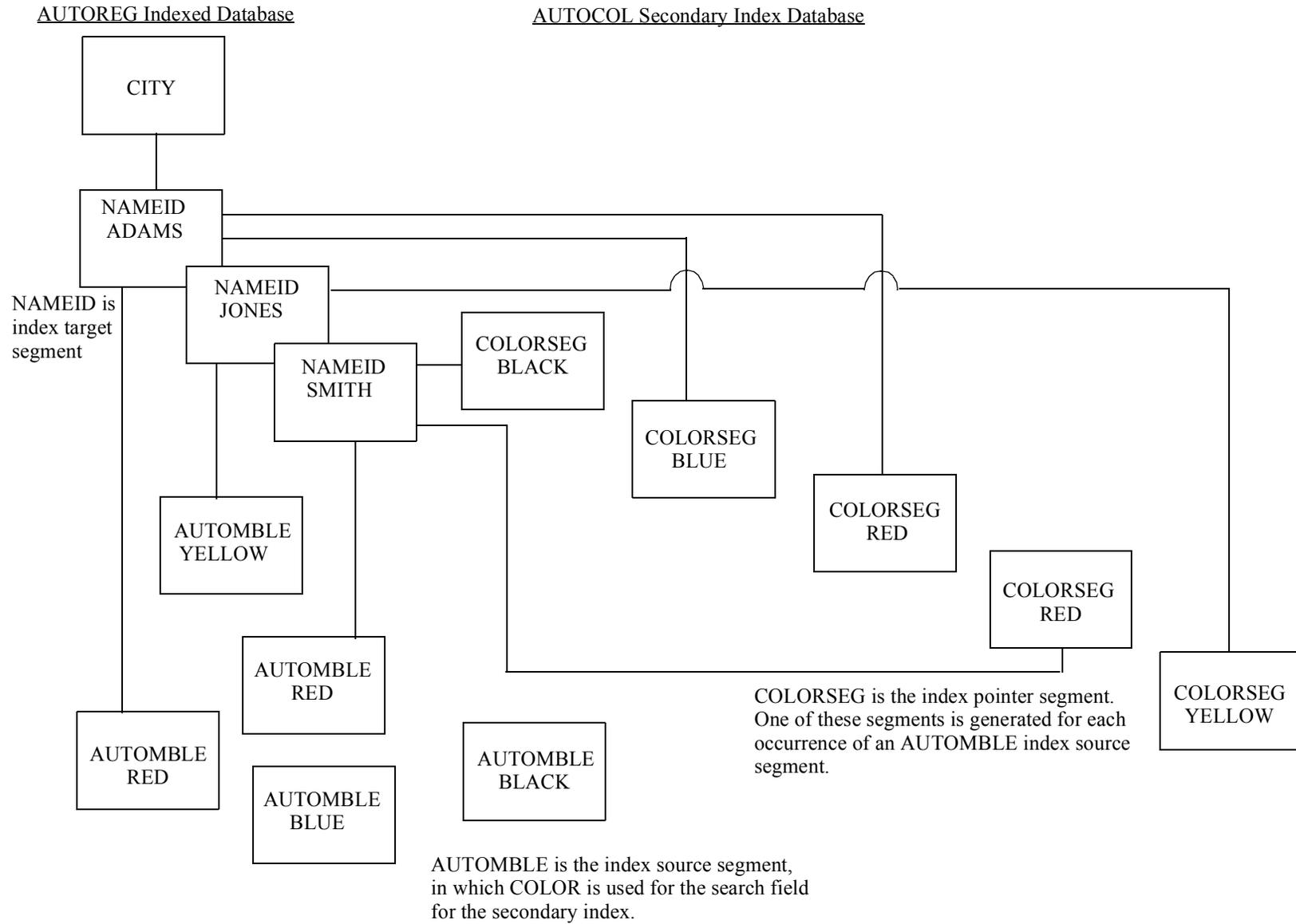
```
SEGMENT PHYSICAL
ATTRIBUTES
  CONTAINS EXPCODE,EXPRTIME
  INSERT-POSITION FIRST
;
ADD EDUCMAST;
SEGMENT PHYSICAL
ATTRIBUTES
  CONTAINS QUALCODE
SEQUENCE-KEY QUALCODE UNIQUELY
;
```

The secondary indexing facility of IMS(DL/I) enables users 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 10](#) illustrates the concept of secondary indexing for an automobile register database.

Using the DataManager IMS (DL/I) Interface, each of the segments shown in [Figure 3 on page 10](#) can be defined as a data dictionary SEGMENT type member.

Figure 3 • An Example of Secondary Indexing



If certain assumptions are made regarding the specific attributes of the segments, then the following 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
  POINTERS FORWARD-HIERARCHICAL
;
ADD NAMEID;
SEGMENT PHYSICAL
RELATED-AS TARGET-SEGMENT
ATTRIBUTES
  CONTAINS INITIAL, SURNAME, IDENTCDE
  SEQUENCE-KEY IDENTCDE UNIQUELY
  POINTERS FORWARD-HIERARCHICAL
;
ADD AUTOMBLE;
SEGMENT PHYSICAL
RELATED-AS SOURCE-SEGMENT
ATTRIBUTES
  CONTAINS MODEL, COLOR, WEIGHT
  INSERT-POSITION LAST
  POINTERS FORWARD-HIERARCHICAL
;
ADD COLORSEG;
SEGMENT INDEX-POINTER
RELATED-TO NAMEID ON COLCODE
  POINTERS SYMBOLIC
  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 IMS (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.

IMS (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 IMS-DATABASE (or 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 6](#), [Figure 2 on page 7](#), and [Figure 3 on page 10](#), respectively, then the following would be the method of using DataManager data definition statements to define those databases:

- For the database in [Figure 1 on page 6](#):

```
ADD SKILLEMP;  
INS-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 7](#):

```
ADD PAYRLLDB;  
IMS-DATABASE HISAM  
ACCESS ISAM  
DATASETS PRIME PAYRF BLOCK 4 RECORD 256  
          OVERFLOW PAYRFO BLOCK 4 RECORD 256  
          DEVICE 3340  
CONTAINS NAMEMAST,  
  ADDRMAST PARENT NAMEMAST,  
  PAYRMAST PARENT NAMEMAST
```

```
;
ADD SKILLINV;
IMS-DATABASE HISAM
ACCESS ISAM
DATASETS PRIME SKLLF BLOCK 8 RECORD 512
          OVERFLOW SKLLFO BLOCK 8 RECORD 512
          DEVICE 3340
CONTAINS SKLLMAST,
          SKILLNAM PARENT SKLLMAST,
          EXPRMAST PARENT SKILLNAM,
          EDUCMAST PARENT SKILLNAM
```

- For the databases in [Figure 3 on page 10](#):

```
ADD AUTOREG;
IMS-DATABASE HDAM
ACCESS VSAM RANDOMIZING-MODULE AUTRTNE
          ANCHOR-POINTS 1
          RELATIVE-BLOCK-MAXIMUM 500
          INSERTION-BYTES-MAXIMUM 824
DATASETS PRIME AUTOF BUFFER 1648
          DEVICE 2314
          SCAN 5
CONTAINS CITY,
          NAMEID PARENT CITY,
          AUTOMBLE PARENT NAMEID
;
ADD AUTOCOL;
IMS-DATABASE SECONDARY-INDEX
ACCESS VSAM
DATASETS PRIME COLORF BUFFER 1024
          OVERFLOW COLORFO BUFFER 1024
          DEVICE 2314
CONTAINS COLORSEG
;
```

## Application View

Finally, when specifying an IMS (DL/I) database system, the applications view of the databases and segments that they access must be defined. This must be done before an IMS (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 IMS (DL/I) Interface language facilities:

**PROGRAM-COMMUNICATION-BLOCK or PCB member type.** A member of this type defines a PCB accessed by an application program.

**PROCESSES clause that 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:

- 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 can be used by the Source Language Generation Facility when generating DBD control statements [see ["Application View" on page 19](#) and ["Generating IMS \(DL/I\) DBD Control Statements" on page 176](#)].

Generating SSAs and PCB masks is described in ["Generation of COBOL, PL/I, or Assembler Data Description Statements for PCB Masks" on page 200](#) and ["Generation of COBOL, PL/I, or Assembler Data Description Statements for Segment Search Arguments" on page 203](#), respectively.

If certain assumptions are made, then the following 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 6](#) and [Figure 2 on page 7](#), respectively:

```
PROCESSES  IMS
CONTAINS  SKILLEMP-PCB, AUTOREG-PCB
          SEGMENT-SEARCH-ARGUMENTS
          SEGMENT SKILL USED-IN SKILL-SSA
                                COMMAND-CODE FIRST-OCCURRENCE
                                QUALIFIED-ON SKLLTYPE EQ
          SEGMENT EXPR USED-IN EXPR-SSA
                                QUALIFIED-ON EXPRCODE EQ
                                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 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:

- 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 where the respective constituent fields reflect the same total length for the intersection data, because when IMS (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 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 ["The Member Type for a HIDAM Type IMS \(DL/I\) Database" on page 94](#) and ["Generating IMS \(DL/I\) DBD Control Statements" on page 176](#)], 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.

### **IMS (DL/I) Data Fields**

As stated in ["Segments" on page 5](#), 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.

### **IMS (DL/I) Databases**

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

However, an IMS-DATABASE or DL/I-DATABASE data definition may be used for Source Language Generation to produce, for example, DBD or 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 IMS (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 ["The Member Type for a HIDAM Type IMS \(DL/I\) Database" on page 94](#) and ["Generating IMS \(DL/I\) DBD Control Statements" on page 176](#)], then they are created by suffixing I to the respective names for the HIDAM database.

### **Special DataManager Member Types**

For the IMS (DL/I) Interface, DataManager automatically and maintains members of special internal types. These internal member types are for:

- Sequence key fields
- Datasets
- Index search fields
- 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
- 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
- An IMS (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 IMS-DATABASE (DL/I-DATABASE) data definition statements can include the names and definitions of the databases' constituent datasets. 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, then so are any of the internal members that were created for its constituent datasets, 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
- Fields from which IMS (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 slash (/) that must be the first character of the name is added by DataManager when the Source Language Generation Facility is used to produce 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, 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 ["The Member Type for a HISAM Type IMS \(DL/I\) Database" on page 79](#)]. 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 IMS (DL/I) Interface allows the member types described above 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 IMS \(DL/I\) Databases," on page 139](#)]. 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. For further information, see [Chapter 5, "IMS \(DL/I\) Source Language Generation," on page 175](#).

## **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 PSB control statements for an application, the Source Language Generation Facility produces a 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 IMS (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 IMS (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 IMS (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 that 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, through 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 DBD control statements for a database, the user can specify in the PRODUCE command whether DataManager is to generate IMS (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 the following actions must be taken to ensure that DataManager will recognize these 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

## Member Types

---

This chapter includes these sections:

<b>Introduction</b> . . . . .	<b>22</b>
<b>Member-type Syntax for IMS (DL/I) Segments</b> . . . . .	<b>22</b>
Physical Segments . . . . .	24
Logical Segments . . . . .	51
Segments that Reside in a Secondary Index Database . . . . .	55
<b>Member-type Syntax For IMS (DL/I) Databases</b> . . . . .	<b>69</b>
Outline of the IMS-DATABASE Member Type . . . . .	69
Member Type of a GSAM Type IMS (DL/I) Database Syntax . . . . .	70
The Member Type for a HSAM Type IMS (DL/I) Database . . . . .	75
The Member Type for a HISAM Type IMS (DL/I) Database . . . . .	79
The Member Type for a HDAM Type IMS (DL/I) Database . . . . .	86
The Member Type for a HIDAM Type IMS (DL/I) Database . . . . .	94
The Member Type for a LOGICAL Type IMS (DL/I) Database . . . . .	106
The Member Type for a SECONDARY-INDEX Type IMS (DL/I) Database . . . . .	111
<b>Member-type Descriptions for IMS (DL/I) Program Communication Blocks</b>	<b>117</b>
PROGRAM-COMMUNICATION-BLOCK . . . . .	117
Example of a GSAM type PCB . . . . .	129
Examples of OUTPUT-MESSAGE Type PCBs . . . . .	129
Examples of STRUCTURE Type PCBs . . . . .	130
<b>The PROCESSES Clause</b> . . . . .	<b>132</b>
Syntax of the PROCESSES Clause . . . . .	132

## Introduction

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

**Segment.** See ["Member-type Syntax for IMS \(DL/I\) Segments" on page 22](#) for more information on the SEGMENT member type.

**Database.** See ["Member-type Syntax For IMS \(DL/I\) Databases" on page 69](#) for more information on the member types IMS-DATABASE and DL/I-DATABASE. (Any of the alternative forms DL/1-DATABASE, DLI-DATABASE, or DL1-DATABASE are accepted for the member type identifier DL/I-DATABASE.)

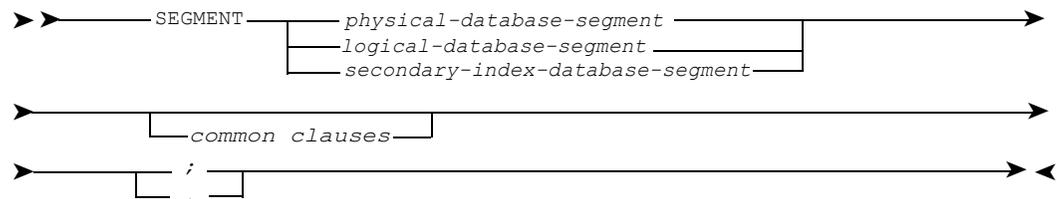
**Program Communication Block.** See ["Member-type Descriptions for IMS \(DL/I\) Program Communication Blocks" on page 117](#) for more information on the member type PROGRAM-COMMUNICATION-BLOCK (or PCB), which is used to specify the application view of a database and the segments that the application 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 ["The PROCESSES Clause" on page 132](#)).

## Member-type Syntax for IMS (DL/I) Segments

IMS (DL/I) provides a comprehensive selection of keywords and operands in its SEGMENT member type in order to define all possible attributes and relationships of segments that can reside in several fundamentally different types of database.

This is the overall outline format of the SEGMENT member type:



where:

*physical-database-segment* is the definition for the type of segment that resides in a physical database (see ["Physical Segments" on page 24](#)).

*logical-database-segment* is a logical view of physical segments or of concatenated physical segments (see ["Logical Segments" on page 51](#)).

*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 ["Segments that Reside in a Secondary Index Database" on page 55](#)).

*common clauses* are as defined in the *ASG-Manager Products Dictionary/Repository User's Guide*.

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 Manager Products 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 ["The Member Type for a HIDAM Type IMS \(DL/I\) Database" on page 94](#)] or in the Source Language Generation Facility's PRODUCE command [see ["Generating IMS \(DL/I\) DBD Control Statements" on page 176](#)]. 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:

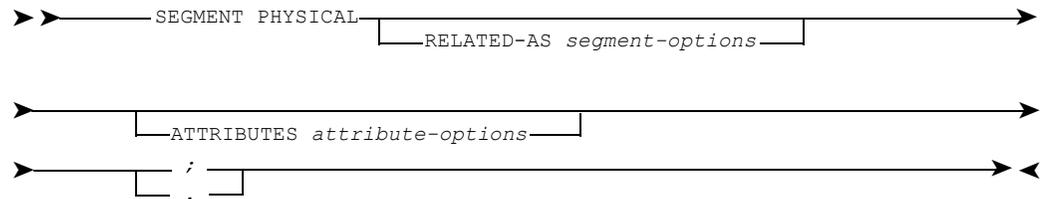
- A segment type keyword.
- A RELATED-AS clause (for a physical-database-segment) or a RELATED-TO clause (for a secondary-index-database-segment), to define the logical and secondary indexing relationships in which the segment participates. There is no RELATED clause for a logical-database-segment definition.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

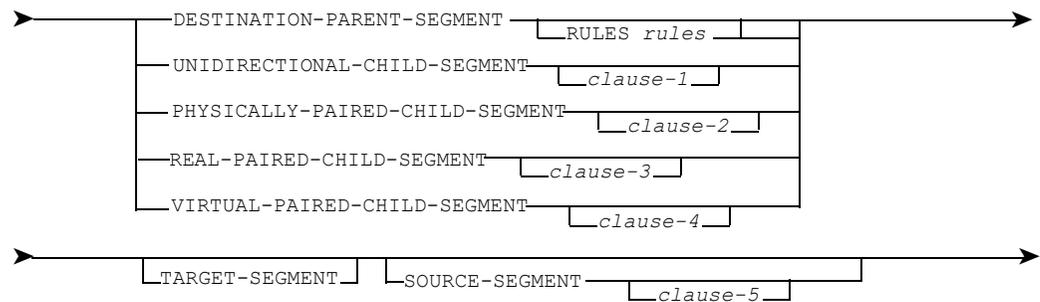
## Physical Segments

### Syntax



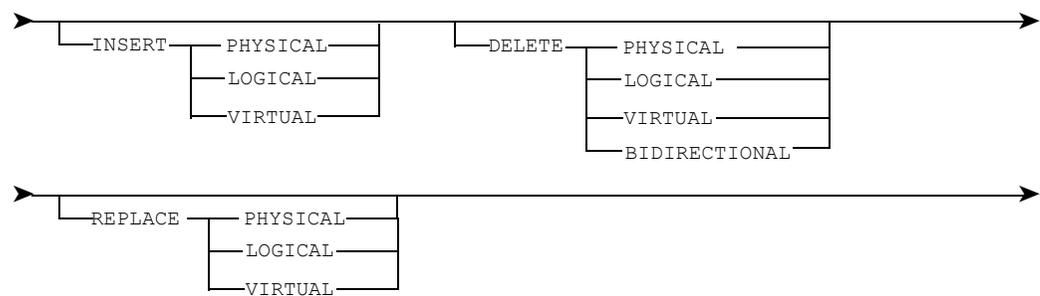
where:

*segment-options* are:



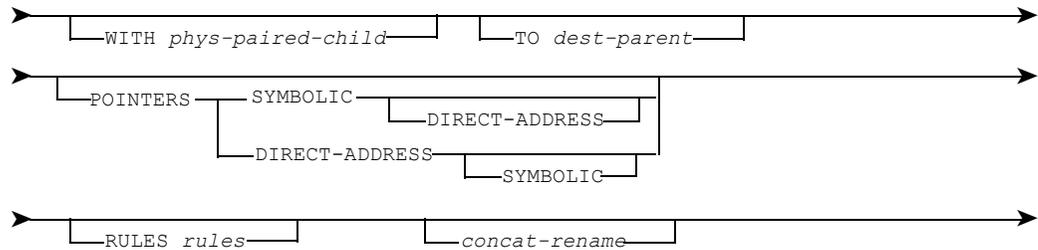
where:

*rules* are:





*clause-2* is:

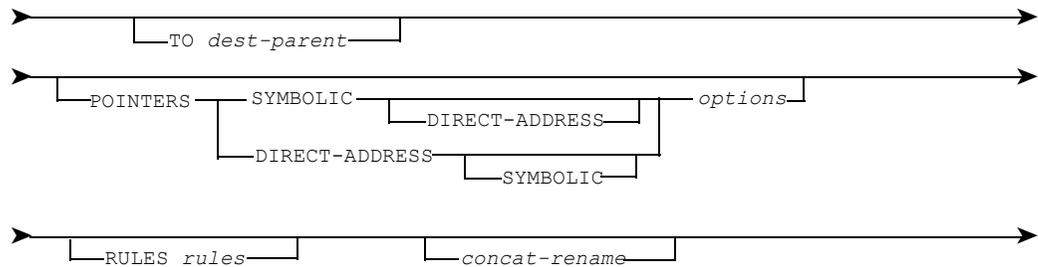


where:

*phys-paired-child* is the name of a PHYSICALLY-PAIRED-CHILD-SEGMENT member.

*dest-parent*, *rules*, *concat-rename* are as defined above.

*clause-3* is:



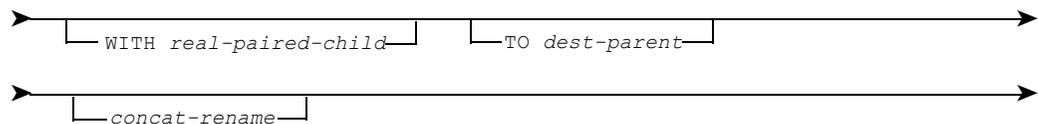
where:

*dest-parent*, *concat-rename*, *phys-paired-child*, and *rules* are as defined above.

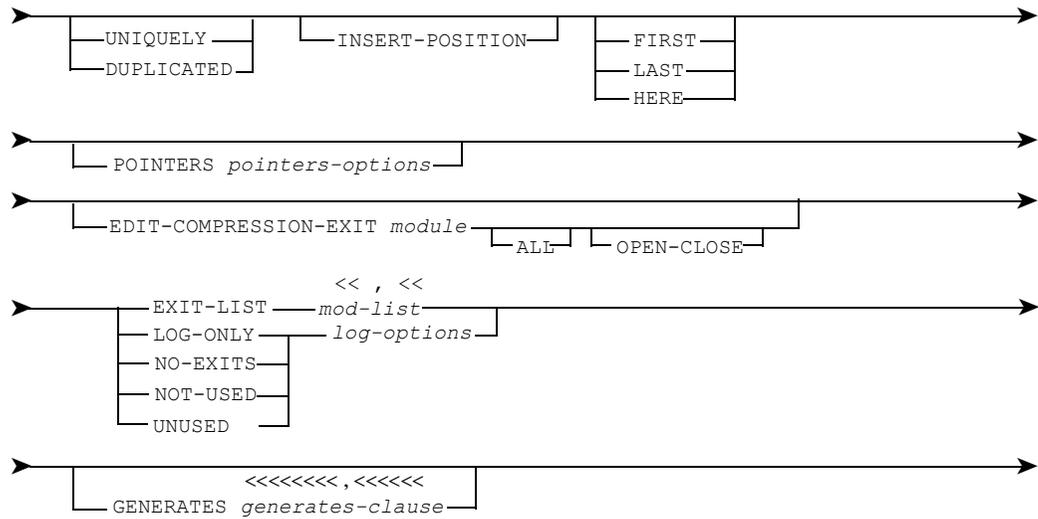
*options* are:



*clause-4* is:

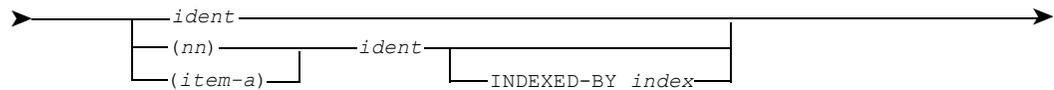




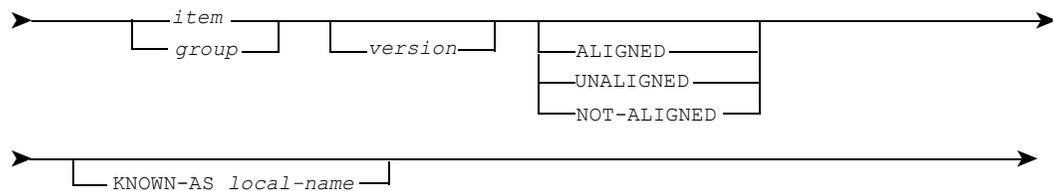


where:

*content* is:



where *ident* is:



where:

*item*, *group* are as defined above.

*version* is an unsigned integer in the range 1 to 15.

*local-name* is as defined above.

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

*item-a* is the name of an ITEM member.

*index* is a name, conforming to the rules for member names.

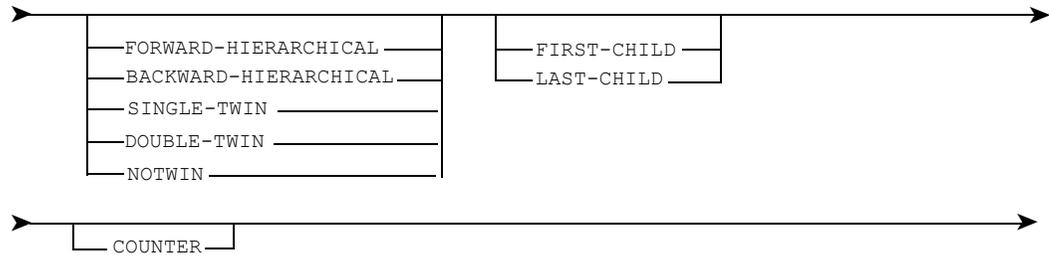


*name* is the name of a CONCATENATED-KEY member.

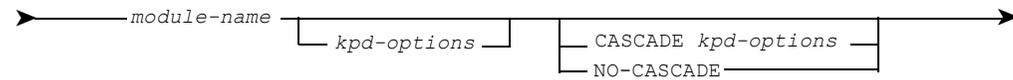
*module* is the name of a MODULE member.

*freq*, *group*, *item*, *key*, and *module* are as defined above.

*pointers-options* are:



*mod-list* is:



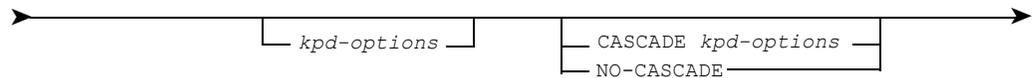
where:

*module-name* is the name of a MODULE or PROGRAM member.

*kpd-options* are:

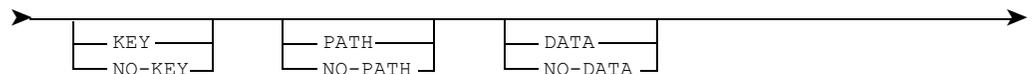


*log-options* are:



where:

*kpd-options* are:





- 3 If the segment participates in a logical relationship, then one of the following 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.
  - UNIDIRECTIONAL-CHILD-SEGMENT, which specifies that the segment being defined is a logical child segment in a unidirectional logical relationship.
  - PHYSICALLY-PAIRED-CHILD-SEGMENT, which specifies that the segment being defined is a logical child segment in a physically paired 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 For a PHYSICALLY-PAIRED-CHILD-SEGMENT or a VIRTUAL-PAIRED-CHILD-SEGMENT, the WITH clause specifies the logical child segment with which the segment is physically or virtually paired respectively. In either case, the paired segment must be a physical child of *destination-parent-name*. A WITH clause is not specified for a REAL-PAIRED-CHILD-SEGMENT.
  - The POINTERS clause specifies the type of pointer that connects a logical child segment and its logical parent segment. It is invalid for a virtual logical child segment. The clause can contain the keyword SYMBOLIC and/or the keyword DIRECT-ADDRESS.
  
- 6 The POINTERS clause specifies the type of pointer that connects a logical child segment and its logical parent segment. It is invalid for a virtual logical child segment. The clause can contain the keyword SYMBOLIC and/or the keyword DIRECT-ADDRESS.

- 7** SYMBOLIC specifies that the symbolic pointer to the logical parent segment (the logical parent's concatenated key) is stored as part of the logical child segment on the storage device. It must be specified if:
- The logical child segment being defined is sequenced on its physical twin chain through the use of any part of the logical parent's concatenated key.
  - Or the sequence key field of the logical child being defined consists of any part of the logical parent's concatenated key.

If SYMBOLIC is not specified and the logical parent resides in a HISAM database, SYMBOLIC is assumed.

- 8** DIRECT-ADDRESS specifies that a 4-byte logical parent pointer field is reserved in the prefix of the logical child segment being defined. This keyword can be specified if the logical parent segment resides in a HDAM or HIDAM database. If it is not specified and the logical parent resides in a HDAM or HIDAM database, DIRECT-ADDRESS is assumed. DIRECT-ADDRESS is invalid if the logical parent segment resides in a HISAM database.
- 9** For a REAL-PAIRED-CHILD-SEGMENT, 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.
- 10** For a REAL-PAIRED-CHILD-SEGMENT, 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.
- 11** The RULES clause specifies the rules for inserting, deleting, and replacing a segment.
- 12** The rules DELETE BIDIRECTIONAL are valid only for a segment that is the physical parent segment of a real logical child segment in a virtually paired logical relationship.
- 13** The default of LOGICAL is assumed for any rule that is omitted or invalidly specified.

- 14** 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 16 on page 34](#)). 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.
- 15** Interrogations can be performed on the concatenated key internal member type (see ["Condition Keywords for WHICH and WHAT Commands" on page 141](#)). However, meaningful results will only be obtained in response to interrogations concerning members that use the internal member type, as the member type has no entries in the uses table.
- 16** 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. When the Source Language Generation Facility encounters a member of the concat-name type, the concatenated key is constructed, and it is output in a form appropriate to the language or record layouts being generated.
- 17** 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 the destination parent segment. The rules governing a local name are as defined in the syntax section above.
- 18** A segment cannot be a logical child segment and a destination parent segment.
- 19** A segment cannot be a logical child segment if it is the root segment of a database.
- 20** A segment and its physical child segment cannot both be logical child segments.
- 21** Only one logical child segment in a physically paired logical relationship can have physical child segments.
- 22** A virtual logical child segment cannot have physical child segments.
- 23** IMS (DL/I) automatically reserves a 4-byte counter field in the prefix of logical parent segments if they are not connected to any of their logical child segments by logical child pointers. This is generated regardless of whether or not COUNTER is specified.

- 24** 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.
- 25** The keyword SOURCE-SEGMENT specifies that the segment being defined is an index source segment.
- 26** 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<sub>xxxxxx</sub>, 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.
- 27** 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. The member 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 ["Interrogation Syntax" on page 154](#)). The Source Language Generation Facility can operate on members of this type.
- 28** The UNIQUE-KEY-FIELDS clause defines any number of system related fields of the type that prompts IMS (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<sub>xxxxxx</sub> 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.

- 29** When a source segment definition that contains a UNIQUE-KEY-FIELDS 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.
- 30** The UNIQUE-KEY-FIELDS clause is valid only if the segment being defined resides in a HDAM or HIDAM database.
- 31** A segment cannot be an index source segment and a logical child segment.
- 32** The ATTRIBUTES clause must be present if the segment is to be completely defined.
- 33** 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.
- 34** ALIGNED is the equivalent of COBOL SYNCHRONIZED or PL/I ALIGNED. It means that (subject to [remark 38 on page 37](#)) 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 (see "[Segment Input/Output Areas: Items Defined as BINARY or BITS](#)" on page 190). The way in which this is achieved is dependent upon the language being generated, and this is described for COBOL, PL/I, and Assembler in the publication *ASG-Manager Products Source Language Generation*.

- 35** UNALIGNED means that (subject to [remark 38 on page 37](#)) 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.)
- 36** 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.
- 37** 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 nonalignment of such indirectly referenced items.
- 38** 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 42 on page 38](#) to [remark 46 on page 39](#)). 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 37 on page 37](#)).
- 39** 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.

- 40** The entries in the CONTAINS clause must include, directly or indirectly, references to the following fields, if they are applicable to the segment being defined:
- The sequence key fields (for a logical child segment, this applies only if contained in the intersection data)
  - 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
- 41** 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.
- 42** Entries in the CONTAINS clause may be conditional (IF clauses, see [remark 44 on page 38](#)) and/or may have alternative content declarations (ELSE clauses, see [remark 43 on page 38](#)), 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.
- 43** 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.

- 44** Any content declaration can be specified as conditional; that is, as applying 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.
- 45** 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.

- 46** 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).
- 47** The `FREQUENCY` clause states the expected frequency of the segment being defined. If the segment is the root segment in a HIDAM database, the frequency entered will be applied to its corresponding primary index pointer segment when generated. For root segments, the frequency entered must be an integer. The `FREQUENCY` clause is invalid for a virtual logical child segment.
- 48** The `SEQUENCE-KEY` clause specifies the field that is the sequence key of the segment being defined; or 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.
- 49** 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`.
- 50** 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.
- 51** 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.
- 52** Use the `WITH` clause if a logical child segment is being defined, to enable 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.

- 53** Each GROUP/ITEM listed in the WITH clause, must be the name of:
- 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.

- 54** 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.
- 55** 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.
- 56** If 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.
- 57** 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 ["Interrogation Syntax" on page 154](#)). The Source Language Generation Facility can operate on members of this type.
- 58** UNIQUELY (the default) indicates that only unique values are allowed in the sequence key field being defined. DUPLICATED indicates that duplicate values are allowed. All of the sequence keys for a virtual logical child segment must be uniformly defined as either UNIQUELY or DUPLICATED.
- 59** 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.
- 60** 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.

- 61** If the segment is an index target segment and symbolic pointing is used to point to the index target segment from the index pointer segment, then a unique sequence key field must be specified for the segment and for each of the segments on which it depends.
- 62** If a SEQUENCE-KEY clause is specified for a segment that resides in an HDAM or HIDAM database, then hierarchical or twin pointers must be specified (see [remark 66 on page 41](#) to [remark 70 on page 42](#)).
- 63** You must not specify a SEQUENCE-KEY clause and a NOTWIN pointer.
- 64** 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.
- 65** 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.
- 66** 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.
- 67** FORWARD-HIERARCHICAL specifies that a 4-byte hierarchical forward pointer field is reserved in the prefix of the segment.

- 68** BACKWARD-HIERARCHICAL specifies that a 4-byte hierarchical forward pointer field and a 4-byte hierarchical backward pointer field are reserved in the prefix of the segment.
- 69** SINGLE-TWIN specifies that a 4-byte physical twin forward pointer field is reserved in the prefix of the segment.
- 70** 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.
- 71** NOTWIN specifies that no space is to be reserved in the prefix of the segment for a physical twin forward pointer field. NOTWIN can be specified:
- For the root segment of a HIDAM database
  - For a dependent segment of a HIDAM or HDAM database if:
    - Its physical parent segment does not have hierarchical pointers specified
    - No more than one occurrence of the dependent segment will be stored as a physical child of any occurrence of its physical parent segment
- 72** NOTWIN is invalid if:
- A SEQUENCE-KEY clause has been specified for this segment and the segment is a dependent segment, or if hierarchical pointers have been specified for the segment's physical parent.
  - The segment is a real paired logical child segment and a SEQUENCE-KEY clause has been specified for its virtually paired logical child segment.
- 73** 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.
- 74** 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.

It should be noted that if a physical parent segment and its physical child segment appear in different dataset groups, then they must be connected by physical child and physical twin pointers.

- 75** COUNTER specifies that a 4-byte counter field is to be reserved in the segment prefix. This is in anticipation of establishing this segment as a logical parent without logical child pointers, when IMS maintains an internal count of logical children pointing to this logical parent to manage delete operations. Where this segment is already established as a logical parent, IMS will determine the need for a counter internally and it need not be specified explicitly. COUNTER should be used where the need for a logical parent can be anticipated before the segment actually becomes a logical parent in order that the requirement for database reorganization may be avoided.
- 76** 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 a database that does not use the VSAM operating system access method, or is a virtual logical child segment.
- 77** ALL specifies that the user exit routine can condense or modify any of the fields in the segment. If ALL is omitted, then only the data fields that do not change the position of the sequence key field relative to the start of the segment can be condensed or modified.
- 78** The ALL keyword is invalid if the segment is the root segment of a HISAM database.
- 79** OPEN-CLOSE specifies that initialization and termination processing control is required by the segment edit routine; that is, the edit/compression routine will gain control after database open and after database close.
- 80** 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, additional to those fields required by IMS (DL/I) for which the Source Language Generation Facility always provides DBD FIELD control statements (see ["Generating IMS \(DL/I\) DBD Control Statements" on page 176](#)). 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.
- 81** 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.

**82** The facility [described in "[Generating IMS \(DL/I\) DBD Control Statements](#)" on [page 176](#)], which automatically generates DBD FIELD control statements for the fields described below, cannot be used when fields are duplicated across segments, as it is assumed that there is no such duplication. Instead, the GENERATES clause must be used if it is required to generate DBD FIELD control statements for the following 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.

**83** The length of the segment is not specified in the segment definition, as it is automatically calculated when required.

**84** 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.

- 85** 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
  - For a virtual logical child segment, any fields preceding its sequence key fields in the real logical child segment with which it is paired
  - For a source segment, any fields constituting the search field, subsequence fields or duplicate data fields of its corresponding index pointer segment; and any fields preceding those fields
- 86** 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 four.
- 87** 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 192](#)).
- 88** A variable length segment is defined by specifying that the segment contains, directly or indirectly, a variable length ITEM member.
- 89** 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.

- 90** Common clauses can be present in any type of data definition statement; therefore, they are 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.
- 91** 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.
- 92** When an ATTRIBUTES clause followed by a FREQUENCY clause is encoded, for a segment that is not a virtual logical child segment, it assumes that the FREQUENCY clause is subordinate to the ATTRIBUTES clause, specifying the expected frequency of the segment being defined.  
  
If you need to specify the FREQUENCY common clause following an ATTRIBUTES clause, you should thus specify another common clause before the FREQUENCY common clause. This allows Manager Products to recognize that the clauses that follow are all common clauses.
- 93** 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.
- 94** Within the ATTRIBUTES clause, the subordinate clauses can be in any order (see [remark 33 on page 36](#)). The optional keywords in the POINTERS clause can be in either order.
- 95** A SEGMENT PHYSICAL can be contained by any number of physical databases provided that it does not participate in a logical or a secondary indexing relationship; that is, it does not have a RELATED-AS clause in its definition.

- 96** 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 where the 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-COMPRESS-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
- 97** 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.
- 98** The *local-name* specified in the RENAME clause must conform to the rules for member names stated in the *ASG-ControlManager User's Guide*. This 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 Source Language Generation Facility. The *local-name* is not separately recorded in the repository (that is, 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 repository. The local-name is the name by which the member is known only within the segment defined by this data definition.

**99** In the CONTAINS clause:

- *Version* specifies 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 41 on page 38](#). The default assumed (or if the stated version does not exist) is the lowest numbered existing version.
- *Item-a* is an array declaration that declares that when the segment here defined is processed by an application program or module, *item-a* contains the number of times item or group occurs in the array.

**100** The index specified in the INDEXED-BY clause is to be used as the index name when COBOL data descriptions are generated. The index name is not separately recorded in the repository (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 repository.

**101** Up to 15 conditional terms can be specified in the IF clause. A conditional term compares the contents of an item with a comparand; it has three elements: item name, operator, and comparand. If there are two or more conditional terms, they must be separated by an AND or OR keyword; they are evaluated from left to right in a Boolean logical manner.

- *version-b* specifies the version (within the HELD-AS form or within a defaulted form as stated in [remark 41 on page 38](#)) of *item-b* that is relevant to the comparison. If *version-b* is omitted, a default value of 1 is assumed.
- *version-c* specifies the version (within the HELD-AS form or within a defaulted form as stated in [remark 41 on page 38](#)) of *item-c* whose contents 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 *item-b*'s *form-description* (and *contents-description*, if *item-b* contains a CONTENTS clause). *literal* can be either a character string of up to 256 printable and/or non printable characters, enclosed in quotes, or 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 ITEM member-type documentation in the *ASG-Manager Products Dictionary/Repository User's Guide*) not enclosed in quotes

- 102** The CHANGED-DATA-CAPTURE-FACILITY clause is only valid for segments contained within HDAM, HIDAM, HISAM, or SHISAM databases. If a MODULE or PROGRAM is specified alone in the list (i.e. without any keywords before the next entry or end of list), then during PRODUCE IMS for the database default values of KEY, NOPATH, DATA, and CASCADE will be generated and a warning message issued. Up to 16 entries can be specified in the EXIT-LIST clause.
- 103** LOG-ONLY or NO-EXITS may be specified to enable generation of the DBDGEN DBD or SEGM specification of

```
EXIT- ( (* , LOG ) ,
```

This may be required if data changes are to be written only to the IMS log without any exit processing. Where exits are specified, the default specification of NOLOG will be generated.

### 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 7](#) and [Figure 3 on page 10](#). Also in those examples are segments participating in a unidirectional logical relationship and an index target segment.

[Figure 4 on page 51](#) illustrates two physical data structures that contain segments participating in a virtually paired logical relationship.

In [Figure 4 on page 51](#):

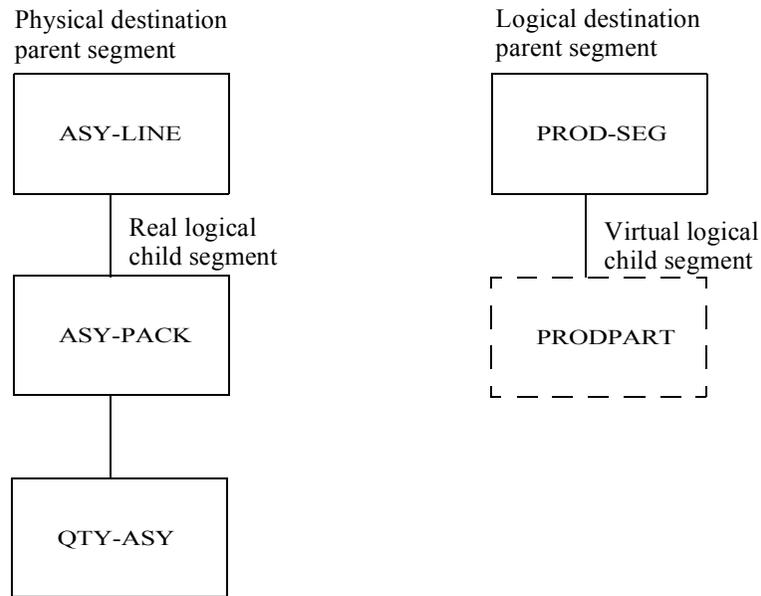
- ASY-LINE is the physical segment for an assembly line that assembles packs of assembly parts to make a product.
- ASY-PACK is the physical segment for a pack of assembly parts being assembled on that assembly line.
- QTY-ASY is the physical segment for the number of those packs of assembly parts assembled 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 4 on page 51](#). 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
;
```

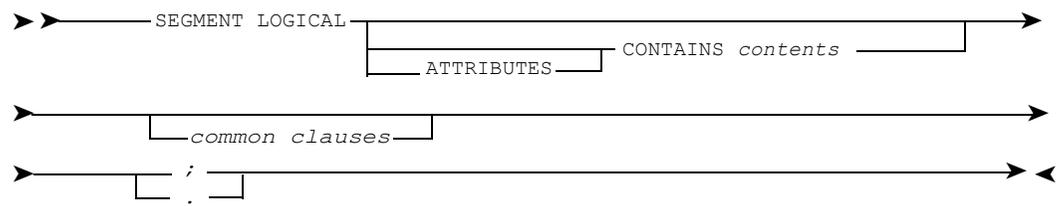
For examples of logical data structures that can be defined from the virtually paired logical relationship illustrated above, see [Figure 5 on page 55](#) and the accompanying narrative.

**Figure 4 • Example of Physical Data Structures With Segments Participating in a Virtually Paired Logical Relationship**



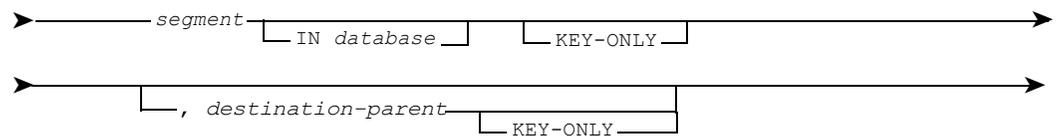
## Logical Segments

### Syntax



where:

*contents* are:



*segment* is the name of a PHYSICAL SEGMENT.

*database* is the name of a HISAM, HDAM, or HIDAM database.

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

*common-clauses* are as defined in the *ASG-Manager Products Dictionary/Repository User's Guide*.

### **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 HISAM, 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 IMS (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 IMS (DL/I) DBD control statements are produced, Manager Products finds an appropriate physical database in one of the ways described in ["The Member Type for a LOGICAL Type IMS \(DL/I\) Database" on page 106](#).
- 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 and the KEY-ONLY keyword are also assumed. If RXLOG01 is specified as YES by the macro DGDBD, then this processing is not undertaken, so that a SEGM statement is generated with a SOURCE operand for the logical child alone.

- 7** The KEY-ONLY keyword specifies that the concatenated key (if any) of the physical segment is to be placed in the key feedback area of the logical segment's PCB; and, that the physical segment is not to be placed in the user input/output area when a call is issued to retrieve the logical segment. If KEY-ONLY is omitted, the concatenated key of the physical segment is placed in the key feedback area, and the physical segment is placed in the user input/output area.
- 8** The sequence key for a concatenated segment is the sequence key of the logical child segment.
- 9** Common clauses can be present in any type of data definition statement; therefore, they are 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.
- 10** The common clauses can be in any order. If common clauses are present, they must follow the ATTRIBUTES clause, if it is present.
- 11** 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 where the 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 5 on page 55](#) illustrates logical data structures that can be defined from the physical data structures illustrated by [Figure 4 on page 51](#).

In [Figure 5 on page 55](#), 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 5 on page 55](#), 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 5 on page 55](#), example B:

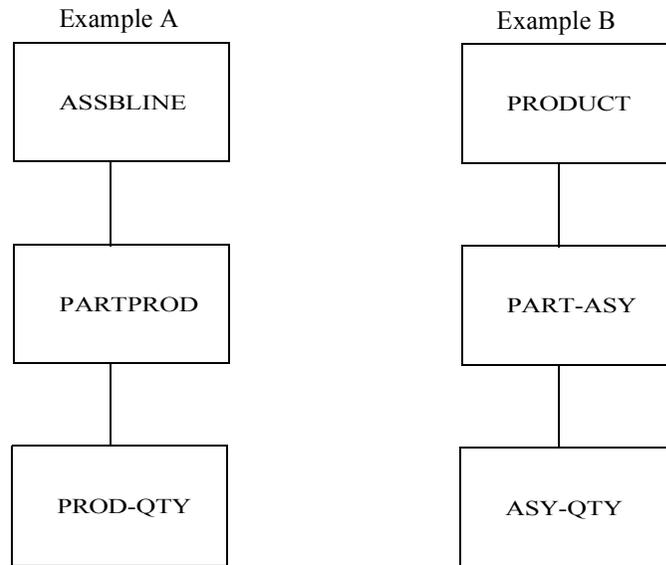
PRODUCT is a logical segment representing a product.

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.

These are examples of the data definition statements that could define the segments illustrated in [Figure 5 on page 55](#), example B:

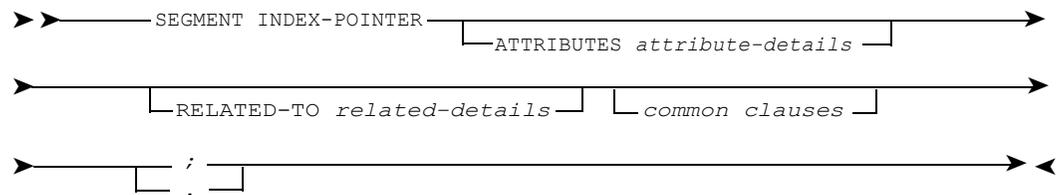
```
ADO PRODUCT;  
SEGMENT LOGICAL  
CONTAINS PROD-SEG  
;  
ADD PART-ASY;  
SEGMENT LOGICAL  
CONTAINS PRODPART, ASY-LINE  
;  
ADD ASY-QTY  
SEGMENT LOGICAL  
CONTAINS QTY-ASY  
;
```

Figure 5 • Examples of Logical Data Structures



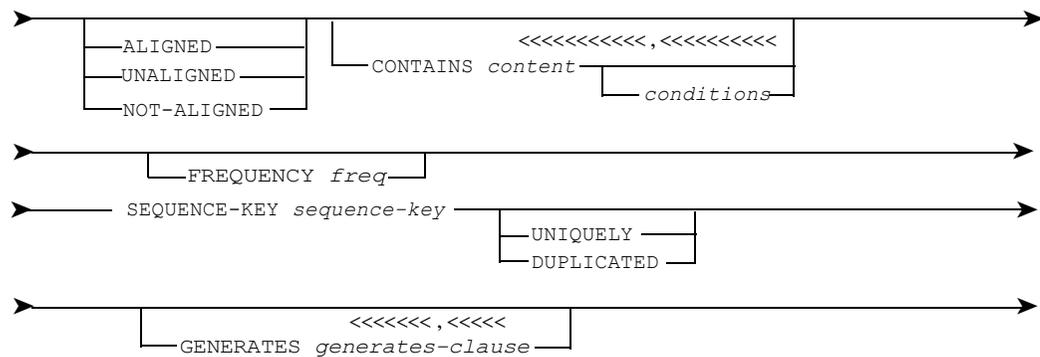
**Segments that Reside in a Secondary Index Database**

**Syntax**



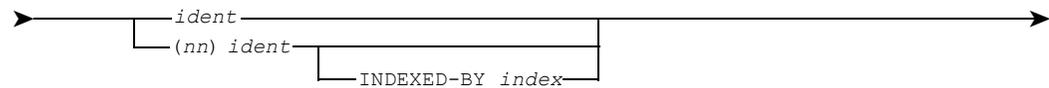
where:

*attribute-details* are:

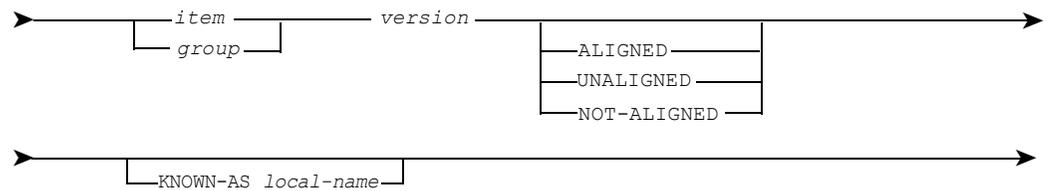


where:

*content* is:



where *ident* is:



where:

*item* is the name of an ITEM repository member.

*group* is the name of a GROUP repository.

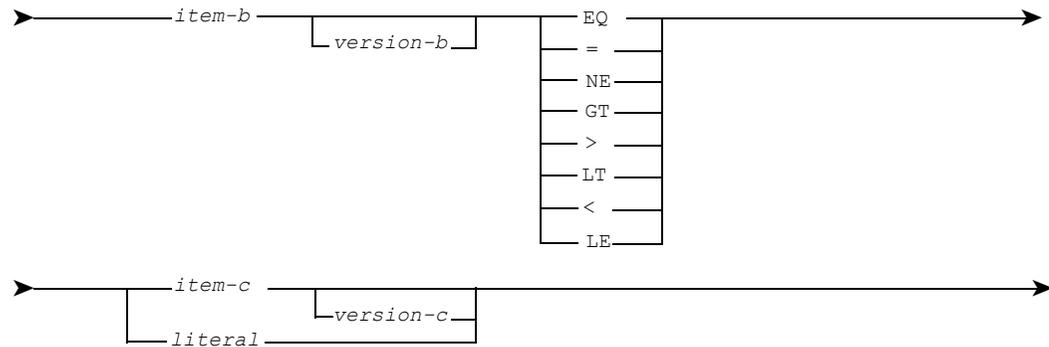
*version* is an unsigned integer in the range 1 to 15.

*local-name* is a name, conforming to the rules for member names stated in the *ASG-ControlManager User's Guide*.

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

*index* is a name, conforming to the rules for member names, that is to be used as the index name when COBOL data descriptions are generated by the Source Language Generation Facility.

*conditions* are:



where:

*literal* is a literal comparand.

*item-b* is the name of the ITEM where the contents are to be compared with the comparand.

*version-b* is an unsigned integer in the range 1 to 15.

*item-c* is the name of the ITEM where the contents are the comparand.

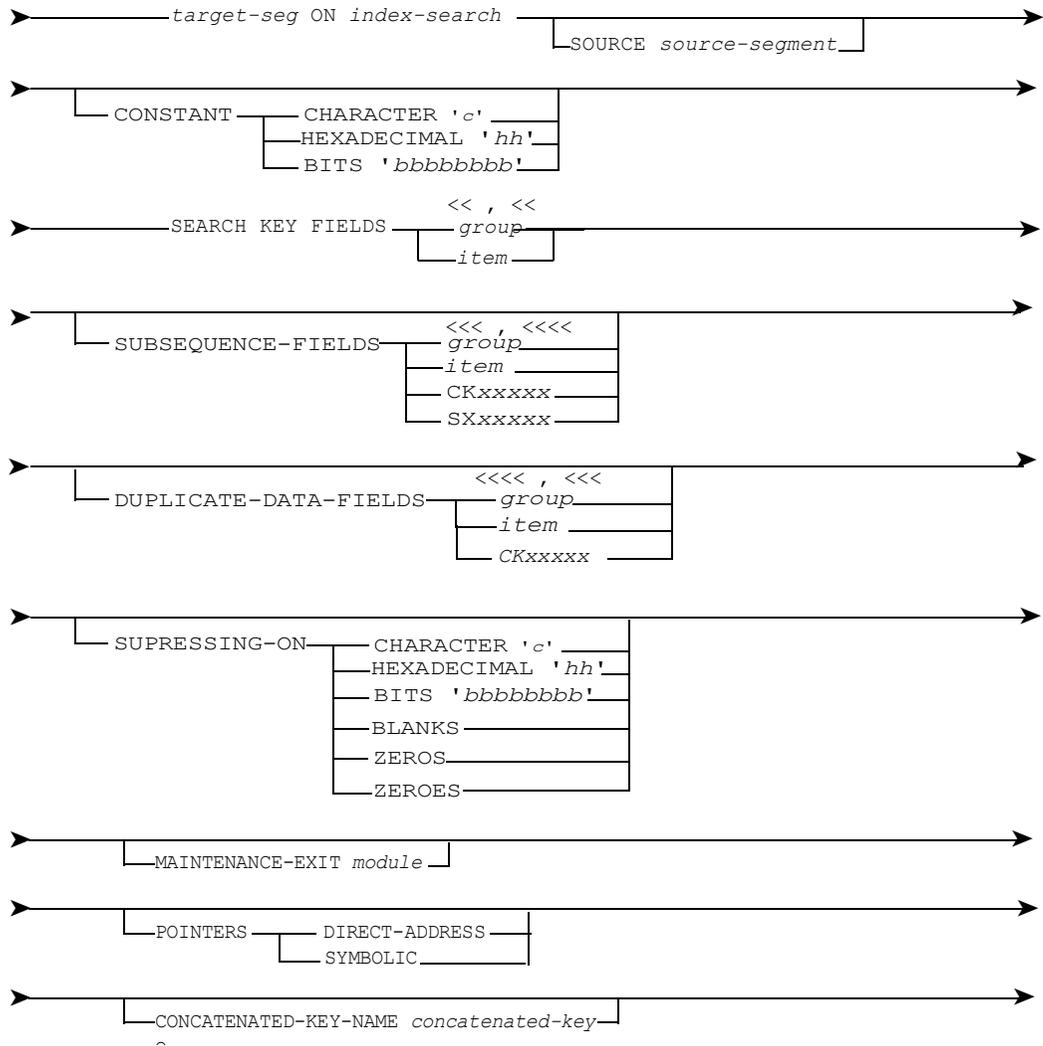
*version-c* is an unsigned integer in the range 1 to 15.

*content* is as defined above.

*freq* is an unsigned integer in the range 1 to 16777215.

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

related-details are:



where:

*target-seg* is the name of a SEGMENT that is a PHYSICAL TARGET-SEGMENT member.

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

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

*module* is the name of a MODULE member.

*group* and *item* are as defined above.



- 4** ALIGNED is the equivalent of COBOL SYNCHRONIZED, or PL/I ALIGNED. It means that (subject to [remark 8 on page 60](#)) 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 (see "[Segment Input/Output Areas: Items Defined as BINARY or BITS](#)" on page 190). 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 60](#)) 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 nonalignment of such indirectly-referenced item.
- 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 61](#) to [remark 15 on page 62](#)). It is not meaningful to include either of these keywords in a content declaration that declares a group (see [remark 7 on page 60](#)).

- 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 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 61](#)) and/or may have alternative content declarations (ELSE clauses, see [remark 12 on page 61](#)), 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 of 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; that is, as applying 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 FREQUENCY clause specifies the expected frequency of the segment.
- 17** 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 user table entries:
- An entry for the *index-search-field-name* (XDFLD) when specified for the segment (see [remark 25 on page 64](#))
  - 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. When the Source Language Generation process is performed on such members, the CONSTANT field will also be generated if it has been specified.

- 18** 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.
- 19** 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 further in ["Generating IMS \(DL/I\) DBD Control Statements" on page 176.](#)) 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.
- 20** 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.

- 21** 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; that is, the search, subsequence and duplicate-data fields, and fields constituting the concatenated key of the index target segment, if present (see [remark 35 on page 65](#) and [remark 36 on page 65](#)). Normally, DBD FIELD control statements are only generated for the sequence key field and for fields in the user data (see "[Generating IMS \(DL/I\) DBD Control Statements](#)" on page 176).

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.

- 22** 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 through 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 IMS \(DL/I\) DBD Control Statements](#)" on [page 176](#)), 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 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 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.

- 23** 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

- 24 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.
- 25 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.

- 26 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.
- 27 The CONSTANT clause specifies a character identifies every index pointer segment created for this secondary index. It is required if this secondary index resides in an index database shared by other secondary indexes.
- 28 The SEARCH-KEY-FIELDS clause lists the names of one to five 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.
- 29 The SUBSEQUENCE-FIELDS clause lists the names of one to five 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.
- 30 The combined length of the fields declared by CONSTANT, SEARCH-KEY-FIELDS, and SUBSEQUENCE-FIELDS must not exceed 240 bytes.
- 31 The DUPLICATE-DATA-FIELDS clause lists the names of one to five 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.

- 32** 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.
- 33** The MAINTENANCE-EXIT clause specifies that a user-supplied index maintenance exit routine is used to suppress the creation of selected index pointer segments.
- 34** The POINTERS clause specifies how the index pointer segment is to point to the index target segment.
- DIRECT-ADDRESS clause specifies that a 4-byte direct address pointer to the index target segment is to be placed in the prefix of the index pointer segment.
  - SYMBOLIC specifies that symbolic pointing from the index pointer segment to the index target segment is to be used, and that no space is to be reserved in the prefix of the index pointer segment for a 4-byte direct address pointer. SYMBOLIC must be specified if the index target segment resides in a HISAM database.

If the POINTERS clause is omitted, then SYMBOLIC is assumed if the index target segment resides in a HISAM database; otherwise DIRECTADDRESS is assumed.

- 35** If symbolic pointing is used to point to the index pointer segment, the concatenated key of the index target segment must form part of the index pointer segment. If it does not, then when the Source Language Generation Facility is used to generate DBD control statements, record layouts, or COBOL, PL/I, or Assembler data descriptions, the concatenated key is constructed automatically and inserted into the index pointer segment after any sequence key and duplicate-data fields that have been specified.
- 36** The CONCATENATED-KEY-NAME clause can be used to specify a name to be given to the concatenated key of the index target segment that will be constructed. If the CONCATENATED-KEY-NAME clause is used, a member of a special internal type is created for the concatenated key and given the name specified in the clause. This internal member has no entries in the uses table, as the members that constitute it are not calculated until the Source Language Generation Facility is used. 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. Interrogations can be performed on internal members of this type (see ["Interrogation Syntax" on page 154](#)). However, meaningful results will only be obtained in response to interrogations about members that refer to the internal member type.

- 37** 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
  - The concatenated key of the index target segment if constructed by Manager Products
  - Any user data
- 38** 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.
- 39** 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.
- 40** When an ATTRIBUTES clause followed by a FREQUENCY clause is encoded, it is assumed that the FREQUENCY clause is subordinate to the ATTRIBUTES clause, specifying the expected frequency of the segment being defined.

If it is required to specify the FREQUENCY common clause following an ATTRIBUTES clause, it is also necessary to specify another common clause before the FREQUENCY common clause. This causes Manager Products to recognize that the clauses that follow are all common clauses.

- 41** 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 where the 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.)

- 42** If an encoded segment record is deleted, any internal 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.
- 43** In the KNOWN-AS clause, *local-name* can be used instead of the name or alias of the contained member, when DBD control statements, record layouts, or source language data descriptions are generated from this member. *local-name* is not separately recorded in the repository (that is, no dummy data entries record and no index record is created for *local-name* when the member in which it appears is encoded), so *local-name* cannot be interrogated and can be the same as another name, an alias, or a catalogue classification in the repository. *local-name* is the name by which the contained member is known only within the segment defined by this member.

**44** In the CONTAINS clause:

- *Version* specifies 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 on page 61](#). If *version* is omitted or if the stated version does not exist, the lowest numbered existing version is assumed to be relevant.
- *Literal* must be compatible with *item-b's form-description* (and *contents-description*, if *item-b* contains a CONTENTS clause) The literal can be either a character string of not more than 256 printable and/or non printable characters, enclosed in quotes, or a numerical 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
- *Version-c* specifies the version (within the HELD-AS form, or within a defaulted form as stated in [remark 10 on page 61](#)) of *item-c* whose contents are the comparand. If *version-c* is omitted, a default value of 1 is assumed.
- The conditional operators have these meanings:

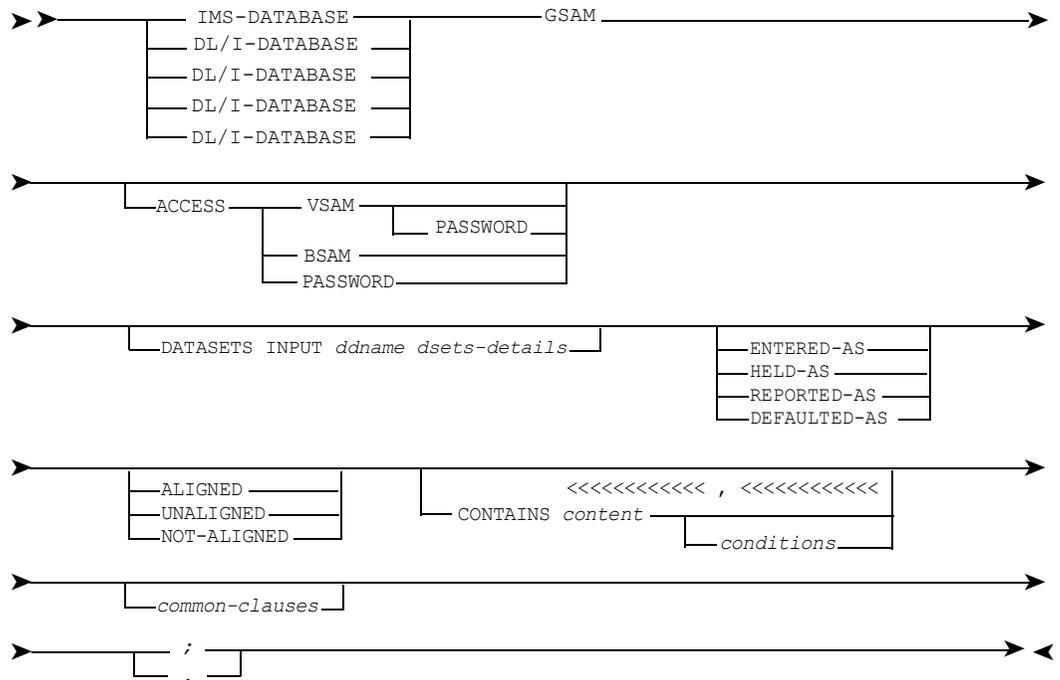
<b>Operator</b>	<b>Meaning</b>
EQ or =	equal to
NE	not equal to
GT or >	greater than
GE	greater than or equal to
LT or <	less than
LE	less than or equal to

**Example**

For an example of a SEGMENT INDEX-POINTER, see the example illustrated by [Figure 3 on page 10](#).



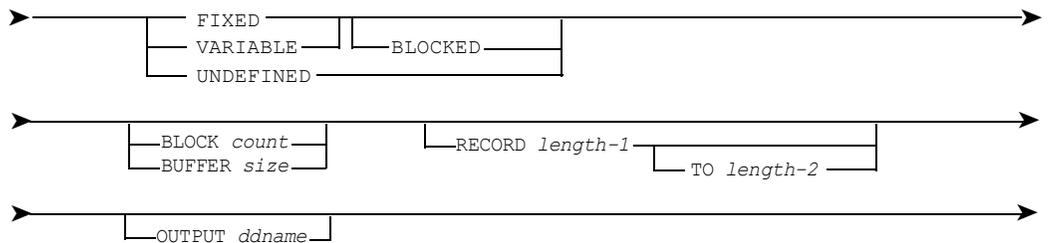
**Member Type of a GSAM Type IMS (DL/I) Database Syntax**



where:

*ddname* is 1 to 8 alphanumeric characters, being the logical name used in the job control to identify the database dataset.

*dsets-details* are:



where:

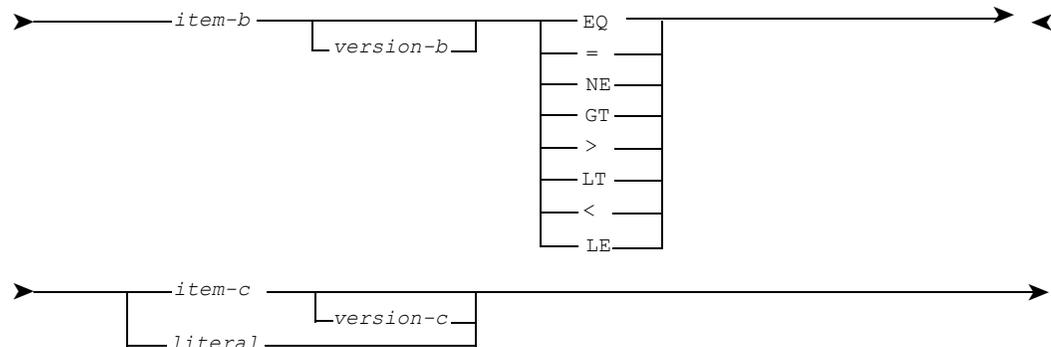
*count* is an unsigned, nonzero integer, being the number of logical records per physical block.

*size* is an unsigned, nonzero integer, being the number of bytes per physical block or control interval.



where:

*cond* is:



where:

*literal* is a literal comparand.

*item-b* is the name of the ITEM whose contents are to be compared with the comparand.

*version-b* is an unsigned integer in the range 1 to 15.

*item-c* is the name of the ITEM whose contents are the comparand.

*version-c* is an unsigned integer in the range 1 to 15.

*common-clauses* are as defined in the *ASG-Manager Products Dictionary/Repository User's Guide*.

### Remarks

- 1 The keyword GSAM must immediately follow the member type identifier to indicate that a OSAM database is being defined.
- 2 The ACCESS clause can be omitted; but, if it is present, it must immediately follow the GSAM keyword.
- 3 If the ACCESS clause is omitted, or if neither of the operating system access method keywords VSAM or BSAM is present in the ACCESS clause, then VSAM is assumed.
- 4 PASSWORD specifies that the database name is to be used when opening any dataset in this database. It is not accepted if the BSAM access method is specified.

- 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. Only one DATASETS clause is permitted.
- 6 INPUT ddname specifies the logical file name of the input dataset. It must be unique within the data dictionary.
- 7 The format of the records in the dataset is specified by one of these keywords: FIXED, VARIABLE, or UNDEFINED.
- 8 If the database uses the VSAM access method, the control interval size, specified either by BUFFER size or by the product of the BLOCK count and *length-1* (if records are fixed length) or *length-2* (if records are variable length), must not exceed 30720.

If the control interval size is specified by BUFFER size, then:

- If size is less than 8192 and is not a multiple of 512, on encoding it is rounded up to the next multiple of 512.
- If size is greater than 8192 and is not a multiple of 2048, on encoding it is rounded up to the next multiple of 2048.

If the control interval size is specified by the product of the BLOCK count and the RECORD length, no rounding is performed by DataManager, but on encoding, warning messages are output if:

- The product is less than 8192 and is not a multiple of 512
- The product is greater than 8192 and is not a multiple of 2048

- 9 The element *length-1* specifies the record size for a fixed length logical record or the minimum record size for a variable length logical record.
- 10 If records are variable length, TO *length-2* must be declared, where *length-2* specifies the maximum size for a record.
- 11 OUTPUT ddname specifies the logical file name of the output dataset.

**Note:** \_\_\_\_\_

If this is the same as the INPUT ddname described in [remark 6 on page 73](#), a Warning message is issued and a common DL/I-DATASET internal member is referred to.

\_\_\_\_\_

- 12** The optional CONTAINS clause and its optional preceding form and alignment keywords are not relevant to IMS (DL/I) for the definition of the GSAM database. They are provided to enable the user to define the records of the dataset group accessed, for documentation or other purposes. As the CONTAINS clause and its optional preceding keywords are based on the corresponding elements of the FILE data definition statement, defined in the *ASG-Manager Products Dictionary/Repository User's Guide*, they are not defined again here.
- 13** Common clauses can be present in any type of data definition statement; therefore, they are 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 an 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 record-definition 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, the DL/I-DATASET internal member type, is created for each ddname that appears in the database's data definition. When a GSAM database is encoded, the DL/I-DATASET internal member type has no references to other members. However, DL/I-DATASET members may be referred to by other members; for example, they might appear in the INPUTS clause of a PROGRAM definition. DL/I-DATASET members can be interrogated (see ["Examples" on page 142](#)), although meaningful results will be obtained only in response to interrogations concerning the members that refer to DL/I-dataset members.

- 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. Any DL/I-DATASET member, which is referred to by other members, is made into a dummy member rather than being deleted.
- 17** In the KNOWN-AS clause, the *local-name* variable can be used instead of the name or alias of the contained member, when DBD control statements, record layouts, or source language data descriptions are generated from this member by the Source Language Generation Facility.



*device* is one of these keywords or numbers:

DRUM	2311	3310	3370	3420
CELL	2314	3330	3375	
TAPE	2319	3340	3380	
	3390	2301	2321	3344
	3400	2305	2400	3350

From IMS version 4 onwards, this clause is purely documentary.

*model* is an integer, 1 or 2 if *device* is 2305, or 1 or 11 if *device* is 3330. From IMS version 4 onwards, this clause is purely documentary.

*segment* is the name of any physical segment.

*common-clauses* are as defined in the *ASG-Manager Products Dictionary/Repository User's Guide*.

### 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 2 SIMPLE 2 keyword(s).
- 4 The keyword PASSWORD specifies that the database name must be used when opening any dataset in this 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. Only one DATASETS clause is permitted.
- 6 INPUT *ddname* specifies the logical file name of the input dataset. It must be unique within the data dictionary.
- 7 OUTPUT *ddname* specifies the logical file name of the output dataset. It must be unique within the data dictionary.

- 8** 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 dataset must be equal to or greater than the length specified in the RECORD clause for the input dataset.
- 9** The DEVICE clause specifies the physical storage device for these datasets. The MODEL clause is subordinate to the DEVICE clause and must not be present unless device is 2305 or 3330, in which case, the MODEL clause is optional.
- 10** 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.
- 11** The CONTAINS clause for a SIMPLE RSAM database states the name of the one segment that resides in the database.
- 12** 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.
- 13** The PARENT clauses identify the physical parents of the segment where the 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.
- 14** Common clauses can be present in any type of data definition statement; therefore, they are 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.
- 15** The common clauses can be declared in any order. If present, they must follow the ACCESS, DATASETS, and CONTAINS clauses, if these are present.
- 16** 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 user 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 ["Interrogation Syntax" on page 154](#)).

If, when the encoded record is generated, any segment where the 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.

- 17 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 a dummy member rather than being deleted.
- 18 From IMS version 4 onwards, the DEVICE and MODEL clauses are purely documentational. This means that DEVICE and MODEL clauses are not generated by PRODUCE IMS VERSION 4/4.1 DBDGEN. For further information on PRODUCE IMS, see [Chapter 5, "IMS \(DL/I\) Source Language Generation," on page 175](#).

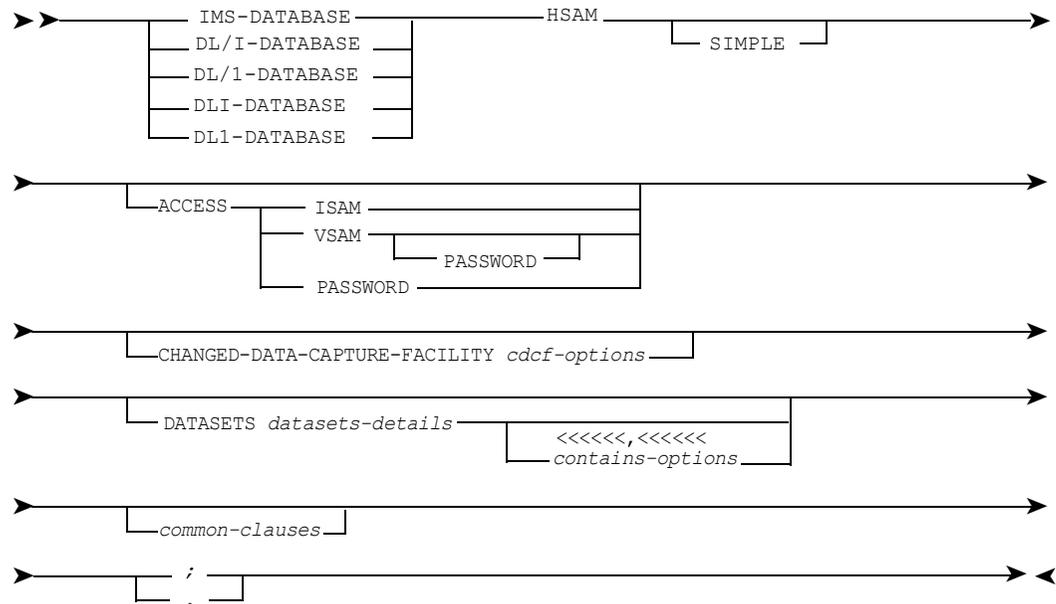
### Example

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 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 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-STATHS
```

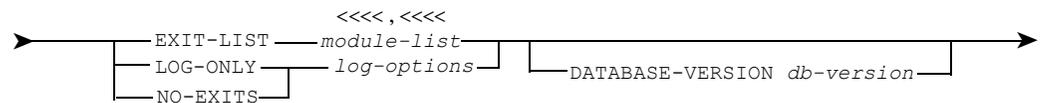
## The Member Type for a HISAM Type IMS (DL/I) Database

### Syntax



where:

*cdc**f*-*options* are:



where:

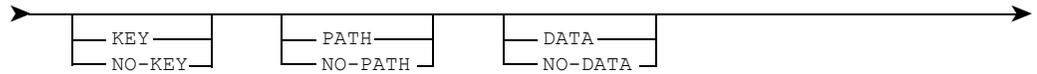
*module-list* is:



where:

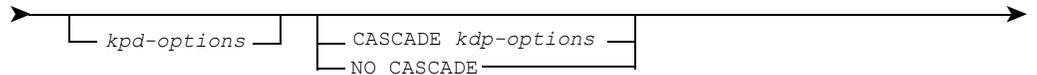
*module-name* is the name of a MODULE or PROGRAM member.

*kpd-options* are:



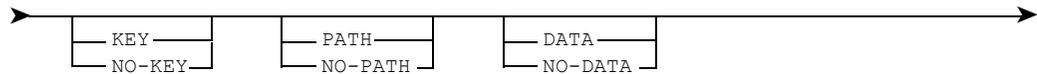
*db-version* is a delimited string of up to 255 characters.

*log-options* are:

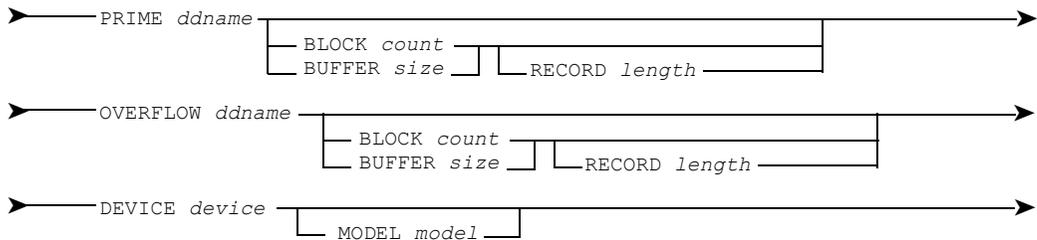


where:

*kpd-options* are:



*datasets-details* are:



where:

*ddname* is 1 to 8 alphanumeric characters, being the logical name used in the job control to identify the database dataset.

*count* is an unsigned, nonzero integer, being the number of logical records per physical block.

*size* is an unsigned, nonzero integer, being the number of bytes required per physical block or control interval.

*length* is an unsigned nonzero 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.



- 5** The operating system access method must be VSAM if any of the following conditions apply:
- The database is a SIMPLE HISAM database.
  - Any segment residing in this database participates in a secondary index relationship; that is, the database being defined is to be indexed by a secondary index.
  - Any segment residing in this database has EDIT-COMPRESSION-EXIT specified in its data definition.
  - Any segment residing in this database is a variable length segment.
  - The target environment is IMS/ESA Version 3 onwards.

If none of these conditions apply, then either VSAM or ISAM can be specified.

- 6** The keyword PASSWORD specifies that the database name is to be used when opening any dataset in this database. It is not accepted if the ISAM access method is specified.
- 7** The DATASETS clause defines a dataset group within this database. It must be present if the definition of the database is to be complete. If VSAM is the operating system access method, only one dataset group can be specified. If ISAM is the operating system access method, the database can be divided into up to 10 dataset groups, provided that it is not indexed by a secondary index.
- 8** Each DATASETS clause is followed by a CONTAINS clause listing the segments that constitute the dataset group. The DATASETS clauses must be entered in the correct sequence to enable the segments residing in the database to be specified in hierarchical sequence generated and inserted into the data entries dataset.
- 9** The first DATASETS clause defines the primary dataset group; subsequent DATASETS clauses define the secondary dataset groups.
- 10** Within the DATASETS clause, the PRIME clause must always be specified. It defines the prime dataset of the dataset group.
- 11** The OVERFLOW clause defines the overflow dataset of the dataset group. It must not be entered for a SIMPLE HISAM database. For a HISAM database, it must be entered unless the database contains only one segment type and the access method is VSAM, in which case it is optional.
- 12** 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.

- 13** If an OVERFLOW clause and a PRIME clause are both present:
- 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.
  - If a BUFFER subordinate clause is present in either, a BUFFER subordinate clause must be present in both.
- 14** 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.
- 15** The RECORD length specified for a SIMPLE HISAM database must be equal to the length of the contained segment.
- 16** If the database uses the VSAM access method:
- 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 30720 if the control interval size is specified by BUFFER size.
  - If size is less than 8192 and is not a multiple of 512, on encoding it is rounded up to the next multiple of 512.
  - If size is greater than 8192 and is not a multiple of 2048, on encoding it is rounded up to the next multiple of 2048.
  - If the control interval size is specified by the product of the BLOCK count and the RECORD length, no rounding is performed, but on encoding warning messages are output if:
    - The product is less than 8192 and is not a multiple of 512
    - The product is greater than 8192 and is not a multiple of 2048
- 17** The DEVICE clause specifies the physical storage device for the dataset group. The MODEL clause is subordinate to the DEVICE clause and must not be present unless device is 2305 or 3330, in which case the MODEL clause is optional.
- 18** 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 that defines the dataset group in which the segments listed in that CONTAINS clause reside.
- 19** A SIMPLE HISAM database can only contain one segment. For a HISAM database, 1 to 255 different segments can be specified in total.

- 20** A HISAM ISAM database can be divided into multiple dataset groups only at the second level of the hierarchy. Therefore:
- In the CONTAINS clause associated with the first dataset group, the name of the root segment must be the first physical-segment-name listed.
  - In any CONTAINS clause relating to a secondary dataset group (that is, a CONTAINS clause associated with any DATASETS clause except the first) the first physical-segment-name listed must be the name of a segment that is a second level dependent of the root segment.
- 21** Regardless of how many CONTAINS clauses are present, the segments must be specified throughout the database definition in hierarchical sequence, that is from top to bottom and left to right.
- 22** The PARENT clauses identify the physical parents of the segments where the 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.
- 23** Common clauses can be present in any type of data definition statement; therefore, they are 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.

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 user 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 ["Interrogation Syntax" on page 154](#)).

If, when the encoded record is generated, any segment where the 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, 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.
- 27** From IMS version 4 onwards, the DEVICE and MODEL clauses are purely documentational. This means that DEVICE and MODEL clauses are not generated by PRODUCE IMS VERSION 4/4.1 DBDGEN. For further information on PRODUCE IMS, see [Chapter 5, "IMS \(DL/I\) Source Language Generation," on page 175](#).

### Examples

These examples of data definition statements for HISAM databases relate to the hierarchical structure of segments listed in ["The Member Type for a HSAM Type IMS \(DL/I\) Database" on page 75](#). 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.

The first example illustrates the specification of the database with the VSAM access method, and with all of the segments contained in one dataset group. The database could be defined thus:

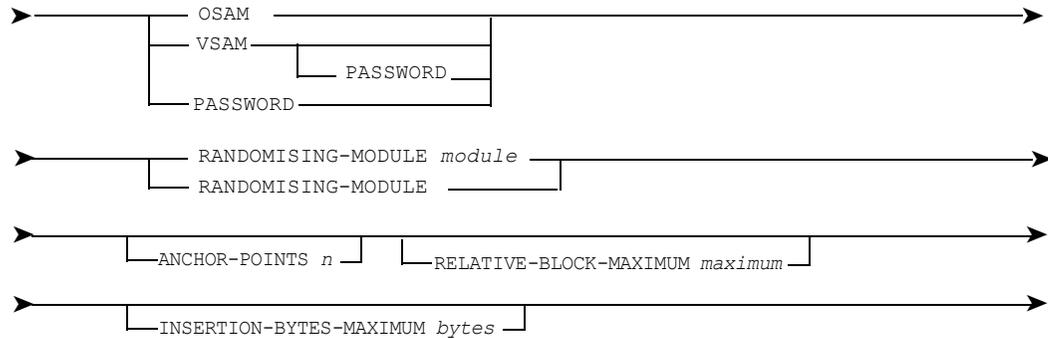
```
ADD EMPLOYEE-DETAILS :
IMS-DATABASE HISAM
ACCESS VSAM PASSWORD
DATASETS PRIME EMPLP BUFFER 2048
          OVERFLOW EMPLO BUFFER 4096
          DEVICE 3330 MODEL 1
CONTAINS DEPARTMENT.
          EMPLOYEE-NUMBER PARENT DEPARTMENT,
          NAME PARENT EMPLOYEE-NUMBER.
          ADDRESS PARENT EMPLOYEE-NUMBER.
          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
```

```
;
```



where:

*access-details* are:



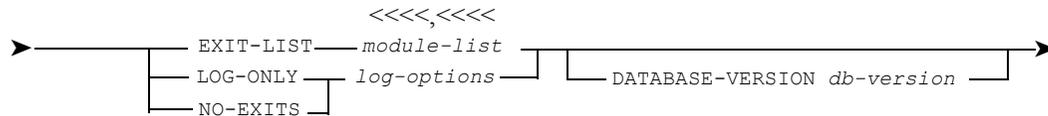
*module* is the name of a MODULE member.

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

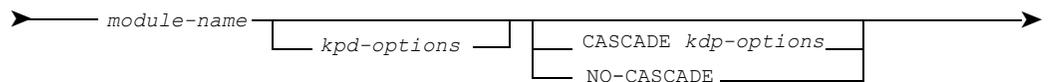
*maximum* is an unsigned integer in the range 1 to 16777215, being the maximum block number to be produced by the randomizing module.

*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.

*cdcf-options* are:



*module-list* is:



*module-name* is the name of a MODULE or PROGRAM member.

*kps-options* are:





*common-clauses* are as defined in the *ASG-Manager Products Dictionary/Repository User's Guide*.

**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 must be specified if the definition of the database is to be complete. If present, it must immediately follow the HDAM keyword.
- 3** If neither of the operating system access method keywords OSAM or VSAM is present in the ACCESS clause, VSAM is assumed.
- 4** The operating system access method must be VSAM if any of the following conditions apply:
  - Any segment residing in the database participates in a secondary index relationship; that is, the database being defined is indexed by a secondary index.
  - Any segment residing in this database has EDIT-COMPRESSION-EXIT specified in its data definition.
  - Any segment residing in this database is a variable length segment.
- 5** The keyword PASSWORD specifies that the database name is to be used when opening any dataset in this database. It is not accepted if the OSAM access method is specified.
- 6** 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.
- 7** 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.
- 8** 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.
- 9** Each DATASETS clause is followed by a CONTAINS clause listing the segments that constitute the dataset group. The DATASETS clauses must be entered in the correct sequence to enable the segments residing in the database to be specified in hierarchical sequence.

- 10** The first DATASETS clause defines the primary dataset group—those subsequent DATASETS clauses that contain PRIME clauses define the secondary dataset groups.
- 11** A HDAM database can be divided into multiple dataset groups at any level of the hierarchy; however, a physical parent segment and its physical child segments must be connected by physical child/physical twin pointers when they are placed in different dataset groups.
- 12** The purpose of the DATASETS clauses containing ADD-TO clauses is to enable segments to be placed in dataset groups according to their size or frequency of access rather than according to their hierarchical position in the data structure, while still maintaining the hierarchical sequence of specification of the segments (see [remark 24 on page 91](#)).
- 13** A DATASETS clause containing a PRIME clause must be present for each dataset group specified. It defines the prime dataset of the dataset group.
- 14** The ddname in each PRIME clause must be unique in the data dictionary.
- 15** If the database uses the VSAM access method:
  - The control interval size, specified either by the BUFFER size or by the BLOCK size, should not exceed 30720.
  - If the control interval size is specified by BUFFER size, then:
    - If size is less than 8192 and is not a multiple of 512, on encoding it is rounded up to the next multiple of 512.
    - If size is greater than 8192 and is not a multiple of 2048, on encoding it is rounded up to the next multiple of 2048.
  - If the control interval size is specified by the BLOCK size, no rounding is performed and no messages are output. This is because the IMS BLOCK operand in this context specifies the control interval size without overheads; therefore, the total control interval size cannot be validated.
- 16** The DEVICE clause specifies the physical storage device for the dataset group. The MODEL clause is subordinate to the DEVICE clause and must not be present unless device is 2305 or 3330, in which case the MODEL clause is optional.
- 17** 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 three cylinders is assumed.
- 18** The FREQUENCY-FREE-BLOCKS clause specifies that, where frequency = every *n*th control interval or block in this dataset group is to be left as free space during database load or reorganization.

- 19** 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 dataset group during database load or reorganization.
- 20** The ADD-TO clause indicates that the segments specified in the following CONTAINS clause are to be placed in a dataset group that has been defined in a previous DATASETS clause containing a PRIME clause with the same ddname as is specified in the ADD-TO clause.
- 21** When the Source Language Generation Facility produces DBD control statements for the HDAM database, labels are created to connect the DATASET statements by using the ddname.
- 22** 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 (containing either a PRIME clause or an ADD-TO clause) that defines the dataset group in which the segments listed in that CONTAINS clause reside.
- 23** One to 255 different segments can be specified in total for the database.
- 24** Regardless of how many CONTAINS clauses are entered, the segments must be specified throughout the database definition in hierarchical sequence; that is, from top to bottom and left to right.
- 25** The first physical-segment-name listed in the first CONTAINS clause must be the name of the root segment. Each of the subsequent CONTAINS clauses can have the name of a segment at any level of the hierarchy as its first physical-segment-name.
- 26** 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.
- 27** Common clauses can be present in any type of data definition statement; therefore, they are 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.
- 28** The common clauses can be declared in any order. If present, they must follow any ACCESS, DATASETS, and CONTAINS clauses.

- 29** 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 user 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 ["Interrogation Syntax" on page 154](#)). If, when the encoded record is generated, any segment or module where the 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.
- 30** 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.
- 31** The SPACE-SEARCH-ALGORITHM clause specifies the selection of a HD free space search algorithm. This does not apply to IMS/VS releases prior to IMS/VS 2.2 and should only be specified when IMS/VS 2.2 or subsequent releases are installed. Values may be set to 1 or 2:
- If 1, IMS should not look for the second most desirable block. This is as per the processing prior to IMS/VS 2.2.
  - If 2, the second most desirable block should be searched for free space. This option is new to IMS/VS 2.2.
- The IMS default value if SEARCHA is omitted is specified at IMS SYSGEN time.
- 32** From IMS version 4 onwards, the DEVICE and MODEL clauses are purely documentary. This means that DEVICE and MODEL clauses are not generated by PRODUCE IMS VERSION 4/4.1 DBDGEN. For further information on PRODUCE IMS, see [Chapter 5, "IMS \(DL/I\) Source Language Generation," on page 175](#).

### Examples

The two following examples of data definition statements for HDAM databases relate to the hierarchical structure of segments illustrated in ["The Member Type for a HSAM Type IMS \(DL/I\) Database" on page 75](#). 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.

The first example illustrates the specification of the database with the VSAM access method, and with all of the segments contained in one dataset group. The example includes a number of the optional keywords. 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
;
```

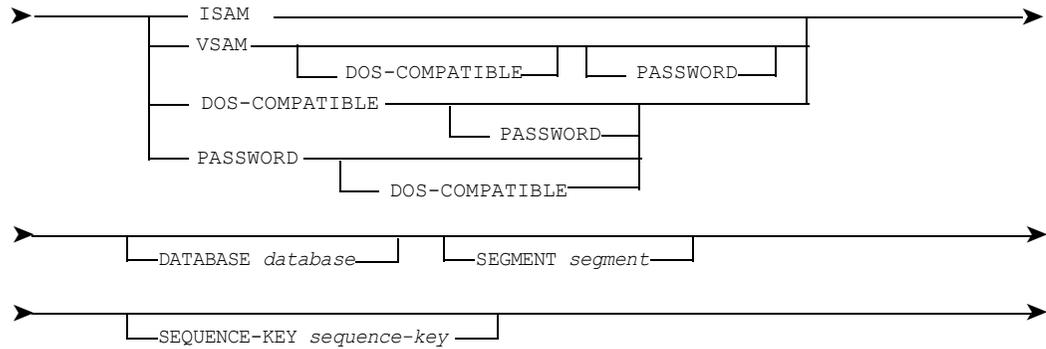
The second example illustrates the specification of the database with the OSAM access method, and with:

- The segments DEPARTMENT, EMPLOYEE-NUMBER, TAXCODE, DEDUCTION-TABLE-REF, and SOCIAL-SECURITY-NUMBER in the primary dataset group
- The segments NAME and ADDRESS in a secondary dataset group
- The segments JOB-STATUS, SALARY, and JOB-TITLE in another secondary dataset group.





where *index-details* are:



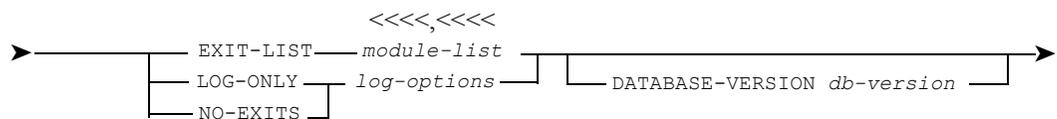
where:

*database* is 1 to 8 alphanumeric characters, being the IMS name of the primary index database associated with this HIDAM database.

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

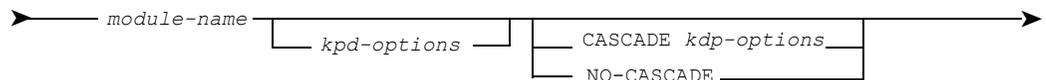
*sequence-key* is 1 to 8 alphanumeric characters, being the IMS sequence key name of the primary index database associated with this HIDAM database.

*cdcf-options* are:



where:

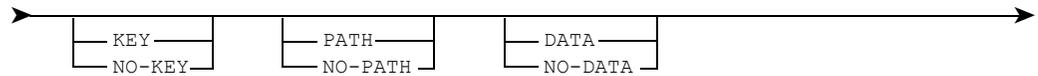
*module-list* is:



where:

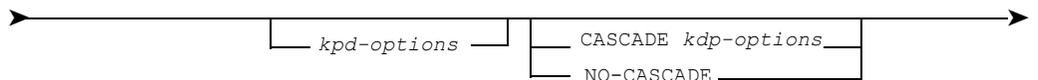
*module-name* is the name of a MODULE or PROGRAM member.

*kpd-options* are:



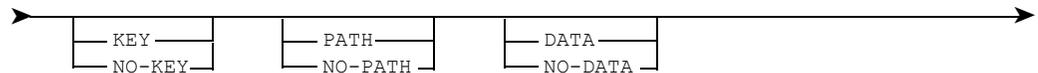
*db-version* is a delimited string of up to 255 characters.

*log-options* are:

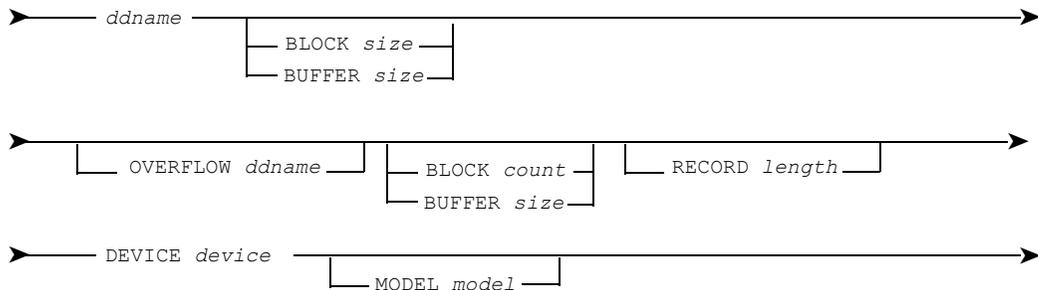


where:

*kpd-options* are:



*i-options* are:



where:

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

*count*, *size*, and *length* are all unsigned nonzero integers.

*size* is an unsigned nonzero integer.

*length* is an unsigned nonzero integer.

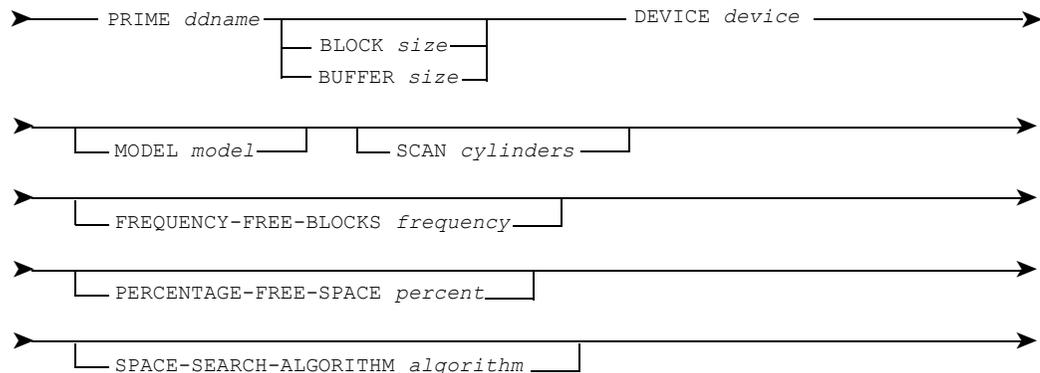
*device* is one of the keywords or numbers from the list:

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

From IMS version 4 onwards, this clause is purely documentary.

*model* is an integer, 1 or 2 if *device* is 2305, or 1 or 11 if *device* is 3330. From IMS version 4 onwards, this clause is purely documentary.

*dset-details* are:



where:

*ddname*, *size*, *device*, and *model* are as defined above.

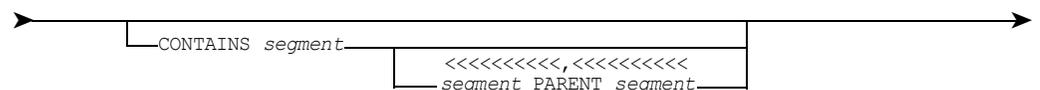
*cylinders* is an unsigned integer in the range 0 to 255.

*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.

*algorithm* is an unsigned integer in the range 0 to 2.

*contains-options* are:



where:

*segment* is the name of a physical segment.

*common-clauses* are as defined in the *ASG-Manager Products Dictionary/Repository User's Guide*.

## 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 IMS \(DL/I\) DBD Control Statements](#)" on page 176.)
  - 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 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** If the ACCESS clause is omitted, or if neither of the operating system access method keywords OSAM or VSAM is present in the ACCESS clause, the VSAM operating system access method is assumed for the HIDAM database.
- 10** The operating system access method for the HIDAM database must be VSAM if:
  - Any segment residing in this database has EDIT-COMPRESSION-EXIT specified in its data definition.
  - Any segment residing in this database is a variable length segment.
- 11** The INDEX subclause 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 98](#) through [remark 6 on page 98](#)). If the clause is not present, or if neither of the operating system access method keywords ISAM or VSAM is present in the clause, VSAM is assumed for the primary index database. If both the keywords DOS-COMPATIBLE and PASSWORD are present, they can be in either order; but, neither of these keywords must precede the VSAM keyword, if that keyword is also present. 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.
- 12** The DOS-COMPATIBLE keyword specifies that the INDEX database was created using DL/1 -DOS. It is applicable only if VSAM is the operating system access method for the INDEX database.
- 13** The PASSWORD keyword may apply to the HIDAM database or to the primary index database, or to both, and is applicable only if VSAM is the operating system access method specified for the database. PASSWORD indicates that the database's name is to be used when opening any dataset in the database.
- 14** Each DATASETS clause defines a dataset group within the primary index database or within the HIDAM database. These clauses must be present if the definition of the databases is to be complete.
- 15** For the primary index database, only one dataset group can be defined.
- 16** The HIDAM database can be divided into one to 10 dataset groups.
- 17** The DATASETS clause that defines the dataset group for the primary index database has no associated CONTAINS clause.

- 18** Each DATASETS clause for the HIDAM database is immediately followed by a CONTAINS clause listing the segments that constitute the dataset group. The DATASETS clauses must be entered in the correct sequence to enable the segments residing in the database to be specified in hierarchical sequence.
- 19** The first DATASETS clause for the HIDAM database (that is, the first clause containing the PRIME keyword) defines the primary dataset group for that database; those subsequent DATASETS clauses that contain PRIME clauses define the secondary dataset groups.
- 20** A HIDAM database can be divided into multiple dataset groups at any level of the hierarchy; however a physical parent segment and its physical child segments must be connected by physical child/physical twin pointers when they are placed in different dataset groups.
- 21** The purpose of the DATASETS clauses containing ADD-TO clauses is to enable segments to be placed in dataset groups according to their size or frequency of access rather than according to their hierarchical position in the data structure, while still maintaining the hierarchical sequence of specification of the segments (see [remark 38 on page 102](#)).
- 22** The INDEX ddname clause defines the prime dataset in the dataset group for the primary index database. The ddname must be unique in the data dictionary.
- 23** The OVERFLOW ddname clause defines the overflow dataset in the dataset group for the primary index database. The ddname must be unique in the data dictionary. The OVERFLOW clause is specified only if ISAM is the operating system access method for the primary index database.
- 24** 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.
  - If a BUFFER subordinate clause is present in either, a BUFFER subordinate clause must be present in both.
- 25** The RECORD length specified for the OVERFLOW clause must be equal to or greater than the RECORD length specified for the INDEX clause, if both are specified.

- 26** If the database uses the VSAM access method:
- 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 30720.
  - If the control interval size is specified by BUFFER size, then:
    - If size is less than 8192 and is not a multiple of 512, on encoding it is rounded up to the next multiple of 512.
    - If size is greater than 8192 and is not a multiple of 2048, on encoding it is rounded up to the next multiple of 2048.
  - If the control interval size is specified by the product of the BLOCK count and the RECORD length, no rounding is performed; but on encoding, warning messages are output if:
    - The product is less than 8192 and is not a multiple of 512.
    - The product is greater than 8192 and is not a multiple of 2048.
- 27** Each DEVICE clause specifies the physical storage device for the dataset group defined by its containing DATASETS clause. The MODEL clause is subordinate to the DEVICE clause and must not be present unless device is 2305 or 3330, in which case the MODEL clause is optional.
- 28** A DATASETS clause containing a PRIME clause must be present for each dataset group specified for the HIDAM database. It defines the prime dataset in the dataset group.
- 29** The ddname in each PRIME clause must be unique in the data dictionary.
- 30** If the database uses the VSAM access method:
- The control interval size, specified either by the BUFFER size or by the BLOCK size, should not exceed 30720.
  - If the control interval size is specified by BUFFER size, then:
    - If size is less than 8192 and is not a multiple of 512, on encoding it is rounded up to the next multiple of 512.
    - If size is greater than 8192 and is not a multiple of 2048, on encoding it is rounded up to the next multiple of 2048.
  - If the control interval size is specified by the BLOCK size, no rounding is performed and no messages are output. This is because the IMS BLOCK operand in this context specifies the control interval size without overheads; therefore, the total control interval size cannot be validated.

- 31** 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 three cylinders is assumed.
- 32** 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.
- 33** 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.
- 34** The ADD-TO clause in the DATASETS clause for the HIDAM database indicates that the segments specified in the following CONTAINS clause are to be placed in a dataset group that has been defined in a previous DATASETS clause containing a PRIME clause with the same ddname as is specified in the ADD-TO clause.
- 35** When the Source Language Generation Facility produces DBD control statements for the HIDAM database, labels are created to connect the DATASET statements by using the ddname.
- 36** The CONTAINS clauses list the segments that reside in the HIDAM database. For the definition of the database to be complete, the CONTAINS clauses must be present, and each CONTAINS clause must immediately follow the DATASETS clause (containing either a PRIME clause or an ADD-TO clause) that specifies the dataset group in which the segments listed in that CONTAINS clause reside.
- 37** One to 255 different segments can be specified in total for the HIDAM database.
- 38** Regardless of how many CONTAINS clauses are entered, the segments must be specified throughout the database definition in hierarchical sequence; that is, from top to bottom and left to right.
- 39** The first physical-segment-name listed in the first CONTAINS clause must be the name of the root segment. Each of the subsequent CONTAINS clauses can have the name of a segment at any level of the hierarchy as its first physical-segment-name.
- 40** The PARENT clauses identify the physical parents of the segments where the 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.

- 41** Common clauses can be present in any type of data definition statement; therefore, they are 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.
- 42** The common clauses can be declared in any order. If present, they must follow the ACCESS, DATASETS, and CONTAINS clauses, if these are present.
- 43** 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 user 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 ["Interrogation Syntax" on page 154](#)).

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 as a dummy segment record for that member.

- 44** 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.

- 45** The SPACE- SEARCH-ALGORITHM clause specifies the selection of a HD free space search algorithm. This does not apply to IMS/VS releases prior to MS/VS 2.2 and should only be specified when IMS/VS 2.2 or later releases are installed. Values may be set to 1 or 2:
- If 1, IMS should not look for the second most desirable block. This is as per the processing prior to IMS/VS 2.2.
  - If 2, the second most desirable block should be searched for free space. This option is new to IMS/VS 2.2.
- 46** The IMS default value, if SEARCHA is omitted, is specified at IMS SYSGEN time. In the INDEX clause:
- Count is the number of logical records per physical block.
  - Size is the number of bytes required per physical block or control interval.
  - Length is the maximum length (in bytes) of a logical record. If VSAM is the operating system access method, length must be an even value.
- 47** From IMS version 4 onwards, the DEVICE and MODEL clauses are purely documentary. This means that DEVICE and MODEL clauses are not generated by PRODUCE IMS VERSION 4/4.1 DBDGEN. For further information on PRODUCE IMS, see [Chapter 5, "IMS \(DL/I\) Source Language Generation," on page 175](#).

### **Examples**

These two examples of data definition statements for HIDAM database relate to the hierarchical structure of segments illustrated in ["The Member Type for a HSAM Type IMS \(DL/I\) Database" on page 75](#). 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.

The first 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, which 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;
IMS-DATABASE HIDAM
ACCESS VSAM INDEX VSAM DOS-COMPATIBLE PASSWORD
DATASETS INDEX EMPLI BUFFER 1024
                DEVICE 3330 MODEL 1
DATASETS PRIME EMPL BUFFER 2048
                DEVICE 3330 MODEL 1
                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
;
```

The second example shows the specification of the OSAM access method for the HIDAM database and the ISAM access method for its primary index database.

Again, the first DATASETS clause defines the dataset group for the primary index database. The segments constituting the HIDAM database are divided into three dataset groups. Thus:

- The segments DEPARTMENT, EMPLOYEE-NUMBER, TAXCODE, DEDUCTION-TABLE-REF, and SOCIAL-SECURITY-NUMBER are contained in the primary dataset group.
- The segments NAME and ADDRESS are contained in a secondary dataset group.
- The segments JOB-STATUS, SALARY, and JOB-TITLE are contained in another secondary dataset group.



where:

*segment* is the name of a logical or physical segment.

*common-clauses* are as defined in the *ASG-Manager Products Dictionary/Repository User's Guide*.

### **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, SEGM is a 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** If either of the following is specified:
  - A physical segment that is a logical child segment
  - A logical segment that contains in its data definition only one physical segment, which is a logical child segment

then such segments, for the purpose of validation checks, are treated as if they were logical concatenated segments. When DBD control statements are being generated for such segments, Manager Products obtains the destination parent segment to which the logical child segment is related, and a SEGM statement for a logical concatenated segment is generated, with the KEY operand specified as the SOURCE operand for the destination parent. If RXLOG01 is specified as YES by the DGDBD macro, then this processing is not undertaken, so that a SEGM statement is generated with a SOURCE operand for the logical child alone.

- 14** 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. If only one logical relationship has been crossed, dependent segments of any of the inverted order segments can be included, but with their order unchanged.

- 15** 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 nonconcatenated segments.

- 16** Different logical concatenated segments can be specified as dependents of the same logical segment. These concatenated segments can represent different variations of the same physical segments. These variations are specified in the SEGMENT data definitions of the logical concatenated segments by the presence or absence of the KEY-ONLY clause for either of the physical segments represented. In such a situation, only one of the concatenated segments can have dependent logical segments, and this concatenated segment must be specified as the leftmost segment, unless RXLOG02 is specified as YES through the DGDBD macro, in which case the rule that this must be the leftmost is relaxed.
- 17** An application program can be sensitive to one only of the concatenated segments that represent different variations of the same physical segments.
- 18** A physical target segment or a logical segment representing a target segment cannot be accessed through a secondary index if it is a dependent of a concatenated segment.
- 19** 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.

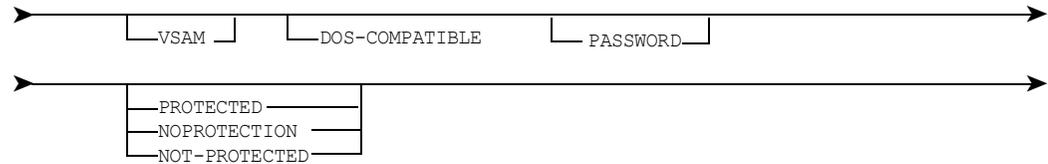
- If the physical segment resides in more than one physical database and does not represent the root segment in the logical database, then a physical database is selected in one of these ways:
  - If the physical database named in the preceding SEGM statement appears anywhere in the used-by table of the physical segment currently being processed, then the name of this physical database is output in the SEGM statement for the current segment also.
  - 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) where the 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.

- 20** 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.
- 21** The common clauses can be declared in any order. If present, they must follow the CONTAINS clause, if that clause is present.
- 22** 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 where the 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.



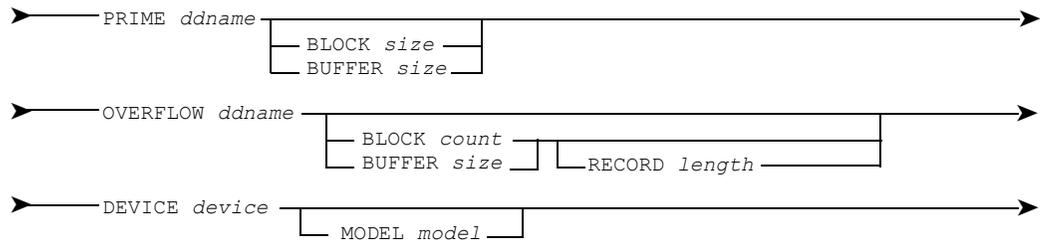
where:

*acc-details* are:



*index-database-name* is the name of another IMS(DL/I) SECONDARY-INDEX database.

*dsets-details* are:



where:

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

*count*, *size*, and *length* are all unsigned non-zero integers.

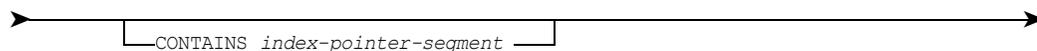
*device* is one of the keywords or numbers from the list:

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

From IMS version 4 onwards, this clause is purely documentary.

*model* is an integer, 1 or 2 if device is 2305, or 1 or 11 if device is 3330. From IMS version 4 onwards, this clause is purely documentary.

*cont* is:



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

*common-clauses* are as defined in the *ASG-Manager Products Dictionary/Repository User's Guide*.

### 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 definition of the database is to be complete, one of the following must be specified:
  - A DATASETS clause, optionally preceded by an ACCESS clause
  - A SHARES-WITH (or SHARING-WITH) clause
  - A DATASETS clause and a SHARES-WITH (or SHARING-WITH) clause must not both be present
- 3 If the ACCESS clause is present, it must immediately follow the INDEX or SECONDARY-INDEX keyword.
- 4 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.
- 5 The keyword DOS-COMPATIBLE specifies that the database was created using DL/I-DOS.
- 6 The keyword PASSWORD specifies that the database name is to be used when opening any dataset in this database.
- 7 The keyword PROTECTED means that an application program is prevented from replacing any of the fields in the index pointer segment, although delete operations are still enabled. The keyword NOPROTECTION or NOT-PROTECTED means that an application program can replace or delete all of the fields in the index pointer segment except the constant, search, and subsequent fields. If none of these keywords are preset, PROTECTED is assumed.
- 8 The DATASETS clause defines a dataset group within the secondary index database. Only one dataset group can be defined. The DATASETS clause must precede the CONTAINS clause, if both these clauses are present.

- 9** The PRIME clause specifies the prime dataset of the dataset group.
- 10** The OVERFLOW clause specifies the overflow dataset 0 the dataset group. This clause must be specified if the index pointer segments contain nonunique keys.
- 11** 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.
- 12** If an OVERFLOW clause and a PRIME clause are both present:
- 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.
  - If a BUFFER subordinate clause is present in either, a BUFFER subordinate clause must be present in both.
- 13** 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.
- 14** 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 30720.
- If the control interval size is specified by BUFFER size, then:
- If size is less than 8192 and is not a multiple of 512, on encoding it is rounded up to the next multiple of 512.
  - If size is greater than 8192 and is not a multiple of 2048, on encoding it is rounded up to the next multiple of 2048.
- If the control interval size is specified by the product of the BLOCK count and the RECORD length, no rounding is performed, but on encoding, warning messages are output if:
- The product is less than 8192 and is not a multiple of 512.
  - The product is greater than 8192 and is not a multiple of 2048.
- 15** The DEVICE clause specifies the physical storage device for the dataset group. The MODEL clause is subordinate to the DEVICE clause and must not be present unless device is 2305 or 3330, in which case the MODEL clause is optional.
- 16** The SHARES-WITH or SHARING-WITH clause, specified instead of the ACCESS and DATASETS clauses, indicates that the secondary index resides in a shared INDEX database.

- 17** The index-database-name must be the same of a secondary index database that has been defined using the DATASETS clause.
- 18** For secondary indexes to be combined into a shared INDEX database, the following conditions must all be true:
- All of the contained index pointer segments must be of equal length.
  - The key fields of each of the index pointer segments must be equal in length with equal key offset positions.
  - Each of the key fields must include a constant that uniquely identifies its index pointer segment.
- 19** A maximum of 16 secondary indexes can share the same secondary index database. That is, a maximum of 15 INDEX database definitions may have the same index-database-name in a SHARES-WITH or SHARING-WITH clause in their data definitions.
- 20** The CONTAINS clause must be present if the definition of the database is to be complete. If present, it must follow the DATASETS clause, the SHARES-WITH, or SHARING-WITH clause if either of those clauses are present. It specifies the index pointer segment that is contained in the secondary index database.
- 21** Not more than one of each of the common 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.
- 22** The common clauses can be declared in any order. If present, they must follow the ACCESS, DATASETS (or SHARES-WITH or SHARING-WITH), and CONTAINS clauses, if these are present.
- 23** 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 ["Interrogation Syntax" on page 154](#)).

If, when the encoded record is generated, any database or segment or module where the 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, a dummy segment record, or a dummy module record, respectively.

- 24** 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.
- 25** In the DATASETS clause:
- Count specifies the number of logical records per physical block.
  - Size specifies the number of bytes required per physical block or control interval.
  - Length specifies the maximum length (in bytes) of a logical record. If VSAM is the operating system access method, length must be an even value.
- 26** From IMS version 4 onwards, the DEVICE and MODEL clauses are purely documentary. This means that DEVICE and MODEL clauses are not generated by PRODUCE IMS VERSION 4/4.1 DBDGEN. For further information on PRODUCE IMS, see [Chapter 5, "IMS \(DL/I\) Source Language Generation," on page 175](#).

### Examples

```
ADD EMPIND:
IMS-DATABASE SECONDARY- INDEX
ACCESS VSAM DOS-COMPATIBLE PASSWORD PROTECTED
DATASETS PRIME EMPIP BUFFER 1024
OVERFLOW EMPIO BUFFER 2046
DEVICE 3340
CONTAINS EMPIND-SEG
;
```

Below is an example of a data definition statement for a secondary index database, using the SHARES-WITH clause:

```
ADD EMPIND2:
IMS-DATABASE INDEX
SHARES-WITH EMPIND
CONTAINS EMPIND2 - SEG
;
```

## Member-type Descriptions for IMS (DL/I) Program Communication Blocks

The data definition statements for PCB members are used to define these PCB types:

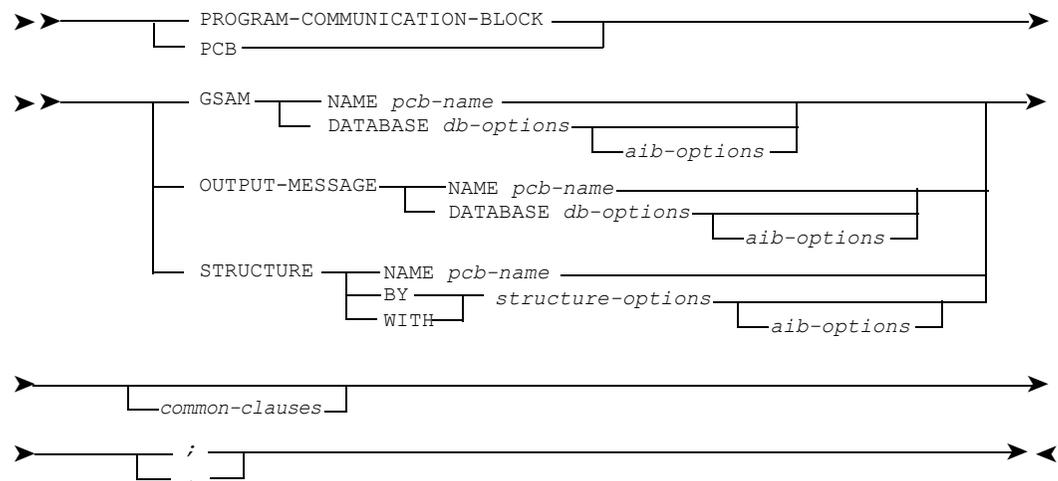
- GSAM database
- Output message destination (that is, IMS Alternate PCBs)
- Logical data structure

PCB members must be defined for any application for which PSB control statements are to be generated.

If specified, IMS will automatically add an I/O PCB for the input message source to the PSBGEN when the program is run in the Batch DL/I region; therefore, a PCB must not be defined for any I/O PCB. The user can specify, on the PRODUCE IMSPSBGEN command, that IMS (DL/I) is to add such a PCB automatically to the PSBGEN.

### PROGRAM-COMMUNICATION-BLOCK

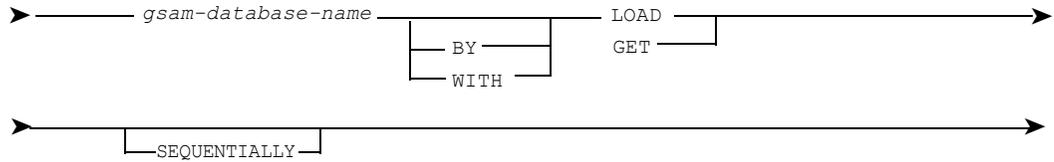
#### Syntax



*pcb-name* is the name of another PROGRAM-COMMUNICATION-BLOCK member.

**ASG-DataManager IMS (DL/I) Interface**

*db-options* are:

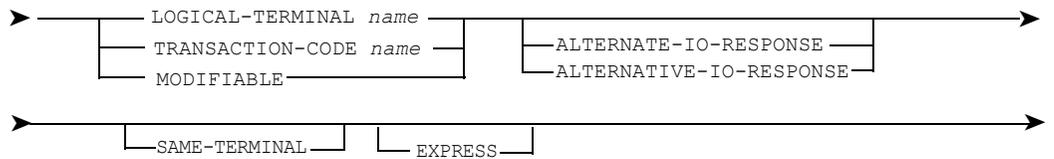


*gsam-database-name* is the name of a database member of the GSAM type.

*aib-options* are:

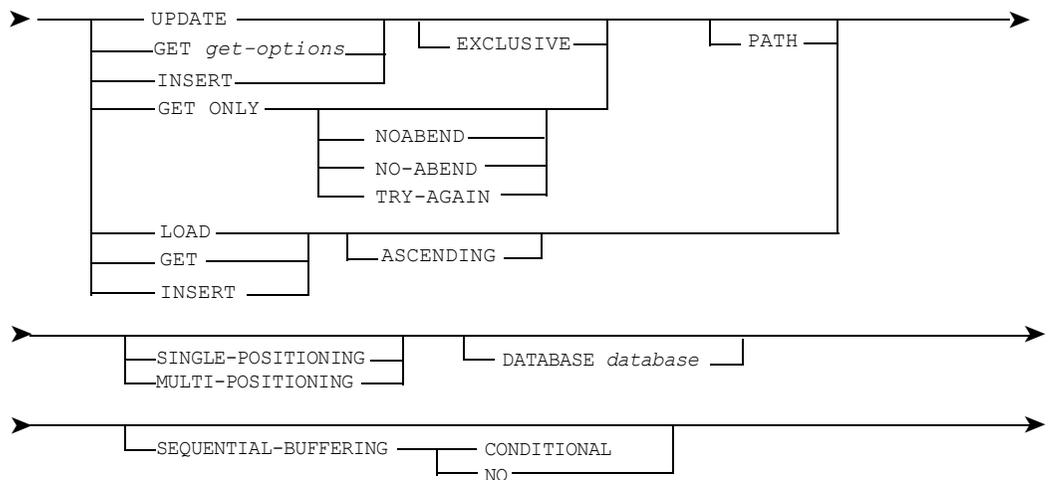


*out-options* are:



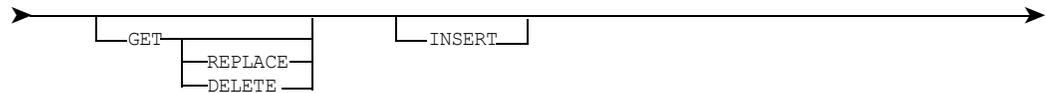
*name* is an alphanumeric name 1 to 8 characters in length.

*structure-options* are:





*grdi-options* are:

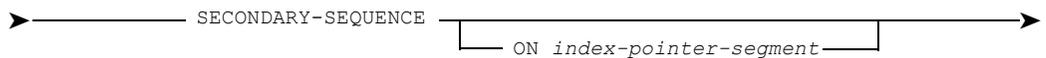


**Note:**

The UPDATE, KEY-SENSITIVE, GET, REPLACE, DELETE, INSERT, EXCLUSIVE, and PATH keywords can all be optionally separated by commas.

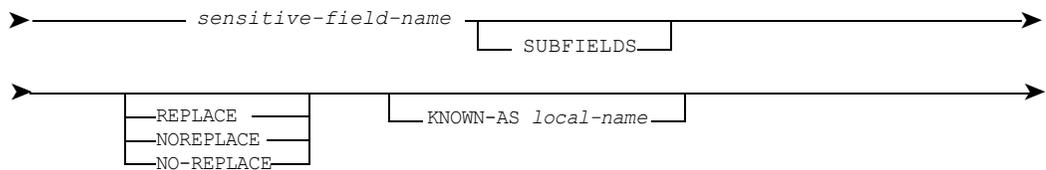
You must specify at least one keyword in *grdi-options*.

*sec-seq-options* are:



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

*sensitive-field* is:



*local-name* is a name conforming to the rules for member names stated in the *ASG-ControlManager User's Guide*.

*sensitive-field-name* is the name of a GROUP, ITEM, sequence key member, or concatenated key member.

*filler-bytes* is an unsigned integer in the range 1 to 32767.

*common-clauses* are as defined in the *ASG-Manager Products Dictionary/Repository User's Guide*.

**Remarks**

- 1** The member type identifiers PROGRAM-COMMUNICATION-BLOCK and PCB are synonymous.
- 2** The first element following the member type identifier must be a keyword that indicates which type of PCB member is being defined; thus:
  - GSAM: the application view of a GSAM database is being defined.
  - OUTPUT-MESSAGE: the application view of an output message destination is being defined.
  - STRUCTURE: the application view of a logical data structure is being defined.
- 3** More than one PCB member specifying the same logical data structure can be defined, provided that each has a unique member name. This enables parallel processing of dependent segment types to be defined without using the multipositioning feature.
- 4** For any type of PCB member, the NAME clause specifies that the data definition of the reference PCB, pcb-name, is to be regarded as being also a data definition of this member; with the exception that pcb-name's common clauses are not applied to this member. The reference PCB must be of the same type as the PCB being defined.
- 5** For a GSAM type PCB member, unless the NAME clause is specified, the DATABASE clause identifies the GSAM database that is relevant to this application view and defines the processing options that the application uses to access that database.
- 6** The BY or WITH keyword can be omitted, but is included in the specification to maintain consistency of format of the processing options clause with the processing options clauses available in the STRUCTURE type PCB member syntax.
- 7** The LOAD keyword specifies that the application loads the database.
- 8** The GET keyword specifies that the application retrieves the database.
- 9** The SEQUENTIALLY keyword indicates large scale sequential activity and that the GSAM multibuffering option is to be utilized.
- 10** For an OUTPUT-MESSAGE type PCB member, unless the NAME clause is specified, either LOGICAL-TERMINAL name or TRANSACTION-CODE name or the keyword MODIFIABLE must immediately follow the OUTPUT-MESSAGE keyword.

- 11** The name is the identifier of the actual destination of the message, and is either a logical terminal name or a transaction code name defined during IMS/VS system definition. When it is a transaction code, IMS/VS routes the message to the application program that processes the specified transaction code.
- 12** The MODIFIABLE keyword indicates that the destination of the message is dynamically specified during program execution.
- 13** ALTERNATE-IO-RESPONSE or ALTERNATIVE-IO-RESPONSE means that a response in response mode, conversational mode, or exclusive mode can be directed to a different logical terminal from the one on which the input message originated.
- 14** SAME-TERMINAL specifies that IMS/VS is to check that the logical terminal name is assigned to the physical terminal from which the input message originated.
- 15** The EXPRESS keyword specifies that the output messages are to be sent even if the program ends abnormally.
- 16** If specified, IMS will automatically add an I/O PCB for the input message source to the PSBGEN when the program is run in the Batch DL/I region; therefore, a PCB must not be defined for any I/O PCB. The user can specify, on the PRODUCE IMS PSBGEN command, that IMS (DL/I) is to add such a PCB automatically to the PSBGEN.
- 17** For a STRUCTURE type PCB member, unless the NAME clause is specified, the first subordinate clause within the STRUCTURE clause must be the structure-options 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 in *struc-options-2*. For each segment for which this is not specified, structure-options applies.
- 18** The structure-options clause defines the functions that can be performed on the logical data structure from the application view (except where overridden for individual segments in segment-options). 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

- 19** 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 structure-options 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.
- 20** For database reading with enqueueing to check the availability of segments, GET is specified. A program which specifies GET is protected from accessing segments that have invalid pointers, as IMS prevents the program from retrieving updated segments until the updating program reaches a synchronization point.
- 21** For database reading only, without enqueueing to check the availability of segments, GET ONLY is specified.
- 22** If only GET ONLY is specified, it should be noted that a program may be able to retrieve a segment that has invalid pointers, and this program may then be terminated abnormally by IMS. This situation can be avoided by specifying NOABEND, NO-ABEND, or TRY-AGAIN in conjunction with GET ONLY. (This will prevent the program from being terminated abnormally if it retrieves a segment that contains invalid pointers.)
- 23** If GET ONLY, NOABEND, or NO-ABEND is specified, a program which retrieves a segment that contains invalid pointers will not be terminated abnormally by IMS. Instead, IMS returns a status code to the program.
- 24** If GET ONLY TRY-AGAIN is specified, then if a program retrieves a segment with an invalid pointer, IMS will attempt another call to the database. If, by the time IMS tries the call again, the program that was updating the requested segment has reached a synchronization point, the pointer in the segment will be valid again and the segment can be retrieved. If the pointer is still found to be invalid when the call is repeated, IMS returns a status code to the program.

- 25** For database reading (with enqueueing) and limited updating, GET is specified followed by whichever one of the keywords REPLACE, DELETE, and/or INSERT are relevant, in any order; but not more than three 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.
- 26** 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.
- 27** For database reading and all updating functions, UPDATE is specified. (UPDATE is thus the equivalent of GET, REPLACE, DELETE, INSERT.)
- 28** 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.
  - GET can only be specified with ASCENDING if the segment is contained within a HSAM database.
- 29** 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.
- 30** 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 IMS (DL/I) to determine the maximum length of the input/output area.
- 31** The keywords ASCENDING, EXCLUSIVE, and PATH can, if present, be in any order.
- 32** If either of the keywords SINGLE-POSITIONING or MULTIPositioning is present, it must immediately follow the structure-options clause. It specifies the type of positioning required for the logical data structure. If neither of these keywords is present, SINGLEPOSITIONING is assumed. MULTI-POSITIONING is invalid if the logical data structure belongs to a HSAM database.

- 33** For a STRUCTURE type PCB member (unless the NAME clause is specified), the DATABASE subordinate clause must be specified if the logical data structure resides in a database, where the segments are also contained by other databases. Otherwise, the DATABASE clause is optional.
- 34** The KEYLENGTH clause specifies the maximum concatenated key length for any path of sensitive segments that is used by the application that 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.

\_\_\_\_\_

- 35** All segments in the logical data structure must belong to the same database.
- 36** The KEEP-HIERARCHY keyword allows a left to right order of sibling segments under a parent, which is different from the order specified in the database definition, to be defined and maintained in a PCB.
- 37** If the KEEP-HIERARCHY keyword is present, then when the Source Language Generation Facility is used to produce PSB control statements, the order of segments specified for the PCB, as defined by the subordinate SEGMENT clauses in the PCB definition will be maintained.
- 38** Only the left to right order of sibling segments under each parent segment may be altered in the PCB definition. The top to bottom order of segments must be maintained as it appears in the database definition.
- 39** The user must ensure that the reordering of segments within a PCB is permissible within the IMS environment being used.
- 40** If KEEP-HIERARCHY is specified, every segment along the hierarchical path to the data sensitive segments must be specified in a SEGMENT clause, and the clauses must be specified in the order required by IMS (DL/I). When PSB control statements are generated, Manager Products checks that the specification of segment order from top to bottom and the specification of sibling segments from left to right under their parent segments is valid. If any segments are missing or are specified in an invalid order, Manager Products issues an error message and will not attempt to reorganize the order of segments or to insert missing ones.
- 41** If KEEP-HIERARCHY is not specified, then when PSB control statements are generated, the segments are organized into the order specified in the database definition, regardless of the order in which they occur in the PCB definition. Any segment along the hierarchical path to the data sensitive segments that has no SEGMENT clause is assumed to be key sensitive.

- 42** If the logical data structure belongs to a LOGICAL database for which different logical concatenated segments are specified as representing different variations of the same physical segment, then the PCB member can be sensitive to only one of the logical concatenated segments. (See [remark 16 on page 109](#).)
- 43** If the LOAD processing option is specified for the logical data structure, and the database to which the logical data structure belongs contains multiple dataset groups, then at least one SEGMENT clause should be specified for each dataset group. For any dataset group in respect of which no SEGMENT clause is specified, key-sensitivity is assumed for the first segment in the dataset group.
- 44** If the LOAD processing option is specified for the logical data structure, then SEGMENT clauses must not be entered for virtual logical child segments.
- 45** From 1 to 255 SEGMENT clauses can be defined for a logical data structure. Only one segment clause may be specified for each segment.
- 46** In each SEGMENT clause, segment-name must immediately follow the SEGMENT keyword to identify the sensitive segment to be processed.
- 47** *struct-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 structure-options can apply to the segment, this can be omitted. If structure-options specified LOAD, this must be omitted.
- 48** KEY-SENSITIVE specifies that the application is only key-sensitive to the segment; that is, the segment is not moved to the program's input/output area, but that the key only is placed in the concatenated key feedback area of the logical data structure's PCB.
- 49** GET, INSERT, and UPDATE, and the optional keywords that can be associated with them, have the same meanings and restrictions as are specified for *structure-options* in [remark 21 on page 123](#) through [remark 28 on page 124](#), but applying to the one segment only. GET ONLY and ASCENDING cannot be specified in *struct-options-2*.
- 50** 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 *struct-options-2*, if specified; otherwise, it must, if present, immediately follow segment.
- 51** If SECONDARY-SEQUENCE is specified, segment must identify an index target segment, or a logical segment representing an index target segment. In a logical database, the segment must not be a dependent of a concatenated segment.

- 52** The SECONDARY-SEQUENCE keyword may only be entered once for any one logical data structure.
- 53** 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.
- 54** 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 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)
- 55** Up to a maximum of 255 sensitive fields can be declared for each segment within a maximum of 10,000 for the PCB member.
- 56** The declaration of a sensitive field includes any associated SUBHELDS, REPLACE, NOREPLACE, or NO-REPLACE keyword and/or KNOWN-AS clause; as well as the sensitive field name. These declarations are listed in the SENSITIVE-HELDS clause, each sensitive-field name except the first in the list being preceded by a comma and, in addition, optionally by spaces.
- 57** SUBHELDS specifies that when the Source Language Generation Facility is used to generate PSB control statements, SENFLD 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 sequence key member or a concatenated key member and:
- Is defined by segment-name, the 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
- 58** Sensitive fields can be repeated provided a KNOWN-AS clause is specified for each repetition, so that unique names can be generated when COBOL, PL/I, or Assembler data description statements are generated.
- 59** The sensitive field keyword 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.

- 60** If none of the sensitive field keywords REPLACE, NOREPLACE, and 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.
- 61** The keywords REPLACE, NOREPLACE, and NO-REPLACE are ignored if neither of the processing options UPDATE or REPLACE are specified.
- 62** 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.
- 63** The SENSITIVE-FIELD clause is invalid if:
- Segment-options is KEY-SENSITIVE.
  - The segment is a logical child segment or a logical concatenated segment, and the processing option applicable is INSERT, LOAD, or UPDATE.
- 64** It is the user's responsibility to declare all the sensitive fields required by IMS (DL/I); for example, sequence key fields and segment search fields, because their start positions cannot be anticipated.
- 65** Common clauses can be present in any type of data definition statement; therefore, they are 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.
- 66** The common clauses can be in any order. If present, they must follow the GSAM, OUTPUT-MESSAGE, or STRUCTURE clause.
- 67** A record containing the PCB'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 PCB, database, segment, or sensitive field, where the name appears in the PCB's data definition statement, has no data entries record, a dummy data entries record is created for that member as a dummy PCB record, a dummy database record, a dummy segment, or dummy item record, respectively.

- 68** The SEQUENTIAL-BUFFERING clause allows the specification of OSAM Sequential Buffering for any database PCB requiring this facility to improve performance with sequential access. This does not apply to IMS/VS releases prior to IMS/VS 2.2 and should only be specified when IMS/VS 2.2 or subsequent releases are installed. Values may be specified as follows:
- NO: sequential buffering should not be used for this PCB. This is the default assumed if SEQUENTIAL-BUFFERING is not specified by other means (i.e., control statements or user exits).
  - CONDITIONAL: requests the conditional activation of sequential buffering for this PCB.
- 69** The *local-name* variable is to be used instead of the name or alias of the sensitive field when PSB control statements, record layouts, or source language data descriptions are generated from this member. *local-name* is not separately recorded in the repository (no dummy data entries record and no index record is created for it when the member 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 repository. *local-name* is the name by which the member forming the sensitive field is known only within the PCB defined by this member.

### **Example of a GSAM type PCB**

This example shows a PCB member specifying the loading of a GSAM database, involving large scale sequential activity, where the GSAM multibuffering option is to be utilized:

```
ADD GSAM-PCB;
PCS GSAM
DATABASE GSAM-OB BY LOAD SEQUENTIALLY
;
```

### **Examples of OUTPUT-MESSAGE Type PCBs**

The first example below shows a PCB member defining an output message destination that is to be dynamically specified during program execution:

```
ADD MOD-PCB;
PCS OUTPUT-MESSAGE
MODIFIABLE
;
```

The next example shows a PCB member specifying an alternative logical terminal to which an application can direct its response (rather than to the source from which the input message originated):

```
ADO LOG-TERM-PCB;  
PCB OUTPUT-MESSAGE LOGICAL-TERMINAL TERM3  
ALTERNATE- 10-RESPONSE  
SAME-TERMINAL EXPRESS  
;
```

### **Examples of STRUCTURE Type PCBs**

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 three 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 RDAM 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.

This example shows a PCB member for a logical data structure residing in the HISAM database SKILLINV that is illustrated in [Figure 2 on page 7](#):

```
ADD SKILLINV-PCB;  
PCB STRUCTURE  
BY GET, INSERT  
SEGMENT SKILLMAST  
SEGMENT SKILLNAM  
SEGMENT EXPRMAST BY INSERT PATH  
SEGMENT EDUCMAST 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, a PCB member could be defined thus:

```
ADD ASY-PACK-PCB;  
PCB STRUCTURE  
BY GET ONLY  
SEGMENT ASY-LINE  
SEGMENT ASY-PACK  
SENSITIVE-FIELDS PACK-NO,      PROD-NO,      QTY-REOD  
;
```

This example shows a PCB member for a logical data structure residing in the database EMPLOYEE-DETAILS.

The application is to be data sensitive only to the segments EMPLOYEE-NUMBER, SOCIAL-SECURITY-NUMBER, and TAX-CODE. KEEP-HIERARCHY has been specified in the PCB definition as the order of the segments SOCIAL-SECURITY-NUMBER and TAX-CODE is to be altered. As a result (see [remark 38 on page 125](#)), SEGMENT clauses have had to be specified for each of the key-sensitive segments.

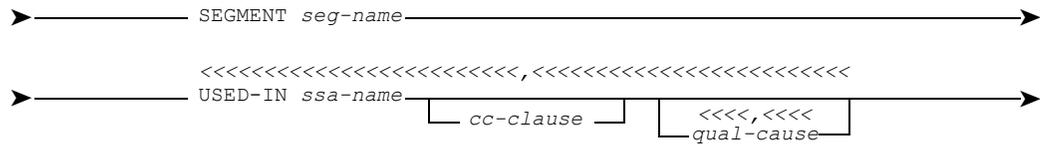
```
ADD EMPL-PCB;  
PCB STRUCTURE  
BY GET ONLY  
DATABASE EMPLOYEE-DETAILS  
KEEP-HIERARCHY  
SEGMENT DEPARTMENT WITH KEY-SENSITIVE  
SEGMENT EMPLOYEE-NUMBER  
SEGMENT JOB-STATUS WITH KEY-SENSITIVE  
SEGMENT SALARY WITH KEY-SENSITIVE  
SEGMENT SOCIAL-SECURITY-NUMBER  
SEGMENT TAX-CODE  
;
```



where:

*pcb* is the name of a PCB member.

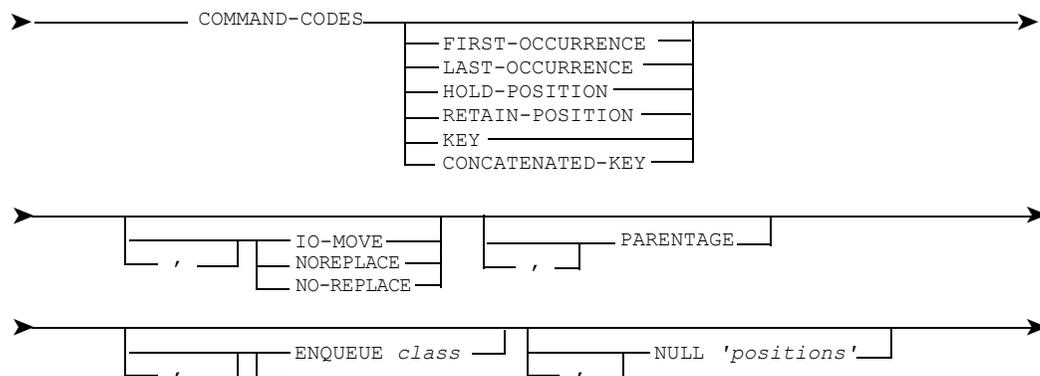
*ssas-clause* is:



*seg-name* is the name of a SEGMENT member.

*ssa-name* is the segment-search-argument name for the language (PL/I, COBOL, or Assembler) relevant to the appropriate member.

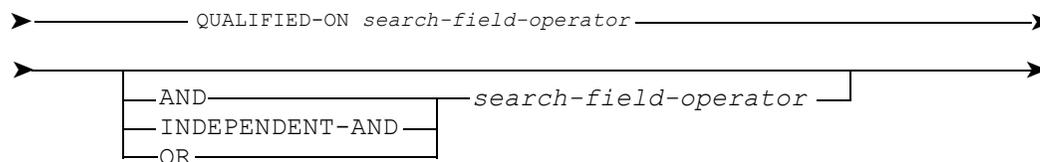
*cc-clause* is:



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

*positions* is an unsigned integer.

*qual-clause* is:



*search-field* is a sequence key field in a definition of virtual logical child segment (including all sequence key fields following it).

*operator* is one of these: EQ or =  
NE  
GT or >  
GE  
LT or <  
LE

### Remarks

- 1 The keyword IMS or DL/I (or one of its permitted variants) must immediately follow the PROCESSES keyword to indicate that an IMS (DL/I) application view is being defined. The keyword IMS is synonymous with DL/I and its variants.
- 2 If the CONTAINS subordinate clause is present, it must immediately follow the IMS or DL/I keyword.
- 3 If the Source Language Generation Facility is to be used to produce 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 five fields are its search fields. [See "[Application View](#)" on page 19 and "[Generating IMS \(DL/I\) DBD Control Statements](#)" on page 176.]
- 4 If the condition stated in [remark 3 on page 134](#) 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 nonunique 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 nonunique 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 command code `HOLD-POSITION` prevents position being moved from an occurrence of this segment under its parent (if position has previously been established on the parent) during a search of its hierarchical dependents. When a call is being satisfied, if position is moved to a level above that at which the command was issued, the code has no effect for occurrences to the segment where the parent changed.
- 12** The command code `RETAIN-POSITION` has the same meaning as the command `HOLD-POSITION` except that the command code is automatically set at all higher levels in the call. This means that position cannot be moved at all from the existing position at the level at which this command code is issued.
- 13** The command code `KEY` or `CONCATENATED-KEY` can be used when the concatenated key of the segment is available. When the Source Language Generation Facility produces the COBOL, PL/I, or Assembler data description for a segment-search-argument with this command code, it generates a parenthesized field containing the appropriate number of hexadecimal zeros, into which the application program can insert the segment's concatenated key. Only one segment search argument with this command code is allowed per call, and it must be the first in the call.
- 14** The `10-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-1. 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.
- 15** 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.
- 16** The command code `PARENTAGE` specifies that parentage is to be set at this level; therefore, succeeding `GET NEXT WITHIN PARENT` calls will treat this level as the parent level rather than the lowest level segment returned on this call. The parentage will remain in effect until a `GET UNIQUE` or `GET NEXT` call is issued.
- 17** The command code `ENQUEUE` class specifies that this segment is to be enqueued for a single update, where class is the class identifier used on the dequeue call to dequeue all resources enqueued by the user with that class.

- 18** 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.
- 19** The `QUALIFIED-ON` clause defines information that IMS (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. This clause is not valid if a command code of `KEY` or `CONCATENATED-KEY` is present in the `USED-IN` clause, as the concatenated-key of this segment then replaces the qualification statement in the `segment-search-argument`.
- 20** The `QUALIFIED-ON` keyword, if present, must be followed immediately by `search-field`, which can identify a field of any of the following 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 `OF` 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 [remark 22 on page 137](#))
  - 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 dependent of a logical concatenated segment, then the field is 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 is defined as `sequence-key-name` in this segment's data definition. In this case, the field specified by `search-field` includes any constant and subsequence fields specified in the segment's data definition.
- 21** 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.

- 22** If *search-field* is an index-search-field-name, then when the Source Language Generation Facility produces PSB control statements for this application, it automatically generates an INDICES = *index-database-name* entry on the SENSEG statement for this index target segment, where *index-database-name* is the name of the secondary index database that contains the index pointer segment, which defines the index search field name.
- 23** The operator specifies the manner in which the contents of the search-field are to be tested against the comparative value.
- 24** Any number of search-field names can be specified in a QUALIFIED-ON clause, connected by the Boolean operators AND, OR, or INDEPENDENT-AND.
- 25** INDEPENDENT-AND is applicable only where the previous search field is the index-search-field-name and the following search field is the same index-search-field-name. It specifies that the call can be satisfied by two different index pointer segments (in the same secondary index) that both point to this index target segment, each satisfying one of the conditions; rather than requiring one index pointer segment that satisfies both of the conditions.
- 26** When the member containing the PROCESSES clause is encoded, if any member where the name appears in that member's data definition has no data entries record, a dummy data entries record is created for the latter member in accordance with the following rules:
- If the name appears in a CONTAINS clause that immediately follows PROCESSES IMS or PROCESSES DL/I (or a variant), a dummy PCB member is created.
  - If the name appears in a CONTAINS clause that does not immediately follow PROCESSES IMS or PROCESSES DL/I (or a variant), a dummy module member is created.
  - If the name immediately follows a SEGMENT keyword, a dummy segment member is created.
  - If the name appears anywhere in the QUALIFIED-ON clause, a dummy item member is created.
  - 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

The example in ["Application View" on page 13](#) 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 that defines no COMMAND-CODE, but does have a QUALIFIED-ON clause for two search fields.

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 nor QUALIFIED-ON clause.

This example shows a PROCESSES clause for an application requiring one PCB member, SKILLINV-PCB:

```
PROCESSES IMS
CONTAINS SKILLINV-PCS
SEGMENT-SEARCH-ARGUMENTS
  SEGMENT SKILMAST USED-IN SKILMAST-SSA
    QUALIFIED-ON SKILLCODE EQ
  SEGMENT SKILLNAM USED-IN SKILLNAM-SSA
    QUALIFIED-ON SURNAME EQ
    AND INITIAL EQ
  SEGMENT EXPRMAST USED-IN EXPRMAST-SSA
    COMMAND-CODE 10-MOVE  SEGMENT EDUCMAST USED-IN EDUCMAST-SSA
```

Segment-search-arguments are specified for four segments. The segment SKILMAST has a USED-IN clause defining a QUALIFIED-ON clause for a search field. The segment SKILLNAM has a USED-IN clause defining a QUALIFIED-ON clause for two search fields. For the segment EXPRMAST, the USED-IN clause defines the COMMAND-CODE 10-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 IMS (DL/I) Databases

This chapter includes these sections:

<b>Introduction</b> .....	139
<b>IMS (DL/I) Member-type Keywords</b> .....	139
<b>Condition Keywords for WHICH and WHAT Commands</b> .....	141
Examples .....	142
Member Type Interrogations .....	146
Interrogation Syntax .....	154
Alternative Verb Keywords .....	173

### Introduction

DataManager provides powerful facilities for documenting, interrogating, and processing the data definitions of the various types of IMS (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 ["IMS \(DL/I\) Member-type Keywords" on page 139](#)].
- Additional condition keywords in the WHICH and WHAT commands (see ["Condition Keywords for WHICH and WHAT Commands" on page 141](#)).

### IMS (DL/I) Member-type Keywords

The syntax of these DataManager commands:

BULK ENCODE  
BULK PRINT  
BULK REPORT  
GLOSSARY  
LIST  
PERFORM  
WHICH

They are defined in the *ASG-Manager Products Dictionary/Repository User's Guide*, 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 DBMS interface is included in the implementation of DataManager, then database members defined under any of the implemented interfaces are selected.

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

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

These are the additional member-type selection keywords:

IMS-DATABASES  
DL/I-DATABASES  
DL/I-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

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

SEQUENCE-KEYS  
IMS-DATASETS  
DL/I-DATASETS  
DL/I-DATASETS  
DLI-DATASETS  
DLI-DATASETS  
INDEX-SEARCH-FIELDS  
SYSTEM-RELATED-FIELDS  
CONCATENATED-KEYS  
CONCATENATED-KEY-NAMES

All of 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 IMS (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 by PERFORM commands.

## Condition Keywords for WHICH and WHAT Commands

The WHICH command enables the user to interrogate the data dictionary as to which members of selected types [see "[IMS \(DL/I\) Member-type Keywords](#)" on page 139], 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 through 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 IMS (DL/I) Interface provides further keywords for the condition clause.

The tables in this section give the following information on these keywords:

- The Member Type Interrogation table on page 146 explains which VIA keywords are appropriate for use with a particular IMS (DL/I) member type to interrogate various aspects of its definition.
- The Interrogation Syntax table on page 155 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 on page 174 offers alternative verb keywords that can be used instead of some USES and CONSTITUTES constructions.

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

$$\text{WHICH } \textit{selection} \left\{ \begin{array}{l} \left\{ \begin{array}{l} \text{USES} \\ \text{CONSTITUTES} \end{array} \right\} \textit{member-name VIA } \textit{via-keyword} \\ \textit{alternative-verb-keyword } \textit{member-name} \end{array} \right\} ;$$

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 in this form:

$$\text{WHICH } \left\{ \begin{array}{l} \text{MODULES} \\ \text{PROGRAMS} \\ \text{SYSTEMS} \end{array} \right\} \text{ USE } \left\{ \begin{array}{l} \textit{index-pointer-segment-name} \\ \textit{logical-segment-name} \\ \textit{physical-segment-name} \end{array} \right\} \text{ VIA SSAS};$$

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 IMS-specific member types.

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

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

Similarly, the alternative forms DL/1-DATASETS, DLI-DATASETS, DL1-DATASETS, and IMS-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 IMS 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, the following 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 was 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 in ["The Member Type for a HSAM Type IMS \(DL/I\) Database" on page 75](#), the response to this 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.

This command:

```
WHICH SEGMENTS DIRECTLY CONSTITUTE 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 this:

```
WHICH SEGMENTS FATHER JOB-TITLE;
```

and the second this:

```
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, this command:

```
WHICH PHYSICAL-SEGMENTS CONSTITUTE ASY-LINE VIA TO;
```

when used with the example in ["Physical Segments" 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 10](#), this 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. This 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 through its SEGMENT-SEARCH-ARGUMENTS clause. This could be achieved by this command:

```
WHICH SEGMENTS DIRECTLY CONSTITUTE process-member-name VIA SSAS;
```

Using the PROCESSES clause example in ["Syntax of the PROCESSES Clause" on page 132](#), the response would consist of the segments SKILMAST, SKILLNAM, EXPEMAST, and EDUGMAST.

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 ["Syntax of the PROCESSES Clause" on page 132](#), 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 PCBs and the sensitive segments and sensitive 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 ["Member-type Descriptions for IMS \(DL/I\) Program Communication Blocks" on page 117](#), this command:

```
WHICH SEGMENTS DIRECTLY CONSTITUTE ASY-PACK-PCB VIA SEGMENT;
```

would respond with the segments ASY-LINE and ASY-PACK.

This 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 1 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, this command:

```
WHICH PHYSICAL-SEGMENTS USE NUMBER-OF-LINES VIA BOUND;
```

would respond with the names of the physical segments with CONTAINS clauses that 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 IMS-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
GSAM-DATABASES	BOUND	Relationship between GSAM databases and variable length arrays specified, directly or indirectly, as groups or items in the CONTAINS clause of the database data definition statement.
	CONTAINS	Relationship between GSAM databases and group and item members contained directly or indirectly in the database.
	DL/I- DATASETS	Relationship between GSAM databases and the datasets that constitute the database (that is, the datasets specified in the DATASETS clause of the database data definition statement).
	IF	Relationship between GSAM databases and group and item members specified, in IF subordinate clauses, in the CONTAINS clause of the database data definition statement.
HDAM-DATABASES	ADD-TO	Relationship between HDAM or HIDAM HI DAM-DATABASES databases and dataset members specified in the ADD-TO subordinate clause of the database data definition statement.
	CONTAINS	Relationship between HDAM or HIDAM databases and segment members that are contained in the database.
	DL/I- DATASETS { FATHERS } { PARENTS }	Relationship between HDAM or HIDAM databases and the dataset 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
	$\left. \begin{array}{l} \text{RANDOMISING-} \\ \text{MODULES} \\ \text{RANDOMIZING-} \\ \text{MODULES} \end{array} \right\}$	<p>Hierarchical parent and child relationship between segments where the names are listed in the CONTAINS clause of the database data definition statement.</p> <p>Relationship between HDAM or HIDAM databases and module members. specified in the RANDOMISING-MODULES clause of the database data definition statement.</p>
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 dataset members that constitute the database (that is, the datasets specified in the DATASETS clause of the database data definition statement).
	$\left\{ \begin{array}{l} \text{FATHERS} \\ \text{PARENTS} \end{array} \right\}$	<p>Hierarchical parent and child relationship between segments where the names are listed in the CONTAINS clause of the database data definition statement.</p>
LOGICAL-DATABASES	CONTAINS	Relationship between LOGICAL databases and segment members contained directly or indirectly in the database.
	$\left\{ \begin{array}{l} \text{FATHERS} \\ \text{PARENTS} \end{array} \right\}$	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).

**Member Type Interrogation**

<b>Member Type</b>	<b>Keyword for Use in VIA Clause</b>	<b>Relationship Interrogated by Keyword</b>
	$\left\{ \begin{array}{l} \text{SHARES-WITH} \\ \text{SHARING-} \\ \text{WITH} \end{array} \right\}$	Relationship between secondary index databases and other secondary indexes sharing the same secondary index database (that is, secondary indexes with names that are specified in the SHARES-WITH or SHARING-WITH 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.
	CONCATENATED-KEY-NAMES	Relationship between index pointer NAMES segment and the name to be used for the concatenated key of its index target segment (that is, the name specified in the CONCATENATED-KEY-NAME clause of the segment data definition statement).
	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.

**Member Type Interrogation**

<b>Member Type</b>	<b>Keyword for Use in VIA Clause</b>	<b>Relationship Interrogated by Keyword</b>
	$\left\{ \begin{array}{c} \underline{\text{IN}} \\ \underline{\text{OF}} \end{array} \right\}$	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.
	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.

**Member Type Interrogation**

<b>Member Type</b>	<b>Keyword for Use in VIA Clause</b>	<b>Relationship Interrogated by Keyword</b>								
LOGICAL-SEGMENT	CONTAINS	Relationship between logical segments and physical segments contained by the segment.								
	<table border="0"> <tr> <td style="font-size: 3em; vertical-align: middle;">}</td> <td style="padding: 0 10px;">DATABASES</td> <td style="font-size: 3em; vertical-align: middle;">{</td> </tr> <tr> <td></td> <td style="text-align: center;">IN-</td> <td></td> </tr> <tr> <td></td> <td style="padding: 0 10px;">DATABASES</td> <td style="font-size: 3em; vertical-align: middle;">}</td> </tr> </table>	}	DATABASES	{		IN-			DATABASES	}
}	DATABASES	{								
	IN-									
	DATABASES	}								
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 CKxxxxx subordinate clause are included.								
	CONCATENATED-KEY-FIELDS (source segments only)	Relationship between physical segments and system related fields specified in the AS CXxxxxx 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.								
	EDIT-COMPRESSION-EXITS	Relationship between physical segments and module members specified in the EDIT-COMPRESSION-EXITS clause of the segment data definition statement.								

**Member Type Interrogation**

<b>Member Type</b>	<b>Keyword for Use in VIA Clause</b>	<b>Relationship Interrogated by Keyword</b>
	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.
	$\left\{ \begin{array}{c} \underline{\text{IN}} \\ \underline{\text{OF}} \end{array} \right\}$	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.

**Member Type Interrogation**

<b>Member Type</b>	<b>Keyword for Use in VIA Clause</b>	<b>Relationship Interrogated by Keyword</b>
	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.
PCB (GSAM type)	{ DATABASES IN- DATABASES }	Relationship between PCBs and the GSAM database named in the PCB data definition statement.
	NAME	Relationship between PCBs and the PCB specified in the NAME clause of the PCB data definition statement.
PCB (OUTPUT-MESSAGE type)	NAME	Relationship between PCBs and the PCB specified in the NAME clause of the PCB data definition statement.
PCB (STRUCTURE type)	{ DATABASES IN- DATABASES }	Relationship between PCBs and the database named in the DATABASE clause 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.

**Member Type Interrogation**

<b>Member Type</b>	<b>Keyword for Use in VIA Clause</b>	<b>Relationship Interrogated by Keyword</b>
	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:

$$\text{WHICH } selection \left\{ \begin{array}{l} \left\{ \begin{array}{l} \text{USES} \\ \text{CONSTITUTES} \end{array} \right\} member\text{-}name \text{ VIA } via\text{-}keyword \\ alternative\text{-}verb\text{-}keyword member\text{-}name \end{array} \right\} ;$$

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 can 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. Note that 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 174](#).

Interrogation  
Syntax

Keyword for use in the VIA clause	USES interrogations		CONSTITUTES interrogations		Explanation/Notes
	Meaningful member-type selection keywords	Meaningful member types for member-name	Meaningful member-type selection keywords	Meaningful member types for member-name	
<u>ADD-TO</u>	<u>HDAM-DATABASES</u> <u>HIDAM-DATABASES</u> <u>PHYSICAL-</u> <u>DATABASES</u> <u>DL/I-DATABASES</u> <u>DATABASES</u>	DL/I-DATASET	<u>DL/I-DATASETS</u>	HDAM-DATABASE HIDAM-DATABASE	USES: Obtains the name of the HDAM or HIDAM database that specifies, in the ADD-TO clause of its data definition, the DL/I- DATASET member specified in the member-name part of the interrogation.  CONSTITUTES: Obtains the name of the DL/I-DATASET member that is specified in the ADD-TO clause of the data definition of the HDAM or HIDAM database named in the member-name part of the interrogation.
<u>BOUND</u>	<u>GSAM-DATABASES</u> <u>DL/I-DATABASES</u> <u>DATABASES</u> <u>INDEX-POINTER-</u> <u>SEGMENTS</u> <u>PHYSICAL-</u> <u>SEGMENTS</u> <u>SEGMENTS</u>	ITEM	<u>ITEMS</u>	GSAME DATABASE INDEX-POINTER- SEGMENT PHYSICAL- SEGMENT	USES: Obtains the name of each GSAM database and/or index pointer segment and/or physical segment that contains a variable length array where the number of occurrences are 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 GSAM database, index pointer segment, or physical segment named in the member-name part of the interrogation.

## Interrogation Syntax

Keyword for use in the VIA clause	USES interrogations		CONSTITUTES interrogations		Explanation/Notes
	Meaningful member-type selection keywords	Meaningful member types for member-name	Meaningful member-type selection keywords	Meaningful member types for member-name	
<u>CONCATENATED-KEY-CONSTITUENTS</u>	<u>PHYSICAL-SEGMENTS</u> <u>SEGMENTS</u>	ITEM GROUP SEQUENCE-KEY	<u>ITEMS</u> <u>GROUPS</u> <u>SEQUENCE-KEYS</u>	PHYSICAL-SEGMENT  (source segments only)	<p>This keyword applies only to physical segments that are defined as index source segments.</p> <p>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 CKxxxxxx subordinate clause.</p> <p>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 CKxxxxxx subordinate clause are included in the response.</p>
<u>CONCATENATED-KEY- FIELDS</u>	<u>PHYSICAL-SEGMENTS</u> <u>SEGMENTS</u>	SYSTEM-RELATED-FIELD	<u>SYSTEM-RELATED- FIELDS</u>	PHYSICAL-SEGMENT (source segments only)	<p>This keyword only applies to physical segments that are defined as index source segments.</p> <p>USES: Obtains the name of the physical segment that specifies, in the AS CKxxxxxx 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 CKxxxxxx (see "<a href="#">Physical Segments</a>" on <a href="#">page 24</a>).</p> <p>CONSTITUTES: Obtains the name of each system related field that is specified in the AS CKxxxxxx subordinate clause of the data definition of the physical segment named in the member-name part of the interrogation.</p>

Interrogation  
Syntax

Keyword for use in the VIA clause	USES interrogations		CONSTITUTES interrogations		Explanation/Notes
	Meaningful member-type selection keywords	Meaningful member types for member-name	Meaningful member-type selection keywords	Meaningful member types for member-name	
<u>CONCATENATED- KEY- NAMES</u>	<u>PHYSICAL- SEGMENTS</u>  <u>INDEX-POINTER- SEGMENTS</u>  <u>SEGMENTS</u>	CONCATENATED- KEY-NAME	<u>CONCATENATED- KEY-NAME</u>	PHYSICAL- SEGMENT  INDEX-POINTER- SEGMENT	<p>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.</p> <p>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.</p>
<u>CONTAINS</u>	<u>GSAM-DATABASES</u>  <u>DL/I-DATABASES</u>  <u>DATABASES</u>	ITEM  GROUP	<u>ITEMS</u>  <u>GROUPS</u>	GSAM-DATABASE	<p>USES: Obtains the name of each GSAM database that specifies, in the CONTAINS clause of its data definition, the item or group named in the member-name part of the interrogation.</p> <p>CONSTITUTES: Obtains the name of each item and/or group specified in the CONTAINS clause of the data definition of the GSAM database named in the member-name part of the interrogation.</p> <p>Instead of using the CONTAINS keyword in the VIA clause, the verb keywords CONTAINS and CONTAINED-BY could be used (see <a href="#">"Interrogation Syntax" on page 154</a>).</p>

## Interrogation Syntax

Keyword for use in the VIA clause	USES interrogations		CONSTITUTES interrogations		Explanation/Notes
	Meaningful member-type selection keywords	Meaningful member types for member-name	Meaningful member-type selection keywords	Meaningful member types for member-name	
<u>CONTAINS</u>	<u>HDAM-DATABASE</u> <u>HIDAM-DATABASES</u> <u>HISAM-DATABASES</u> <u>HSAM-DATABASES</u> <u>PHYSICAL-DATABASES</u> <u>SHISAM-DATABASES</u> <u>SHSAM-DATABASES</u> <u>DL/I-DATABASES</u> <u>DATABASES</u>	PHYSICAL-SEGMENTS	<u>PHYSICAL-SEGMENTS</u>	HDAM-DATABASE HIDAM-DATABASE HISAM-DATABASE HSAM-DATABASE SHISAM-DATABASE SHSAM-DATABASE	<p>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.</p> <p>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.</p> <p>Instead of using the CONTAINS keyword in the VIA clause, the verb keywords CONTAINS and CONTAINED-BY could be used (see <a href="#">"Interrogation Syntax" on page 154</a>).</p>
<u>CONTAINS</u>	<u>LOGICAL-DATABASES</u> <u>DL/I-DATABASES</u> <u>DATABASES</u>	LOGICAL-SEGMENT PHYSICAL-SEGMENT	<u>LOGICAL-SEGMENT</u> <u>PHYSICAL-SEGMENTS</u>	LOGICAL-DATABASE	<p>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.</p> <p>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.</p> <p>Instead of using the CONTAINS keyword in the VIA clause, the verb keywords CONTAINS and CONTAINED-BY could be used (see <a href="#">"Interrogation Syntax" on page 154</a>).</p>

**Interrogation Syntax**

Keyword for use in the VIA clause	USES interrogations		CONSTITUTES interrogations		Explanation/Notes
	Meaningful member-type selection keywords	Meaningful member types for member-name	Meaningful member-type selection keywords	Meaningful member types for member-name	
<u>CONTAINS</u>	<u>SECONDARY-INDEX-DATABASES</u>  DL/I-DATABASES  DATABASES	INDEX-POINTER-SEGMENT	<u>INDEX-POINTER-SEGMENTS</u>  <u>SEGMENTS</u>	SECONDARY-INDEX-DATABASE  DATABASE	<p>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.</p> <p>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.</p> <p>Instead of using the CONTAINS keyword in the VIA clause, the verb keywords CONTAINS and CONTAINED-BY could be used (see <a href="#">"Interrogation Syntax" on page 154</a>).</p>
<u>CONTAINS</u>	<u>INDEX-POINTER-SEGMENTS</u>  <u>PHYSICAL-SEGMENTS</u>  <u>SEGMENTS</u>	ITEM  GROUP	<u>ITEMS</u>  <u>GROUP</u>	INDEX-POINTER-SEGMENT  PHYSICAL-SEGMENT	<p>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.</p> <p>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.</p> <p>Instead of using the CONTAINS keyword in the VIA clause, the verb keywords CONTAINS and CONTAINED-BY could be used (see <a href="#">"Interrogation Syntax" on page 154</a>).</p>

## Interrogation Syntax

Keyword for use in the VIA clause	USES interrogations		CONSTITUTES interrogations		Explanation/Notes
	Meaningful member-type selection keywords	Meaningful member types for member-name	Meaningful member-type selection keywords	Meaningful member types for member-name	
<u>CONTAINS</u>	<u>LOGICAL-SEGMENTS</u> <u>SEGMENTS</u>	PHYSICAL-SEGMENTS	<u>PHYSICAL-SEGMENTS</u> <u>SEGMENTS</u>	LOGICAL-SEGMENT	<p>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.</p> <p>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.</p> <p>Instead of using the CONTAINS keyword in the VIA clause, the verb keywords CONTAINS and CONTAINED-BY could be used (see <a href="#">"Interrogation Syntax" on page 154</a>).</p>
<u>CONTAINS</u>	<u>MODULES</u> <u>PROGRAMS</u> <u>SYSTEMS</u>	PCB PROGRAM-COMMUNICATION-BLOCKS	<u>PCBS</u> <u>PROGRAM-COMMUNICATION-BLOCKS</u>	MODULE PROGRAM SYSTEM	<p>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.</p> <p>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.</p> <p>Instead of using the CONTAINS keyword in the VIA clause, the verb keywords CONTAINS and CONTAINED-BY could be used (see <a href="#">"Interrogation Syntax" on page 154</a>).</p>

**Interrogation  
Syntax**

Keyword for use in the VIA clause	USES interrogations		CONSTITUTES interrogations		Explanation/Notes
	Meaningful member-type selection keywords	Meaningful member types for member-name	Meaningful member-type selection keywords	Meaningful member types for member-name	
<u>DATABASES</u>	<u>LOGICAL-SEGMENTS</u> <u>SEGMENTS</u>	HDAM-DATABASE HIDAM-DATABASE HISAM-DATABASE	<u>HDAM-DATABASES</u> <u>HIDAM-DATABASES</u> <u>HISAM-DATABASES</u> <u>PHYSICAL-DATABASES</u> <u>DL/I-DATABASES</u> <u>DATABASES</u>	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.
<u>DL/I-DATASETS</u>	<u>GSAM-DATABASES</u> <u>HDAM-DATABASE</u> <u>HIDAM-DATABASES</u> <u>HSAM-DATABASES</u> <u>HISAM-DATABASES</u> <u>SHSAM-DATABASES</u> <u>SHISAM-DATABASES</u> <u>PHYSICAL-DATABASES</u> <u>SECONDARY-INDEX- DATABASES</u> <u>DL/I-DATABASES</u> <u>DATABASES</u>	DL/I-DATASET	<u>DL/I-DATASETS</u>	GSAM-DATABASE HDAM-DATABASE HIDAM-DATABASE HISAM-DATABASE SHSAM-DATABASE SHISAM-DATABAS E  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		CONSTITUTES interrogations		Explanation/Notes
	Meaningful member-type selection keywords	Meaningful member types for member-name	Meaningful member-type selection keywords	Meaningful member types for member-name	
<u>DUPLICATE-DATA-FIELDS</u>	<u>INDEX-POINTER-SEGMENTS</u> <u>SEGMENTS</u>	ITEM GROUP SYSTEM-RELATED-FIELD	<u>ITEMS</u> <u>GROUPS</u> <u>SYSTEM-RELATED-FIELDS</u>	INDEX-POINTER-SEGMENT	<p>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.</p> <p>In a USES interrogation, any system-related field specified for member-name must be of the form CKxxxxxx (see "<a href="#">Segments that Reside in a Secondary Index Database</a>" on <a href="#">page 55</a>).</p> <p>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.</p>
<u>EDIT-COMPRESSION-EXITS</u>	<u>PHYSICAL-SEGMENTS</u> <u>SEGMENTS</u>	MODULE	<u>MODULES</u>	PHYSICAL-SEGMENT	<p>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.</p> <p>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.</p>

Interrogation  
Syntax

Keyword for use in the VIA clause	USES interrogations		CONSTITUTES interrogations		Explanation/Notes
	Meaningful member-type selection keywords	Meaningful member types for member-name	Meaningful member-type selection keywords	Meaningful member types for member-name	
<u>FATHERS</u>	<u>LOGICAL- SEGMENTS</u>  <u>PHYSICAL- SEGMENTS</u>  <u>SEGMENTS</u>	LOGICAL-SEGMENT  PHYSICAL- SEGMENT	<u>LOGICAL- SEGMENTS</u>  <u>PHYSICAL- SEGMENTS</u>  <u>SEGMENTS</u>	LOGICAL- SEGMENT  PHYSICAL- SEGMENT	<p>USES: Obtains the name of each segment that is a parent of the segment specified in the member-name part of the interrogation.</p> <p>CONSTITUTES: Obtain the name of each segment that is a dependent of the segment specified in the member-name part of the interrogation.</p> <p>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.</p> <p>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.</p>
<u>GENERATES</u>	<u>INDEX-POINTER- SEGMENTS</u>  <u>SEGMENTS</u>	ITEM  GROUP  INDEX-SEARCH- FIELD  SYSTEM-RELATED- FIELD  SEQUENCE-KEY  CONCATENATED- KEY	<u>ITEMS</u>  <u>GROUPS</u>  <u>INDEX-SEARCH- FIELDS</u>  <u>SYSTEM- RELATED-FIELDS</u>  <u>SEQUENCE-KEYS</u>  <u>CONCATENATED- KEYS</u>  <u>CONCATENATED- KEY-NAMES</u>	INDEX-POINTER- SEGMENT	<p>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.</p> <p>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.</p> <p>Instead of using the keyword GENERATES in the VIA clause, the verb keywords GENERATES and GENERATED-BY can be used (see <a href="#">"Interrogation Syntax" on page 154</a>).</p>

## Interrogation Syntax

Keyword for use in the VIA clause	USES interrogations		CONSTITUTES interrogations		Explanation/Notes
	Meaningful member-type selection keywords	Meaningful member types for member-name	Meaningful member-type selection keywords	Meaningful member types for member-name	
<u>GENERATES</u>	<u>PHYSICAL-SEGMENTS</u> <u>SEGMENTS</u>	ITEM GROUP SEQUENCE-KEY CONCATENATED-KEY	<u>ITEMS</u> <u>GROUPS</u> <u>SEQUENCE-KEYS</u> <u>CONCATENATED-KEYS</u> <u>CONCATENATED-KEY-NAMES</u>	PHYSICAL-SEGMENT	<p>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.</p> <p>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.</p> <p>Instead of using the keyword GENERATES in the VIA clause, the verb keywords GENERATES and GENERATED-BY can be used (see <a href="#">"Interrogation Syntax" on page 154</a>).</p>
<u>IF</u>	<u>GSAM-DATABASES</u> <u>DL/I-DATABASES</u> <u>DATABASES</u> <u>INDEX-POINTER-SEGMENTS</u> <u>PHYSICAL-SEGMENTS</u> <u>SEGMENTS</u>	ITEM GROUP	<u>ITEMS</u> <u>GROUPS</u>	GSAM-DATABASE INDEX-POINTER-SEGMENT PHYSICAL-SEGMENT	<p>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.</p> <p>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.</p>

Interrogation  
Syntax

Keyword for use in the VIA clause	USES interrogations		CONSTITUTES interrogations		Explanation/Notes
	Meaningful member-type selection keywords	Meaningful member types for member-name	Meaningful member-type selection keywords	Meaningful member types for member-name	
<u>IN</u>	<u>INDEX-POINTER- SEGMENTS</u>  <u>SEGMENTS</u>	GROUP  SYSTEM-RELATED- FIELD  INDEX-SEARCH- FIELD  SEQUENCE-KEY  CONCATENATED- KEY	<u>GROUPS</u>  <u>SYSTEM-RELATED - FIELDS</u>  <u>INDEX-SEARCH- FIELDS</u>  <u>SEQUENCE-KEYS</u>  <u>CONCATENATED- KEYS</u>  <u>CONCATENATED- KEY-NAMES</u>	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.
<u>IN-DATABASES</u>					See <a href="#">"DATABASES" on page 161.</a>
<u>MAINTENANCE- EXITS</u>	<u>INDEX-POINTER- SEGMENTS</u>  <u>SEGMENTS</u>	MODULE	<u>MODULES</u>	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.
<u>NAME</u>	<u>PCBS</u>  <u>PROGRAM- COMMUNICATION- BLOCKS</u>	PCB  PROGRAM- COMMUNICATION- BLOCK	<u>PCBS</u>  <u>PROGRAM- COMMUNICATION- BLOCKS</u>	<u>PCB</u>  <u>PROGRAM- COMMUNICATION- BLOCK</u>	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.
<u>OF</u>					See <a href="#">"IN" on page 165.</a>

Interrogation  
Syntax

Keyword for use in the VIA clause	USES interrogations		CONSTITUTES interrogations		Explanation/Notes
	Meaningful member-type selection keywords	Meaningful member types for member-name	Meaningful member-type selection keywords	Meaningful member types for member-name	
<u>ON</u>	<u>INDEX-POINTER- SEGMENTS</u>  <u>SEGMENTS</u>	<u>INDEX-SEARCH- FIELD</u>	<u>INDEX-SEARCH- FIELDS</u>	<u>INDEX-POINTER- SEGMENTS</u>	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 <a href="#">"FATHERS" on page 163.</a>
<u>QUALIFIED-ON</u>	<u>MODULES</u>  <u>PROGRAMS</u>  <u>SYSTEMS</u>	ITEM  GROUP  <u>INDEX-SEARCH- FIELD</u>  SEQUENCE-KEY  CONCATENATED- KEY	<u>ITEMS</u>  <u>GROUPS</u>  <u>INDEX-SEARCH- FIELDS</u>  <u>SEQUENCE-KEYS</u>  <u>CONCATENATED- KEYS</u>  <u>CONCATENATED- KEY- NAMES</u>	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.
	<u>HDAM-DATABASES</u>  <u>PHYSICAL- DATABASES</u>  <u>DL/I-DATABASES</u>  <u>DATABASES</u>	MODULE	<u>MODULES</u>	<u>HDAM-DATABASES</u>	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.

**Interrogation Syntax**

Keyword for use in the VIA clause	USES interrogations		CONSTITUTES interrogations		Explanation/Notes
	Meaningful member-type selection keywords	Meaningful member types for member-name	Meaningful member-type selection keywords	Meaningful member types for member-name	
<u>RENAMES</u>	<u>PHYSICAL-SEGMENTS</u> <u>SEGMENTS</u>	ITEM GROUP SEQUENCE-KEY	<u>ITEMS</u> <u>GROUPS</u> <u>SEQUENCE-KEYS</u>	PHYSICAL-SEGMENT (logical child segments only)	<p>This keyword is only applicable to logical child segments.</p> <p>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.</p> <p>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.</p>
<u>SEARCH-KEY-FIELDS</u>  <div style="display: inline-block; vertical-align: middle;"> <span style="font-size: 2em;">}</span> <span style="display: inline-block; vertical-align: middle; padding: 0 5px;">RANDOMISING-MODULES</span>  <span style="font-size: 2em;">}</span> <span style="display: inline-block; vertical-align: middle; padding: 0 5px;">RANDOMIZING-MODULES</span> </div>	<u>INDEX-POINTER-SEGMENTS</u> <u>SEGMENTS</u>	ITEM GROUP	<u>ITEMS</u> <u>GROUPS</u>	INDEX-POINTER-SEGMENT	<p>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.</p> <p>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.</p>
<u>SECONDARY-SEQUENCE-ON</u>	<u>PROGRAM-COMMUNICATION-BLOCKS</u> <u>PCBS</u>	INDEX-POINTER-SEGMENT	<u>INDEX-POINTER-SEGMENTS</u> <u>SEGMENTS</u>	PCB (structure type)	<p>This keyword is only applicable to structure type PCBs.</p> <p>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.</p> <p>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.</p>

## Interrogation Syntax

Keyword for use in the VIA clause	USES interrogations		CONSTITUTES interrogations		Explanation/Notes
	Meaningful member-type selection keywords	Meaningful member types for member-name	Meaningful member-type selection keywords	Meaningful member types for member-name	
<u>SEGMENT</u>	<u>PROGRAM-COMMUNICATION-BLOCKS</u> <u>PCBS</u>	<u>INDEX-POINTER-SEGMENT</u> <u>LOGICAL-SEGMENT</u> <u>PHYSICAL-SEGMENT</u>	<u>INDEX-POINTER-SEGMENTS</u> <u>LOGICAL-SEGMENTS</u> <u>PHYSICAL-SEGMENTS</u>	PCB (structure type)	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.
<u>SENSITIVE-FIELDS</u>	<u>PROGRAM-COMMUNICATION-BLOCKS</u> <u>PCBS</u>	<u>ITEM</u> <u>GROUP</u> <u>SEQUENCE-KEY</u> <u>CONCATENATED-KEY</u>	<u>ITEMS</u> <u>GROUPS</u> <u>SEQUENCE-KEYS</u> <u>CONCATENATED-KEYS</u> <u>CONCATENATED-KEY-NAMES</u>	PCB (structure type)	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.

**Interrogation  
Syntax**

Keyword for use in the VIA clause	USES interrogations		CONSTITUTES interrogations		Explanation/Notes
	Meaningful member-type selection keywords	Meaningful member types for member-name	Meaningful member-type selection keywords	Meaningful member types for member-name	
<u>SEQUENCE-KEY-</u> <u>CONSTITUENTS</u>	<u>PHYSICAL-</u> <u>SEGMENTS</u>  <u>SEGMENTS</u>	ITEM  GROUP	<u>ITEMS</u>  <u>GROUPS</u>	PHYSICAL- SEGMENT  (logical child segments only)	<p>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.</p> <p>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.</p> <p>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.</p>

## Interrogation Syntax

Keyword for use in the VIA clause	USES interrogations		CONSTITUTES interrogations		Explanation/Notes
	Meaningful member-type selection keywords	Meaningful member types for member-name	Meaningful member-type selection keywords	Meaningful member types for member-name	
<u>SEQUENCE-KEYS</u>	<u>LOGICAL-SEGMENTS</u> <u>PHYSICAL-SEGMENTS</u> <u>SEGMENTS</u>	ITEM GROUP SEQUENCE-KEY	<u>ITEMS</u> <u>GROUPS</u> <u>SEQUENCE-KEYS</u>	LOGICAL-SEGMENT PHYSICAL-SEGMENTS	<p>USES: Obtains the name of each physical segment that specifies, in the SEQUENCE-KEY clause of its data definition:</p> <ul style="list-style-type: none"> <li>The item or group named in the member-name part of the interrogation, if an AS subordinate clause is not present, or</li> <li>The sequence key internal member named in the member-name part of the interrogation, if an AS subordinate clause is present.</li> </ul> <p>CONSTITUTES: Obtains the name of one of these:</p> <ul style="list-style-type: none"> <li>Each item or group specified, where no AS subordinate clause is present, or</li> <li>Each sequence key internal member specified, when an AS subordinate clause is present</li> </ul> <p>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.</p>
<u>SEQUENCE-KEYS</u>	<u>INDEX-POINTER-SEGMENTS</u> <u>SEGMENTS</u>	SEQUENCE-KEYS	<u>SEQUENCE-KEYS</u>	INDEX-POINTER-SEGMENTS	<p>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.</p> <p>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.</p>

Interrogation  
Syntax

Keyword for use in the VIA clause	USES interrogations		CONSTITUTES interrogations		Explanation/Notes
	Meaningful member-type selection keywords	Meaningful member types for member-name	Meaningful member-type selection keywords	Meaningful member types for member-name	
	<u>SECONDARY-INDEX- DATABASES</u>  <u>DL/I-DATABASES</u>  <u>DATABASES</u>	<u>SECONDARY-INDEX- -DATABASE</u>	<u>SECONDARY-INDE X-DATABASES</u>  <u>DL/I-DATABASES</u>  <u>DATABASES</u>	<u>SECONDARY- INDEX-DATABASE</u>	USES: Obtains the name of each secondary index database that specifies, in the SHARES-WITH or SHARING-WITH clause of its data definition, the secondary index database named in the member-name part of the interrogation.  CONSTITUTES: Obtains the name of the secondary index database, that is specified in the SHARES-WITH or SHARING-WITH clause of the data definition of the secondary index database named in the member-name part of the interrogation.
<u>SOURCE</u>	<u>INDEX-POINTER- SEGMENTS</u>  <u>SEGMENTS</u>	<u>PHYSICAL- SEGMENT</u>  (source segment only)	<u>PHYSICAL- SEGMENTS</u>  <u>SEGMENTS</u>	<u>INDEX-POINTER- SEGMENT</u>	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
<u>SSAS</u>	<u>MODULES</u>  <u>PROGRAMS</u>  <u>SYSTEMS</u>	<u>INDEX-POINTER- SEGMENT</u>  <u>LOGICAL-SEGMENT</u>  <u>PHYSICAL- SEGMENT</u>	<u>INDEX-POINTER- SEGMENT</u>  <u>LOGICAL- SEGMENTS</u>  <u>PHYSICAL- SEGMENTS</u>  <u>SEGMENTS</u>	<u>MODULE</u>  <u>PROGRAM</u>  <u>SYSTEM</u>	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.

## Interrogation Syntax

Keyword for use in the VIA clause	USES interrogations		CONSTITUTES interrogations		Explanation/Notes
	Meaningful member-type selection keywords	Meaningful member types for member-name	Meaningful member-type selection keywords	Meaningful member types for member-name	
<u>SUBSEQUENCE-FIELDS</u>	<u>INDEX-POINTER-SEGMENTS</u>	ITEM GROUP SYSTEM-RELATED-FIELD	<u>ITEMS</u> <u>GROUPS</u> <u>SYSTEM-RELATED-FIELDS</u>	<u>INDEX-POINTER-SEGMENT</u>	<p>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.</p> <p>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.</p>
TARGET  { <u>SHARES-WITH</u> <u>SHARING-</u> WITH }	<u>INDEX-POINTER-SEGMENTS</u> <u>SEGMENTS</u>	PHYSICAL-SEGMENTS  (target segments only)	<u>PHYSICAL-SEGMENTS</u> <u>SEGMENTS</u>	<u>INDEX-POINTER-SEGMENT</u>	<p>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.</p> <p>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.</p>
<u>TO</u>	<u>PHYSICAL-SEGMENTS</u> <u>SEGMENTS</u>	PHYSICAL-SEGMENT  (destination parent segments only)	<u>PHYSICAL-SEGMENTS</u> <u>SEGMENTS</u>	PHYSICAL-SEGMENT  (logical child segments only)	<p>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.</p> <p>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.</p>

**Interrogation  
Syntax**

Keyword for use in the VIA clause	USES interrogations		CONSTITUTES interrogations		Explanation/Notes
	Meaningful member-type selection keywords	Meaningful member types for member-name	Meaningful member-type selection keywords	Meaningful member types for member-name	
<u>WITH</u>	<u>PHYSICAL- SEGMENTS</u>  <u>SEGMENTS</u>	PHYSICAL- SEGMENT  (logical child segments only)	<u>PHYSICAL- SEGMENTS</u>  <u>SEGMENTS</u>	PHYSICAL- SEGMENT  (logical child segments only)	<p>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).</p> <p>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.</p>

## 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, this interrogation:

WHICH *selection* FATHERS *member-name*

is equivalent to this interrogation:

WHICH *selection* USES *member-name* VIA  $\left\{ \begin{array}{l} \text{PARENTS} \\ \text{FATHERS} \end{array} \right\}$

The equivalences are shown in this table:

<b>Alternative Verb Keyword</b>	<b>Equivalent USES/CONSTITUTES Interrogation</b>
CONTAINS <i>member-name</i>	USES <i>member-name</i> VIA CONTAINS
CONTAINED-BY <i>member-name</i>	CONSTITUTES <i>member-name</i> VIA CONTAINS
FATHERS <i>member-name</i>	USES <i>member-name</i> VIA $\left\{ \begin{array}{l} \text{PARENTS} \\ \text{FATHERS} \end{array} \right\}$
FATHERED-BY <i>member-name</i>	CONSTITUTES <i>member-name</i> VIA $\left\{ \begin{array}{l} \text{PARENTS} \\ \text{FATHERS} \end{array} \right\}$
GENERATES <i>member-name</i>	USES <i>member-name</i> VIA GENERATES
GENERATED-BY <i>member-name</i>	CONSTITUTES <i>member-name</i> VIA GENERATES

---

# 5

## IMS (DL/I) Source Language Generation

---

This chapter includes these sections:

<b>Introduction</b> . . . . .	176
<b>Generating IMS (DL/I) DBD Control Statements</b> . . . . .	176
<b>Generating IMS (DL/I) PSB Control Statements</b> . . . . .	183
<b>Generation of COBOL, PL/I, or Assembler Data Description Statements for Segment Input/Output Areas</b> . . . . .	189
The PRODUCE Command . . . . .	189
Installation Macros . . . . .	189
Segment Input/Output Areas: Items Defined as BINARY or BITS . . . . .	190
Simple Physical Segments . . . . .	190
Logical Child Segments . . . . .	190
Destination Parent Segments . . . . .	191
Index Target and Index Source Segments . . . . .	191
Logical Segments and Logical Concatenated Segments . . . . .	192
Variable Length Segments . . . . .	192
Path Calls . . . . .	194
Index Pointer Segments . . . . .	194
Miscellaneous IMS (DL/I) Fields . . . . .	198
<b>Generation of COBOL, PL/I, or Assembler Data Description Statements for Segment Sensitive Fields Input/Output Areas</b> . . . . .	198
<b>Generation of COBOL, PL/I, or Assembler Data Description Statements for PCB Masks</b> . . . . .	200
<b>Generation of COBOL, PL/I, or Assembler Data Description Statements for Segment Search Arguments</b> . . . . .	203

## Introduction

The Source Language Generation Facility can produce IMS (DL/I) statements of these types:

- IMS (DL/I) DBD control statements, which can subsequently be used as input for an IMS (DL/I) DBD generation
- IMS (DL/I) PSB 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
- Record layouts and or COBOL, PL/I, or Assembler data description statements for users' segment sensitive fields 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

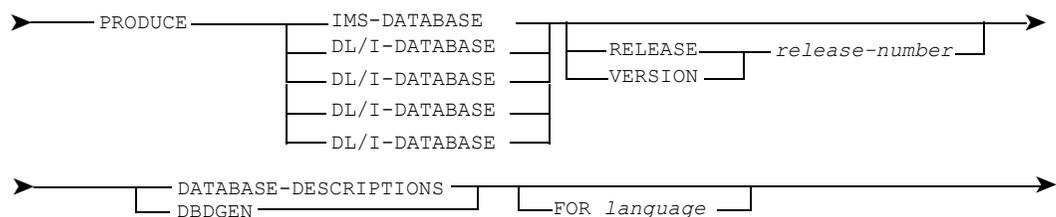
Generation of these statements is achieved by use of the PRODUCE command, described in the publication *ASG-Manager Products Source Language Generation*. The variations of the PRODUCE command required for the generations listed above are described in this chapter.

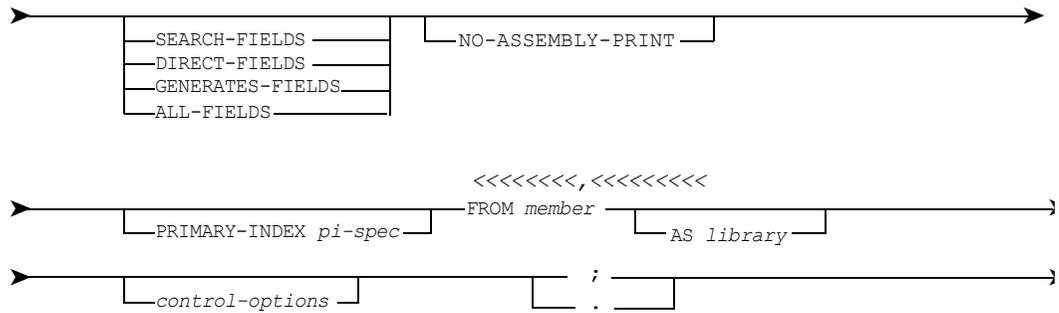
The PRODUCE command can also be used to generate MARK IV File Definition forms from encoded IMS-DATABASE (DL/I-DATABASE) and SEGMENT members. The use of the PRODUCE command for this purpose is documented in the *ASG-DataManager MARK IV Interface* publication.

## Generating IMS (DL/I) DBD Control Statements

The installation macro DGDBD allows you to tailor generated IMS (DL/I) DBD control statements to your own requirements. This macro is described in ["The Macros DGDBD And DGPSB" on page 210](#).

### Syntax



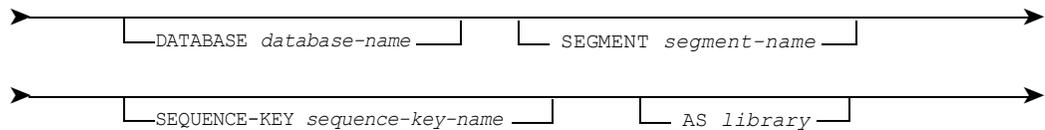


where:

*rel-number* is 1.2, 1.3, 2, 2.0, 2.1, 2.2, 3, 3.0, 3.1, 4, 4.1, 5, or 5.0.

*language* is COBOL, COBOL2, COBOL-2, COBOLII, COBOL-II, PL/I, PL/1, PLI, PL1, PL/IF, PL/1F, PLIF, or PL1F.

*pi-spec* is:



*database-name*, *segment-name*, and *sequence-key-name* are valid IMS (DL/I) names.

*library* is a string of up to 16 characters. The first character must be: #, alphabetic, local currency symbol (internal code hexadecimal 5B), %, or @.

*member* is an IMS-DATABASE or DL/I-DATABASE member.

*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/ALIAS options are valid in the OMITTING clause
- If you specify NO-GENERATION *and* NO-PRINT, no processing occurs

## 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.

RELEASE/VERSION: By default, IMS Version 2 is assumed, unless the supplied macro DGDBD has been tailored. Any value specified with this keyword overrides the default specified in DGDBD.

- 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 178](#) through [remark 12 on page 180](#), and [remark 20 on page 181](#) through [remark 22 on page 182](#) 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 where the 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 178](#) and [remark 6 on page 178](#).
- 9 DIRECT-FIELDS specifies that DBD FIELD control statements are to be generated for the fields described in [remark 5 on page 178](#) and [remark 6 on page 178](#), 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 178](#) 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 178](#).

- 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 where the 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 HDAM or HIDAM database, if a DATASETS clause in the member's data definition contains an ADD-TO clause, the DATASET control statement generated from it has no operands, but is labelled with the ddname stated in the clause. The same label is also generated for the DATASET control statement that contains the operands defining the dataset group.
- 18** 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 are not specified in the command, but are 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 24 on page 182](#)), and generation of the member containing the control statements does not take place.

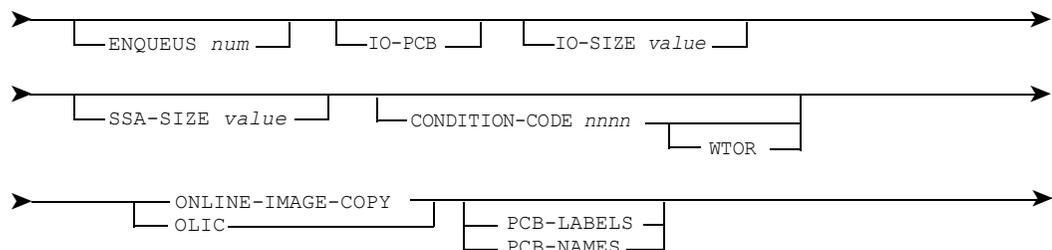
In order to avoid this situation, a valid library name can be specified in the AS library-name subordinate clause of the PRIMARY-INDEX clause, or the MEMLEN parameter of the DGDBD tailoring macro can be used to extend the permissible length of library names (see ["The Macros DGDBD And DGPSB" on page 210](#)).

- 19** For a shared SECONDARY-INDEX database, the member-name in the FROM clause must be that of the index database that is being shared; that is, its definition must contain a DATASETS clause, not a SHARES-WITH clause.
- 20** 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.
- 21** 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.

- 22** The PRIMARY-INDEX clause specifies, in respect of a HIDAM database named in the FROM clause, user names that are to be applied (instead of the DataManager-generated names defined in [remark 18 on page 180](#)) 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 generated library member
- 23** AS clauses are relevant only if DBD control statements are being written to an output dataset.
- 24** 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 catalogued in the output source library dataset.
- 25** 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.
- 26** 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.
- 27** 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 210](#)).
- 28** 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.
- 29** 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 file OMITTING clause may only specify OMITTING KNOWN-AS and/or ALIAS.



options are:



where:

*num* is an unsigned integer.

*value* is an unsigned integer not greater than 256000.

*nnnn* is an unsigned integer in the range 0 to 4095.

*mem* is the name of a SYSTEM, PROGRAM MODULE, or MMR-SYSTEM member.

*library* is a string of up to 16 characters. The first character must be: #, alphabetic, local currency symbol (internal code hexadecimal 5B), %, or @.

*control-options* is as defined in ["Generating IMS \(DL/I\) DBD Control Statements" on page 176](#).

### Remarks

- 1 The first elements in the command must be the command identifier, PRODUCE, followed by the context keyword IMS. Next is optionally the RELEASE clause, then one of the context qualifier keywords, PROGRAM-SPECIFICATION or PSBGEN. These must be in the order shown; control-options can be in any order.
- 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 RELEASE/VERSION clause specifies a version of IMS to produce statements in accordance with the stated version. By default, IMS Version 2 is assumed. Use of this keyword overrides the default specified by IMSLVL in macro DGPSB.
- 4 The optional keyword NO-ASSEMBLY-PRINT and the optional OPTIONS clause must, if present, precede the FROM clause.
- 5 If NO-ASSEMBLY-PRINT is specified in the command, an Assembler PRINT NOGEN statement is generated, to eliminate listing of the PSB control statements when they are assembled.

- 6 The ENQUEUES clause specifies the maximum number of database calls with the IMS (DL/I) command code Q (corresponding to the DataManager command code ENQUEUE), which may be issued between synchronization points. If this number is exceeded, the application program will ABEND.
- 7 IO-PCB specifies that IMS (DL/I) is to add an I/O PCB for the input message to the PSB, even if the program is to run in the Batch-DL/I region. (An I/O PCB is always added for the input message if the program runs in the BMP or MSG region.)
- 8 The IO-SIZE clause enables the user to specify the largest size of input/output area that can be used by the application program. If the clause is omitted, the IMS (DL/I) ACB utility program calculates a maximum size to be used as default.
- 9 The SSA-SIZE clause enables the user to specify the maximum total length of all SSAs to be used by the application program. If the clause is omitted, the IMS (DL/I) ACB utility program calculates a maximum size to be used as default.
- 10 The CONDITION-CODE clause is applicable only in batch type regions. It specifies the condition code that is to be returned to the operating system when IMS (DL/I) terminates normally, and one or more input/output errors have occurred on any database during the application program execution. This enables the user to set a unique operating system condition code when an input/output error occurs and to test the condition code in subsequent job steps. If the clause is not specified, the return code passed from the application program is passed to the operating system and status codes, and console messages are the only indicators of database input/output errors.
- 11 If WTOR is specified, a WTOR for the DFSnnnnA input/output error message is issued, and IMS (DL/I) waits for the operator to respond before continuing. A response of ABEND causes IMS (DL/I) to terminate; a response of CONT causes IMS (DL/I) to continue.
- 12 The ONLINE-IMAGE-COPY and OLIC keywords are synonymous. Either specifies that the user of this PSB is authorized to execute the Online Database Image Copy utility or the Surveyor utility feature.
- 13 The PSBNAME applied to the generated PSB control statements is the SYSTEM, PROGRAM, or MODULE member name.
- 14 The operands for the PSBGEN control statement are obtained from the OPTIONS clause specified in the PRODUCE command, as described in the remarks above. The language operand 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

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.

- 15** If generation is for IMS versions prior to 4, up to 255 occurrences of PCB control statements will be generated. Otherwise, up to 500 occurrences of PCB control statements will be generated.
- 16** PCB control statements are generated in the correct sequence for each PCB member that has been specified in the PROCESSES clause. That is, first alternate PCBs for each of the output-message-destination PCB members, then database PCBs for each of the logical-data-structure PCB members and finally database PCBs for each of the GSAM-database PCB members. Within each type of PCB, statements are generated in the order in which the PCB members are specified.
- 17** For the PCB for a logical-data-structure, if KEYLENGTH has not been specified in the PCB definition, then the value of the KEY LENGTH operand is calculated by DataManager as the length of the largest concatenated key for all data-sensitive segments specified in the relevant member.
- 18** The PROCSEQ operand is generated by DataManager if one of the SEGMENT clauses specified for the PCB member contains the keyword SECONDARY-SEQUENCE.
- 19** 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

If generation is for IMS versions prior to 4, up to 1000 occurrences of SENSEG control statements will be generated. Otherwise, up to 3000 occurrences of SENSEG control statements will be generated.

- 20** Under the following circumstances, DataManager produces SENSEG statements for dependent segments of a target segment's parent segments:
- One of the SEGMENT clauses specified for the PCB member contains the keyword SECONDARY-SEQUENCE.
  - The target segment is not the root segment of the relevant database.
  - The dependent segments of the target segment's parent segments are within the scope of the segments specified in the PCB structure definitions.
- 21** Sibling segments may be rearranged to maintain the PCB segment order by specifying the KEEP-HIERARCHY keyword.
- 22** The INDICES operand is generated by DataManager if the SEGMENT clause is for an index target segment or a logical segment representing an index target segment, and contains any USED-IN clauses which name index-field-name fields (XDFLDs) for search fields.
- 23** 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 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
- 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.
- 24** All names generated are subject to any editing specified in the command.
- 25** AS clauses are relevant only if DBD control statements are being written to an output dataset.
- 26** 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 catalogued in the output source library dataset.
- 27** 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.

- 28** 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.
- 29** 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 ["The Macros DGDBD And DGPSB" on page 210](#)).
- 30** The USE or USING 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 is obtained from the containing SEGMENT data definition.
- 31** Other control-options clauses are as specified in the *ASG-Manager Products Source Language Generation* manual, except that the GIVING clause may only specify KNOWN-AS and the OMITTING clause may only specify OMITTING KNOWN-AS and/or ALIAS.
- 32** If GIVING KNOWN-AS is specified, generated data names are based on the KNOWN-AS clauses specified for sensitive fields 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.

- 33** PCB-LABELS specifies that for each PCB statement generated, there should be a label in columns 1 to 8. This label is generated from the PCB member name or its alias, if one is specified on the PRODUCE command.
- 34** PCB-NAME specifies that for each PCB statement generated, there should be a PCBNAME= clause. This clause is generated from the PCB member name or its alias, if one is specified on the PRODUCE command.
- 35** These two apply to all PCB member types (DATABASE, OUTPUT-MESSAGE, and GSAM).
- 36** If a PCB contains a AIB-LIST-ADDRESS specified as NO, and neither of these keywords is specified then the PCB-NAMES option is assumed and a warning message is issued.

## 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 IMS (DL/I) fields for which data dictionary members of special internal types exist. See ["Miscellaneous IMS \(DL/I\) Fields" on page 198](#). 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 Macros DGSCOB, DGSPLI, DGSBAL, and DGSREC" on page 213](#).

The data description statements that are generated for the various types of segments are described in ["Simple Physical Segments" on page 190](#) through ["Miscellaneous IMS \(DL/I\) Fields" on page 198](#).

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 Macros DGSCOB, DGSPLI, DGSBAL, and DGSREC" on page 213](#).

The installation macros DGCOD, DOPLI, DGBAL, and DGREC, described in the *ASG-Manager Products Source Language Generation* manual, also apply, respectively, when segment input/output area data descriptions are generated in COBOL, PL/I, 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 1-byte binary item is rounded up to 2 bytes in length
- A 3-byte binary item is rounded up to 4 bytes in length
- A 5-, 6-, or 7-byte binary item is rounded up to 8 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
- An index pointer segment
- An index source segment
- 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
- An index pointer segment, by virtue of the index target segment's concatenated key
- 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 (e.g., in CONTAINS clauses and in concatenated keys) or in different languages (e.g., 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 IMS \(DL/I\) Fields" on page 198](#).

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 which 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 IMS \(DL/I\) Fields" on page 198](#).

## **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 FIELDDD
    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 (ITEMS) 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 2-byte sized 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  LDGICAL-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  FIELDDD
        05  KEYC
        05  FIELDE
```

If both parts of a logical concatenated segment are variable length, then the 2-byte sized 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.

<b>Data Description Statements</b>	<b>See Remark Number:</b>
01 INDEX-POINTER-SEGMENT-NAME	1
03 KEY-NAME	2
05 CONSTANT-NAME	3
05 INDEX-FIELD-NAME	2, 4
07 FIELD-A	
07 FIELD-B	
07 FIELD-C	
05 SUBSEQUENCE-NAME	5
07 CKA	6
09 KEYA	
09 FIELD-D	
05 SXA	7
07 CKB	8
09 KEYB	
03 DUPLICATE-DATA-NAME	9
05 CKA	
07 KEYA	
07 FIELD-D	
05 CKB	
07 KEYB	
05 CKC	

Data Description Statements	See Remark Number:
07 FIELD-E	
03 CONCATENATED-KEY-NAME	10
05 KEYA	
05 KEYB	
05 KEYC	
05 KEYD	
03 USER-DATA-NAME	11
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** CONSTANT-NAME is obtained from the macro DOSCOB, DGSPLI, or DGSBAL, as appropriate. It is generated only if a CONSTANT field is defined for the index pointer segment.
- 4** 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.
- 5** 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, which are specified in the index pointer segment's definition to constitute the subsequence fields.
- 6** 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.
- 7** This is a system related field of the type that prompts IMS (DL/I) to generate a unique 4-byte value.

- 8 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.
- 9 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, which are specified in the index pointer segment's definition to constitute the duplicate-data fields.
- 10 CONCATENATED-KEY-NAME is obtained either from the name specified in the CONCATENATED-KEY-NAME clause of the segment, if specified, or from the macro DGSCOB, DGSPLI, or DGSBAL, as appropriate. The field contains the concatenated key fields of the related index target segment. The concatenated key is automatically constructed by DataManager if it is not contained in the subsequence or duplicate-data fields, and symbolic pointing is specified for the index pointer segment.
- 11 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 with names that are unique.

When Assembler data description statements are generated, each of the fields constituting the index-field-data, subsequence-data, duplicate-data and the IMS (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, DUPLICATE-DATA-NAME, or CONCATENATED-KEY-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, include editing clauses in the PRODUCE commands.

## Miscellaneous IMS (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 IMS (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 DGC OB, 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 and the related example PCB member ASY-PACK-PCB shown in ["Member-type Descriptions for IMS \(DL/I\) Program Communication Blocks" on page 117](#) (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 8 bytes and a CHARACTER form-description, to receive the database name returned by IMS (DL/I).
- An ITEM member with a length of 2 bytes and a CHARACTER form-description, to receive the segment level number returned by IMS (DL/I).
- An ITEM member with a length of 2 bytes and a CHARACTER form-description, to receive the status code returned by IMS (DL/I).
- An ITEM member with a length of 4 bytes and a CHARACTER form-description, to contain the list of processing options required by IMS (DL/I).
- An ITEM member with a length of 4 bytes and a BINARY form-description, to be used by IMS (DL/I) for internal linkage.
- An ITEM member with a length of 8 bytes and a CHARACTER form-description, to contain the segment name returned by IMS (DL/I).
- An ITEM member with a length of 4 bytes and a BINARY form-description, to contain the length of the key feedback area.
- An ITEM member with a length of 4 bytes and a BINARY form-description, to receive the figure returned by IMS (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 IMS (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-SENSESEG, 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-SENSESEG
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-NAMEPIC X(8) .
  02 SEG-LEVELPIC XX.
  02 STAT-CODEPIC XX.
  02 PROC-OPTPIC X(4) .
  02 FILLERPIC S9(9)COMP.
  02 LEN-KFBPIC S9(9)COMP.
  02 NU-SENSEGPIC S9(9)COMP
  02 KEY-FBPIC 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 DS0CL136
DBNAME DSCL8
SEGLEVEL DSCL2
STATCODE DSCL2
PROCOPT DSCL4
FILLER DSFL4
SEGNAME DSCL8
LENKFB DSFL4
NUSENSEG DSFL4
KEYFB DSCL100
* 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 ["Syntax of the PROCESSES Clause" on page 132](#). 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 8 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 1 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 1 to 4 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 1 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 1 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 8 bytes, a CHARACTER form-description, and a CONTENTS clause that specifies the name of the search field.
- An ITEM member with a length of 2 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 1 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 1 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:

<b>Segment Name</b>	<b>*</b>	<b>Command Code</b>	<b>Begin QS</b>	<b>Field Name</b>	<b>R.O.</b>	<b>Value</b>	<b>End QS</b>
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:

```
ADD TEST-SSA;
GROUP
CONTAINS SSEGNAME, SCCSEP, SCOMCODE, SLPAREN, SFLDNAME,
        SCOMPOP1, SFLDVAL, SRPAREN
;
ADD SSEGNAME;
ITEM
HELD-AS CHAR 8
CONTENTS IS "TEST-SEG"
;
ADD SCCSEP;
ITEM
HELD-AS CHAR 1
CONTENTS IS "*"
;
ADD SCONCODE;
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 (')');
```

Assembler 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 DSDCL27  
SSEGNAME DCCL8 'TEST-SEG'  
SCCSEP DCCL1 '*'  
SCOMCODE DCCL4 '----'  
SLPAREN DCCL14 '('  
SFLDNAME DCCL8 'TESTFLD'  
SCOMPOP DCCL2 'EQ'  
SFLDVAL DCCL2 'AA'  
SRPAREN DCCL1 ')' '  
*                               END OF GROUP TESTSSA
```



---

## Appendix A

---

# Macros for Tailoring the IMS Interface

## Implementation of the IMS (DL/I) Interface Macros

Several macros (in addition to those described in the *ASG-Manager Products Source Language Generation* publication) are available to enable IMS (DL/I) Interface output generated by the PRODUCE command to be tailored to conform to a particular installation's standards. These macros are:

- DGDBD, to enable the output of DBD control statements to be tailored
- DGPSB, to enable the output of 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 output to be tailored
- DGSREC, to enable the output of record layouts to be tailored

These macros are supplied as source modules on the installation magnetic tape. The tables in ["The Macros DGDBD And DGPSB" on page 210](#) and ["The Macros DGSCOB, DGSPLI, DGSBAL, and DGSREC" on page 213](#) list the keywords of the macros, for which values can be specified when Manager Products are 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 OS Environments* publication must be carried out.

These are the names of the resulting assembled modules:

<b>Macro</b>	<b>Module</b>
DGDBD	DIL88
DGPSB	DIL89
DGSBAL	DIL97

<b>Macro</b>	<b>Module</b>
DGSCOB	DIL99
DGSPLI	DIL98
DGSREC	DIL96

## The Macros DGDBD And DGPSB

The macros DGDBD and DGPSB, respectively, enable the generation of DBD control statements and PSB control statements to be tailored. This table lists the keywords of these macros for which values can be specified when Manager Products is installed.

<b>Keyword</b>	<b>Specifies</b>
ACHAR	The hex values of any extra characters that are to be accepted for output in names produced by the Source Language Generation Facility, to enable characters not in the standard source language character set to be output (see <a href="#">remark 1 on page 212</a> ).  Default value: None.  Alternative value: Any valid hexadecimal value, or a sublist of such values.
ACSMETH	Type of file to be generated.  Default Value: BPAM.  Alternative Value: QSAM.
ALIAS	Whether IMS specific aliases are to be generated instead of member names.  Default Value: No.  Alternative Value: Yes (see <a href="#">remark 2 on page 212</a> ).
COLMAIN	Starting character position for statement type.
COLMENT (DGDBD only)	Starting character position for label generated from ADD-TO clause.  Default Value: I.  Alternative Value: Up to 99.
COLSUBS	Starting character position for keyword or operand.  Default Value: 16.  Alternative Value: Up to 99.

Keyword	Specifies
CONCARD	Whether a control card is to be produced. Default Value: Yes. Alternative Value: No (see <a href="#">remark 3 on page 212</a> ).
DDNAME	Default library name. Default Value: 'GENLIB'. Alternative Value: 'name' (see <a href="#">remark 4 on page 212</a> ).
IMSLVL	The level of IMS for which you require Manager Products to provide support (see <a href="#">remark 6 on page 213</a> ). Default value: V3. Alternative value: 1.2, 1.3, V3, V4, VS.
KNOWNAS	Whether local-names from KNOWN-AS clauses are to be generated instead of member names. Default Value: No. Alternative Value: Yes (see <a href="#">remark 2 on page 212</a> ).
LIBCC	The format of the control card output as the first record of a QSAM FILE (unless overridden by an ONTO clause). Default Value: see <i>ASG-Manager Products Source Language Generation</i> . Alternative Value: Delimited character string of 1 to 72 characters including a question mark (?).
MEMLEN	Maximum length of library-name. Default Value: 8. Alternative Value: Up to 16.
RXLOG01 (DGDBD only)	Whether to relax the Manager Products integrity rule which forces a reference to destination parent segments (see <a href="#">remark 7 on page 213</a> ). Default Value: No. Alternative Value: Yes.
RXLOG02 (DGDBD only)	Whether to relax the Manager Products integrity rule that the concatenated segment variation with dependents must be the left-most (see <a href="#">remark 8 on page 213</a> ). Default Value: No. Alternative Value: Yes.

Keyword	Specifies
RXLOG01 (DGPSB only)	Whether to relax the Manager Products integrity rules for indirectly contained segments within logical segments (see <a href="#">remark 9 on page 213</a> ). Default Value: No. Alternative Value: Yes.
RXLOG02 (DGPSB only)	Whether to relax the Manager Products integrity rules so that a PCB may refer to multiple variations of a concatenated segment (see <a href="#">remark 10 on page 213</a> ). Default Value: No. Alternative Value: Yes.
RNDBIN	Rounding of binary items. Default Value: Yes. Alternative Value: No.
RNDBIT	Whether bit string fields are to be generated with byte alignment (see <a href="#">remark 5 on page 212</a> ). Default Value: No. Alternative Value: Yes.

### Remarks

- 1 The standard Source Language Generation Facility output character set for DBD and PSB control statements conforms to that defined for COBOL for the data division. This character set can be extended to allow nonstandard 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 Manager Products 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 The variable *name* is a valid name, delimited, and up to 32 characters long. It must be different from all other values named or used by default for the same macro.
- 5 The effect of the RNDBIT parameter is overridden by any alignment specification stated in the data definition of any group or segment that contains the bit string item.

- 6 You can use the IMSLVL parameter to decide when to take advantage of the additional features available in later IMS releases. The value of IMSLVL is the default for all PRODUCE IMS commands. You can override this value using the RELEASE/VERSION keywords in a PRODUCE IMS command.
- 7 You may want to suppress even key-sensitivity to a destination parent, to reduce I/O overhead. When RXLOGO is specified as YES for DGDBD, the first source segment may now be specified as the only segment contained within the logical segment. As a result, warning message DM02517 with severity level W is never issued when producing the source for an IMS database definition.
- 8 If there are multiple variations of a concatenated segment specified within a logical database, it is possible to allow only one of these variations to have dependents. Normally, this is only allowed to be the left-most, or first, variation in the hierarchy. If RXLOGO2 is specified as YES for DGDBD, this rule is changed so that the variation with dependents need not be the left-most, although there may still only be one of the variations that has dependents.
- 9 If RXLOG01 is specified for DGPSB, it is accepted as valid to generate a PSB containing a PCB that makes reference to segments indirectly contained within a logical database. By indirectly, we mean that either of the sources of a concatenated segment contained by a logical database may be referenced in place of the concatenated segment as the parent of any dependent segments.
- 10 If RXLOG02 is specified for DGPSB, a PSB may be generated containing a PSB that refers to more than one variation of a logically concatenated segment. Note that this is only valid in the context of a HD database where the segments in question are utilizing direct-address pointers and twin pointers are specified.

## **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 Sensitive Fields Input/Output Areas" on page 198](#). This table lists the keywords of these macros for which values can be specified when Manager Products is installed.

Keyword	Specifies
CONKEY	<p>In a logical child segment: the name to be applied to the destination parent's concatenated key. In an index pointer segment that is pointed to by symbolic pointers: the name to be applied to the concatenated key of the corresponding index target segment. This concatenated key is included in the data portion of the index pointer segment if the concatenated key does not appear in the subsequence or duplicate-data fields. The value specified by CON KEY is only used when no CONCATENATED-KEY-NAME clause has been specified in the SEGMENT member.</p> <p>Default Value: CONCAT-KEY.</p> <p>Alternative Value: name.</p>
CONSTNT	<p>The name to be applied to the CONSTANT field of an index pointer segment.</p> <p>Default Value: CONSTANT.</p> <p>Alternative Value: name.</p>
DUPDATA	<p>The name to be applied to the duplicate data fields of an index pointer segment.</p> <p>Default Value: DUP-DATA-FLD.</p> <p>Alternative Value: name.</p>
SIZE	<p>The name to be applied to the SIZE field of a variable length segment.</p> <p>Default Value: SIZE-FIELD.</p> <p>Alternative Value: name.</p>
SUBSEQ	<p>The name to be applied to the subsequence fields of an index pointer segment.</p> <p>Default Value: SUBSEQUENCE-FLD.</p> <p>Alternative Value: name.</p>
USERDAT	<p>The name to be applied to the user-data field of logical child and index pointer segments.</p> <p>Default Value: USER-DATA.</p> <p>Alternative Value: name.</p>

**Note:** \_\_\_\_\_

The variable *name* is a valid name, delimited, and up to 32 characters long. It must be different from all other values named or used by default for the same macro.

---

---

## Appendix B

---

# Manager Products and IMS Keywords

## Introduction

Most keywords used in Manager Products IMS member types are similar to the equivalent IMS usage. Some meanings may not be so clear, and so are explained further in this appendix. Manager Products keywords are given in CAPITALS, and IMS terms are given in *italics*.

## IMS Databases

CONTAINS: This clause represents the *segment hierarchy* for any database.

HDAM/HIDAM databases

The ADD-TO and PRIME clauses indicate one or more PHYSICAL SEGMENTS that must be added to the database within a *secondary set group*.

LOGICAL databases

The CONTAINS clause indicates that either a *logical segment* or a *physical segment* may be included in the *logical database*.

SECONDARY-INDEX databases

The SHARES-WITH clause gives the capability of representing a *shared secondary index database* that may contain several indices.

## Physical Segments

RELATED-AS (logical relationships)

UNIDIRECTIONAL-CHILD-SEGMENT is the *logical child* in a *unidirectional relationship* and represents a *pointer segment*.

The CONTAINS CLAUSE for a UNIDIRECTIONAL-CHILD-SEGMENT represents the *intersection* data in the *logical relationship*.

REAL-PAIRED-CHILD-SEGMENTs represent the real (as opposed to *virtual*) half of a *bidirectional-virtual paired relationship*, also known as the *RLC (Real Logical Child)*.

The POINTERS clause for a REAL-PAIRED-CHILD-SEGMENT allows you to specify these types of pointers:

- FORWARD-LOGICAL-TWIN = *LTF (logical twin forward)*
- BACKWARD-LOGICAL-TWIN = *LTB (logical twin backward)*
- SINGLE-LOGICAL-CHILD = *LCF (logical child first)*
- DOUBLE-LOGICAL-CHILD = *LCL (logical child last)*

ATTRIBUTES clause (*physical characteristics*):

The POINTERS clause for PHYSICAL SEGMENT allows you to specify the following clauses (representing IMS pointer types):

- FORWARD-HIERARCHICAL = *HF (hierarchical-forward)*
- BACKWARD-HIERARCHICAL = *HB (hierarchical-backward)*
- FIRST-CHILD = *PCF (physical child first)*
- LAST-CHILD = *PCL (physical child last)*
- SINGLE-TWIN = *PTF (physical twin forward)*
- DOUBLE-TWIN = *PTB (physical twin backward + physical twin forward)*
- NOTWIN = *no twin pointers*
- COUNTER = *CTR (counter only)*
- unspecified = *NONE (unidirectional and real-paired children only)*.

## A

ADD-TO interrogation keyword 146  
alignment 37  
application view 13  
arrays  
    FIELD control statements for 180

## B

BACKWARD-LOGICAL-TWIN pointer 33  
BOUND interrogation keyword 146, 150  
BSAM access method 72  
BULK command 2, 139

## C

calls, command code 134  
COBOL SYNCHRONIZED keyword 36  
command codes 134  
common clauses 23  
CONCATENATED 34  
concatenated 34, 136  
concatenated key 34  
    as sensitive field in PCB 128  
    construction of 34  
    destination parent 15  
    index source segment 35  
    index target segment 35  
    internal member type 34  
    name 34  
    segment search arguments 136  
    sensitive segments in PCB 186  
concatenated segment 136  
    logical 108  
CONCATENATED-KEY-  
    CONSTITUENTS interrogation  
    keyword 150  
CONCATENATED-KEY-FIELDS  
    clause 35  
CONCATENATED-KEY-NAME 3, 34  
CONCATENATED-KEY-NAME clause 34  
    index pointer segment 65  
concatenated-key-name fields 136

CONCATENATED-KEY-NAME  
    interrogation keyword 148  
CONCATENATED-KEY-NAMES  
    interrogation keyword 150  
CONTAINED-BY interrogation  
    keyword 174  
CONTAINS interrogation keyword 146  
CONTAINS list 15  
CONTENT declaration 28  
control interval size 73  
control module in application 20  
controller's commands 141  
conventions page vi  
counter field in logical parent segment 34  
crossing logical relationships 108

## D

data  
    intersection 15  
data description statement generation  
    for PCB masks 14  
    for segment I/O areas 127  
    segment search arguments (SSA) 132  
data fields 16  
data set  
    groups 73  
data set overflow 82  
database  
    loading (processing option) 122  
    primary index 23  
    reading processing option 122  
    updating processing option 122  
database definition 16  
    BLOCK subordinate clause 83  
    BUFFER clause 83  
    INPUTS clause 74  
    MODEL clause 83  
    OUTPUT clause 77  
Database Description (DBD) Control  
Statements 23  
    for HIDAM database 102  
    for LOGICAL database 107

- for primary index database 98
- DATABASES interrogation keyword 150
- DATASET control statements 180
- DATASETS clause 76
- DBD control statements
  - for shared SECONDARY-INDEX database 181
  - KEY operand 108
- DBD FIELD Control Statements 62
- ddname 18
- DEFAULTED-AS form of ITEM 16
- destination parent segment 32
- destination parent's concatenated key 31
- DEVICE clause 77
- DGDBD macro 209
- DGPSB macro 209
- DGSBAL macro 209
- DGSCOB macro 209
- DGSPLI macro 209
- DGSREC macro 209
- DIRECT-ADDRESS pointer 32
- DL/I- DATASETS
  - interrogation keyword 146
- DL/I-DATABASE member type 2
- DL/I-DATASET internal member type 74
- DL/I-DATASETS 3
- dummy members 18
- duplicate data fields across segments 38
- DUPLICATE-DATA list 20
- DUPLICATE-DATA-FIELDS
  - interrogation keyword 148
- DUPLICATE-DATA-FIELDS clause 61

**E**

- edit/compression routine for segment 43
- EDIT-COMPRESSION-EXITS
  - interrogation keyword 150
- encoding 15
- ENTERED-AS form of ITEM 16
- exit, user 43

**F**

- FATHERED-BY interrogation keyword 174
- FATHERS interrogation keyword 147
- FIELD statements 20
- fixed length logical record 73
- FIXED record format 73
- floating point items 60
- form description of ITEM members 16
- FORWARD-HIERARCHICAL keyword 41
- FORWARD-LOGICAL-TWIN pointer 33
- FREQUENCY clause 39
- FREQUENCY-FREE-BLOCKS clause 90

**G**

- GENERATED-BY interrogation keyword 174
- GENERATES
  - interrogation keyword 148, 151
- GENERATES clause 20
- GENERATES-FIELDS keyword in PRODUCE command 178
- GENLIB output file 182
- GLOSSARY command 2, 139
- GROUP member type 16
- GSAM database 121
  - ACCESS clause 72
  - arrays in 146
  - CONTAINS clause 74
  - input data set 76
  - interrogation of 164
  - multibuffering option 121
  - PCB for 121

**H**

- HDAM database 89
  - ACCESS clause 89
  - ADD-TO clause 90
  - ANCHOR-POINTS clause 89
  - CONTAINS clause 89
  - DATASET control statement 91
  - definition 89
  - dependent segment 42
  - INSERTION-BYTES-MAXIMUM clause 89
  - interrogation of 146
  - pointers 34
  - RANDOMIZING-MODULE clause 89
  - RELATIVE-BLOCK-MAXIMUM clause 89
  - root segment of 40
- HELD-AS form of ITEM 16, 61
- HIDAM database
  - DATASET control statements 180
  - interrogation of 146
  - primary index database 17
- hierarchical
  - relationship interrogation 143
- HISAM database
  - interrogation of 147
- HSAM database
  - interrogation of 147

**I**

- IF
  - interrogation keyword 146

- IMS (DL/I) Control Statements 3
  - IMS member type keywords 139
  - IMS-DATABASE member type 2
  - IMS-DATASETS keyword 3
  - IN-DATABASES interrogation
    - keyword 165
  - index
    - search fields 17
    - source segment 18
    - target segment 19
  - index pointer 20
  - indexing
    - secondary 12
  - INDEX-SEARCH-FIELDS 3
  - installation macros 189
  - internal DataManager member types 17
  - interrogations 146
  - intersection data 15
  - ITEM members 16
- L**
- LCHILD control statements 180
  - LIST command 2, 139
  - logical database 9
    - interrogation of 145
- M**
- MAINTENANCE-EXITS interrogation
    - keyword 149
  - MARK IV file definition forms 176
  - member types
    - internal 17
    - selection keywords 140
  - MODEL clause 77
  - module
    - control 20
  - MODULE member type 20
- N**
- NAME interrogation keyword 152
  - NO-ASSEMBLY-PRINT keyword 184
- O**
- OF interrogation keyword 151
  - ON interrogation keyword 149
  - output source library data set 182
- P**
- PARENTS interrogation keyword 147
  - path calls 194
  - PCB 14
  - PERFORM command 2, 139
  - physical database 5
    - interrogation of 150
  - physical segment 5
  - primary index database 180
  - PROCESSES clause
    - CONTAINS clause 19
    - SEGMENT-SEARCH-ARGUMENT
      - S clause 20
  - PRODUCE command
    - ALL-FIELDS keyword 178
    - AS clause 181
    - CONDITION-CODE clause 185
    - control options 176
    - DIRECT-FIELDS keyword 178
    - ENQUEUES clause 185
    - FROM clause 178, 181
    - IO-PCB keyword 185
    - IO-SIZE clause 185
    - OPTIONS clause 184
    - PRIMARY-INDEX clause 180
    - SEARCH-FIELDS keyword 178
    - SSA-SIZE clause 185
    - USE clause 182
    - USED-IN clause 187
    - USING clause 182
  - Program Communication 14
  - Program Communication Block
    - GSAM database 19
    - interrogation of 160
    - SEGMENT clause 186
    - WTOR keyword 185
  - Program Communication Blocks 13
  - PROGRAM member type 20
  - Program Specification Block (PSB) Control
    - Statements 14
  - PSB control statements
    - generation 183
    - language operand 185
    - library-names 182
    - PROCSEQ operand 186
  - PSBGEN control statement operands 185
- Q**
- qualified segment search arguments 20
  - QUALIFIED-ON interrogation
    - keyword 153
- R**
- RANDOMIZING MODULES interrogation
    - keyword 147
  - REAL-PAIRED-CHILD-SEGMENT 33
  - RENAMES interrogation keyword 151
  - REPORTED-AS form of ITEM 16

**S**

search fields  
    in segment search arguments 20  
SEARCH list 20  
SEARCH-KEY-FIELDS interrogation  
    keyword 149  
secondary index relationship 10  
secondary indexing, fields for 179  
SECONDARY- SEQUENCE-ON  
    interrogation keyword 152  
SEGM control statements 180  
SEGMENT  
    interrogation keyword 152  
SEGMENT member type 7  
segment search argument 11  
segment search field 20  
SEGMENT-SEARCH-ARGUMENTS  
    clause 20  
SENSEG control statements 186  
SENSITIVE-FIELDS interrogation  
    keyword 153  
sequence key fields 17  
SEQUENCE-KEY interrogation  
    keyword 149  
SEQUENCE-KEY-CONSTITUENTS  
    interrogation keyword 151  
SEQUENCE-KEYS interrogation  
    keyword 3  
shared SECONDARY-INDEX database 181  
SHARES-WITH interrogation keyword 148  
SOURCE  
    interrogation keyword 149  
Source Language Generation facility 3  
SSAS  
    interrogation keyword 153  
SUBSEQUENCE list 20  
SUBSEQUENCE-FIELDS interrogation  
    keyword 149  
SYSTEM member type 20  
system related fields 17  
SYSTEM-RELATED-FIELDS internal  
    member type 3

**T**

TARGET interrogation keyword 149  
TO interrogation keyword 152

**U**

UNDEFINED record format 73  
user data 179

**V**

validation performed by DataManager 17

VARIABLE record format 73  
VIA clause  
    interrogation 143  
VSAM access method 72

**W**

WHAT command 141  
WHICH command 2, 139, 141



ASG Worldwide Headquarters Naples Florida USA | [asg.com](http://asg.com)