

ASG-Manager Products™ Source Language Generation

Version 2.5

Publication Number: MPR0500-25-SLG

Publication Date: September 1999

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.

ASG Documentation/Product Enhancement Fax Form

Please FAX comments regarding ASG products and/or documentation to (239) 263-3692.

Company Name	Telephone Number	Site ID	Contact name

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

Enhancement Request:

ASG Support Numbers

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

Please have this information ready:

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

If You Receive a Voice Mail Message:

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

Severity Codes and Expected Support Response Times

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

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

Business Hours Support

Your Location	Phone	Fax	E-mail
United States and Canada	800.354.3578	239.263.2883	support@asg.com
Australia	61.2.9460.0411	61.2.9460.0280	support.au@asg.com
England	44.1727.736305	44.1727.812018	support.uk@asg.com
France	33.141.028590	33.141.028589	support.fr@asg.com
Germany	49.89.45716.222	49.89.45716.400	support.de@asg.com
Singapore	65.6332.2922	65.6337.7228	support.sg@asg.com
All other countries:	1.239.435.2200		support@asg.com

Non-Business Hours - Emergency Support

Your Location	Phone	Your Location	Phone
United States and Canada	800.354.3578		
Asia	65.6332.2922	Japan/Telecom	0041.800.9932.5536
Australia	0011.800.9932.5536	Netherlands	00.800.3354.3578
Denmark	00.800.9932.5536	New Zealand	00.800.9932.5536
France	00.800.3354.3578	Singapore	001.800.3354.3578
Germany	00.800.3354.3578	South Korea	001.800.9932.5536
Hong Kong	001.800.9932.5536	Sweden/Telia	009.800.9932.5536
Ireland	00.800.9932.5536	Switzerland	00.800.9932.5536
Israel/Bezeq	014.800.9932.5536	Thailand	001.800.9932.5536
Japan/IDC	0061.800.9932.5536	United Kingdom	00.800.9932.5536
		All other countries	1.239.435.2200

ASG Web Site

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

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

Contents

Preface	iii
About this Publication	iii
Publication Conventions	iv
1 Source Language Generation	1
What IS Source Language Generation?	1
Why Generate from the Repository?	2
How Are Source Languages Generated?	2
Record Layouts	3
Generation of Data Structures	3
Fillers	4
Comments	5
Tailoring Capabilities	5
2 Generation of COBOL Data Descriptions	7
How to Generate COBOL Data Descriptions	7
Introduction to COBOL Source Generation	8
COBOL Generation from FILES	9
COBOL Generation from GROUPs and Arrays	11
COBOL Generation from ITEMs	14
COBOL Generated from ITEM's Form-description	14
COBOL Generated from ITEM's CONTENTS Clause	16
COBOL Generated from ITEM's NOTE and DESCRIPTION Clauses	17
Generation of COBOL Fillers and Dummy Names	18
Level Numbers	19
3 Generation of PL/I Data Descriptions	21
How to Generate PL/I Data Descriptions	21
Introduction to PL/I Source Generation	22
Storage Attribute Declarations in PL/I	22
PL/I Structures and Level Numbers	22
Based Structures	23
PL/I Generation from Arrays	24
Generating PL/I Elementary Items	25
Generating PL/I INITIAL Attributes	28
Generation of PL/I Fillers and Dummy Names	29
Pointer Variables	29

4	Generation of Assembler Data Descriptions	31
	How to Generate Assembler Data Descriptions	31
	Introduction to Assembler Source Generation	31
	Assembler Generation from GROUPs	32
	Assembler Generation from Arrays	33
	Assembler Generation from ITEMs	33
	Assembler Edit Patterns	35
	Generation of Assembler EQU Statements	37
	Generation of Assembler DC Statements	38
	Generation of Assembly Fillers and Dummy Names	40
5	Generation of Record Layouts	41
	How to Generate Record Layouts	41
	Record Layouts: Overview and Example	42
	Fields in Record Layouts	43
	Format of the Generated Layout	45
6	Tailoring Source Language Generation	47
	Installation Macros	47
	Source Library Dataset Control	48
	Record Layouts Tailoring	48
	Source Language Output Format Tailoring	49
	Import from COBOL Function Filler Name Conversion	50
	Output Source Language Tailoring	51
7	Command Specifications	53
	PRODUCE Command	53
	Generic Overview of the PRODUCE Command	53
	COBOL Generation	54
	PL/I Generation	55
	Assembler Generation	57
	Record Layouts Generation	58
	Output Control Options Overview	59
	Specifying the Output Dataset	59
	Suppressing Output to a Source Library Dataset	61
	Controlling Output During Source Language Generation	61
	Generation Control Options Overview	61
	Deriving Data Names from Aliases	62
	Specifying the Format and Contents of Output	63
	Suppressing Specified Generation Options	65
	Selecting a Form or Version of an ITEM Member	67
	Name Editing Options Overview	68
	Replacing Names or Name Elements	69
	Dropping Names or Name Elements	69
	Inserting Characters Into Names	70
	Conditional Editing	70
	PRODUCE Syntax	72
	SHOW PRODUCE-OPTIONS	76
	Syntax	77

Preface

ASG-Manager Products Source Language Generation describes the ASG-Manager Products (herein called Manager Products) export function that provides for the production of programming source language data descriptions and/or record layouts from the data definitions held in the repository. It is concerned mainly with the generation of COBOL, PL/I, and Assembler data descriptions and record layouts for conventional file environments. It also tells you where source language generation for other environments is documented, and provides a basis on which that documentation builds.

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, "Source Language Generation,"](#) introduces source language generation, lists the benefits it can provide, and gives some information that is common to the generation of COBOL, PL/I, and Assembler data descriptions.
- [Chapter 2, "Generation of COBOL Data Descriptions,"](#) describes in detail how COBOL source language data descriptions are generated.
- [Chapter 3, "Generation of PL/I Data Descriptions,"](#) describes in detail how PL/I source language data descriptions are generated.
- [Chapter 4, "Generation of Assembler Data Descriptions,"](#) describes in detail how Assembler source language data descriptions are generated.
- [Chapter 5, "Generation of Record Layouts,"](#) describes in detail how record layouts are generated.
- [Chapter 6, "Tailoring Source Language Generation,"](#) summarizes the installation macros that can be used to tailor the generation of COBOL, PL/I, and Assembler data descriptions and record layouts to suit the requirements of your installation.
- [Chapter 7, "Command Specifications,"](#) gives the full specifications of the commands that can be used to generate COBOL, PL/I, and Assembler data descriptions and record layouts.

Publication Conventions

The following conventions apply to syntax diagrams that appear in this publication.

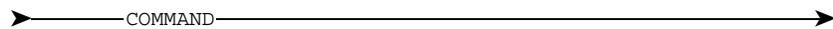
Diagrams are read from left to right along a continuous line (the "main path"). Keywords and variables appear on, above, or below the main path.

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:



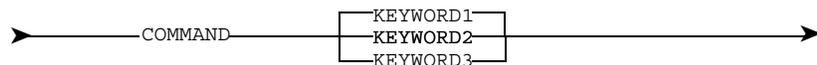
A required keyword appears on the main path:



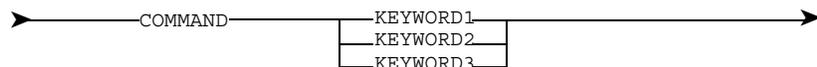
An optional keyword appears below the main path:



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:

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

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

1

Source Language Generation

This chapter explains Manager Products source language generation and contains these sections:

Section	Page
What IS Source Language Generation?	1
Why Generate from the Repository?	2
How Are Source Languages Generated?	2
Generation of Data Structures	3
Fillers	4
Comments	5
Tailoring Capabilities	5

What IS Source Language Generation?

Source language generation, a Manager Products export function, is the production of programming source language data descriptions and/or record layouts from the data definitions held in the repository. Database management system source language statements and MARK IV file management system source language statements can also be generated.

The generation of COBOL, PL/I, and Assembler data descriptions and record layouts for conventional file environments is described here.

Output from source language generation consists of an output source dataset (except in CICS environments) and/or record layouts, a source listing and messages relating to the execution of the command.

Why Generate from the Repository?

Source language generation is a powerful tool for standardizing data descriptions within an organization as well as eliminating the tedious task of coding source language data descriptions in application programs. These are some of the advantages for the Data Administrator:

- Retains control of data definition within the organization
- Extends full Manager Products security procedures right down to coding levels
- Extends naming standards right down to coded data names
- Enables changes to be accomplished swiftly, in a controlled fashion, and with the minimum duplication of effort

These are some of the advantages for the Systems Analyst/Programmer:

- Eliminates most of the work in interpreting the data structure and in manually coding the source language data descriptions
- Obtains dynamically computed record layouts of a data structure (independent of any source language conventions, if desired)
- Obtains automatically generated and therefore correctly defined data descriptions no matter how large or complex the data structure is

How Are Source Languages Generated?

Source language data descriptions and/or record layouts are generated using the export COBOL, export PL/I or export Assembler functions, provided by panels selected from the ASG-MethodManager (herein called MethodManager) menu E00000, or the PRODUCE command. You can specify these details:

- Whether source language data descriptions or record layouts or both are to be generated
- The programming language in which the descriptions are to be generated
- The names of the repository members from which generation is to take place

If you specify a repository member at a FILE, GROUP, or segment level, the reference paths from, and including, the repository members specified are followed, and data descriptions are generated for members on that path. There are some exceptions to this process; see ["Generation of Data Structures" on page 3](#).

Fillers are automatically generated where necessary. See ["Fillers" on page 4](#).

Comments can optionally be output from NOTE and DESCRIPTION clauses. See ["Comments" on page 5](#).

There is provision for you to specify these elements:

- The type of source library dataset to be output, and its name
- The format of the generated output
- The forms and versions of the repository members to be used for generation
- The use of aliases or local names, instead of member names, as the basis for the generated data names
- Editing of the output data names, to be performed as they are being generated

All generated source language data names are checked to conform to source language rules relating to maximum length and illegal characters. Names are modified if necessary by removing any illegal characters and shortening long names by removing middle characters.

Record Layouts

Record layouts are tables containing these details:

- The levels and names of groups and items within the record
- The decimal and hexadecimal offsets of the storage fields for the groups and items in the record
- The length, type, and alignment of the groups and items
- Remarks giving additional information about the member

Use installation macros to tailor layouts for your installation requirements.

Generation of Data Structures

Data descriptions are generated for all encoded members directly and indirectly referred to by each member named in the generation panel or the PRODUCE command. The complete set of data descriptions is known as a *data structure*.

For example, if generation of the data description is to be from the repository member EMP-IDENT and the member refers to EMP-NAME, EMP-ADDRESS, and TELEPHONE, a data structure will be created with EMP-IDENT at the top level and EMP-NAME, EMP-ADDR, and TELEPHONE at the next level down.

Generation takes place in respect of each of the members named, in the order in which they are named. If any of these members is:

- Not encoded
- Of a member type that is not valid in the context (for example, if a COBOL data description is to be generated, and the named member is not a FILE, GROUP, or ITEM, or a member of a user-defined member type based on a FILE, GROUP, or ITEM)
- Protected against access by the current user

then no generation takes place from that member, a message is issued, and processing continues with the next member name.

Reference paths are followed from top to bottom and from left to right, until generation of the structure is complete, subject to these exception conditions:

- If a recursive reference (that is, a reference to a member that has already been encountered earlier in the reference path) is encountered, then no generation takes place in respect of the recursive reference, and that reference path is terminated at that point.
- If a reference to a member of a type that is not valid in the context is encountered, then the path is terminated at that point.
- If a member is encountered with a dummy data entries record, the path is terminated on that member and a one-character item is generated.
- If Manager Products' security capabilities are implemented and a member is encountered whose access security level is higher than the user's security level (specific or general, depending on whether the member is owned or not), then the path from that point, though followed, is concealed (even if it contains unprotected members). A single filler item is generated whose length is equal to the length of the protected member, if that member is an ITEM, else is equal to the sum of the lengths of the items directly or indirectly contained by the protected member. In such cases, the generated source language and/or record layout is complete only so far as the total storage space is concerned.

Fillers

Filler names and data descriptions are automatically generated from referenced members where appropriate. Filler names are generated at these times:

- When a protected member is encountered in a reference path. In this case, a data description is generated whose length is:
 - The length of the protected member, if the protected member is an ITEM
 - The sum of the lengths of the items directly or indirectly contained by the protected member, if the protected member is a GROUP
- When a blank data name would otherwise be generated by editing operations or by the automatic removal of characters that are illegal in the source language being generated.
- When a filler data description is generated to equalize a redefining and redefined storage area, or to pad storage when a repository length definition exceeds that permitted in the source language being generated.
- Where a member name is a name that was generated from a source language filler name by the import from COBOL functions. See "[Import from COBOL Function Filler Name Conversion](#)" on page 50.

Filler names and data descriptions are language-dependent.

Comments

If tailored in an installation macro, or if specified in the generation panel or in the PRODUCE command, comments are generated from NOTE and/or DESCRIPTION clauses. You can specify whether the generation is to be from all or from the first *n* delimited strings of the specified clause(s). See ["Specifying the Format and Contents of Output" on page 63](#).

If a NOTE or DESCRIPTION character string is too long to be accommodated in one comment line, then it is continued over as many comment lines as necessary.

Comments are output immediately following the source code associated with the member containing the NOTE and/or DESCRIPTION clause.

Comments are not subject to editing.

Tailoring Capabilities

Many aspects of source language generation can be controlled at run-time by specifying requirements in the generation panel or the PRODUCE command. However, provision is made for the output to be tailored in advance to your installation standards by the Administrator. This is done by specifying values for keywords in a series of *installation macros* when your Manager Products configuration is installed.

There are separate installation macros to tailor COBOL, PL/I, and Assembler generation, together with one to tailor the generation of record layouts when no programming language is specified. Similar macros are provided for database interfaces.

The macros can be used to:

- Specify the source library dataset for output
- Specify which parts of a definition are to be used when generating output
- Tailor the data names to be generated from the definitions in the repository
- Change the format of record layouts
- Tailor the layout of the generated data descriptions
- Tailor filler names
- Tailor the maximum size of data structures

The settings of the installation macros provide the default or standard requirements for tailoring source language generation; in some cases they can be overridden by the requirements specified at run time. A command, SHOW PRODUCE-OPTIONS, is provided by which you can interrogate the settings that are in force at any time. See [Chapter 7, "Command Specifications," on page 53](#).

2

Generation of COBOL Data Descriptions

This chapter explains how to generate COBOL data descriptions and contains these sections:

Section	Page
How to Generate COBOL Data Descriptions	7
Introduction to COBOL Source Generation	8
COBOL Generation from FILEs	9
COBOL Generation from GROUPs and Arrays	11
COBOL Generation from ITEMs	14
Generation of COBOL Fillers and Dummy Names	18
Level Numbers	19

How to Generate COBOL Data Descriptions

To generate COBOL data descriptions, use the export COBOL function, provided by the MethodManager panel TE11000 or the PRODUCE COBOL command.

How to generate COBOL using panel TE11000 is described in that panel's Help.

For the specifications of the PRODUCE command, see [Chapter 7, "Command Specifications," on page 53](#).

Introduction to COBOL Source Generation

These are some of the COBOL constructs supported by source language generation:

```
FD filename
BLOCK CONTAINS
RECORDING MODE IS
LABEL RECORDS ARE
LABEL RECORD IS
DATA RECORD IS
DATA RECORDS ARE
01-49 data-name
ASCENDING KEY IS
DESCENDING KEY IS
FILLER
REDEFINES
OCCURS n TIMES
OCCURS n1 TO n2 TIMES DEPENDING ON
INDEXED BY
PICTURE
SIGN IS LEADING/TRAILING
SIGN IS LEADING/TRAILING SEPARATE CHARACTER
JUSTIFIED
JUSTIFIED RIGHT
SYNCHRONIZED
COMPUTATIONAL - BINARY (for COBOL II)
COMPUTATIONAL-1 (for COBOL II)
COMPUTATIONAL-2 (for COBOL II)
COMPUTATIONAL-3 - PACKED DECIMAL (for COBOL II)
88 condition-name VALUE literal THROUGH literal ...
VALUE literal
VALUE ZERO/ZEROS
VALUE SPACE/SPACES
VALUE LOW-VALUES/HIGH-VALUES
VALUE QUOTES
VALUE ALL literal
EXTERNAL (for COBOL II)
GLOBAL (for COBOL II)
USAGE IS INDEX (for COBOL II)
USAGE IS POINTER (for COBOL II)
USAGE IS DISPLAY-1 (for COBOL II)
```

Source language generation of COBOL from the repository is checked to conform to American National Standard COBOL as relating to the data division. However, the character set may be extended, by tailoring the installation macro, to allow specified non-standard characters to be output in data names. Warning messages are given where COBOL data descriptions are generated that do not strictly adhere to the standard (although acceptable to certain compilers), and error messages are given where illegal COBOL data descriptions would be generated.

Generated data names are not checked against COBOL reserved names.

Responsibility for ensuring that any additional characters specified for inclusion in the output character set are acceptable to the software with which the output will be used rests with the user.

The elements of repository member definitions that generate particular elements of COBOL code are set out in the following sections.

You can use them to work out what COBOL code is generated from an existing member definition and to create member definitions to match the contents of the DATA DIVISIONs of existing COBOL programs.

In the tables, the names referred to as *data-name-1*, *data-name-2*... in the COBOL column correspond to *member-name-1*, *member-name-2*... in the repository definition column after any editing; except that aliases or local names may be used instead of member names if so tailored or if specified in the generation panel or the PRODUCE command.

COBOL Generation from FILES

File definitions (FDs) and 01-level record descriptions are generated from a FILE member on the assumption that the definition of the FILE's contained members is in accordance with COBOL hierarchy conventions. You can, however, specify in the generation panel or in the PRODUCE command, or by tailoring the installation macro, that only FDs or only record descriptions are to be generated.

Definition in Repository	COBOL Source
FILE's <i>member-name</i>	FD <i>file-name</i>
FIXED <i>n</i> , VARIABLE <i>n</i> , UNDEFINED <i>n</i> or SPANNED <i>n</i> (see note 1)	BLOCK CONTAINS <i>n</i> CHARACTERS RECORDING MODE IS ...
FIXED, VARIABLE, UNDEFINED or SPANNED	RECORDING MODE IS ...
NO-LABELS	LABEL RECORDS ARE OMITTED
STANDARD-LABELS (or no label definition)	LABEL RECORDS ARE STANDARD
USER-LABELS <i>module-name</i> (unless tailored)	LABEL RECORD IS LABEL-AREA
CONTAINS <i>member-name-1</i>	DATA RECORD IS <i>data-name-1</i> . 01 <i>data-name-1</i> .
CONTAINS <i>member-name-1</i> , <i>member-name-2</i>	DATA RECORDS IS <i>file-name-REC</i> . 01 <i>file-name-REC</i> . 02 <i>data-name-1</i> . 02 <i>data-name-2</i> .

Definition in Repository	COBOL Source
CONTAINS <i>member-name-1</i> ELSE <i>member-name-2</i> (see note 2)	DATA RECORDS ARE <i>data-name-1</i> <i>data-name-2</i> . 01 <i>data-name-1</i> . 01 <i>data-name-2</i> .
CONTAINS <i>member-name-1</i> , <i>member-name-2</i> IF <i>condition</i> ELSE <i>member-name-3</i> IF <i>condition</i> ELSE <i>member-name-4</i> (<i>member-name-1</i> may be an indicator)	DATA RECORD IS <i>file-name-REC</i> . 01 <i>file-name-REC</i> . 02 <i>data-name-1</i> . 02 <i>data-name-2</i> . 02 <i>data-name-3</i> . REDEFINES <i>data-name-2</i> . 02 <i>data-name-4</i> . REDEFINES <i>data-name-2</i> .
CONTAINS (<i>member-name-1</i>) <i>member-name-2</i>	DATA RECORD IS <i>file-name-REC</i> . 01 <i>file-name-REC</i> . 02 <i>data-name-2</i> OCCURS n1 TO n 2 TIMES DEPENDING ON <i>data-name-1</i> .

Notes

1. *n* is the *maximum-block-size* (or *block-size* if FIXED). If the clause is not present in the member definition, or if it defines a zero *maximum-block-size* or zero *block-size*, then
 - For a VSAM file, the BLOCK CONTAINS clause is not generated
 - For any other file, BLOCK CONTAINS 0 CHARACTERS is generated
2. If the generation of records only (that is, no FDs) is specified, then the generated output is:

01 *data-name-1*.
01 *data-name-2* REDEFINES *data-name-1*.
3. The REC suffix in *file-name-REC* can be tailored.
4. Although the KEYS clause is available in a FILE member-type for definition of any sequence fields, this clause is ignored during COBOL generation of an FD entry.

COBOL Generation from GROUPs and Arrays

CONTAINS Clause in GROUP Member Definition in Repository	COBOL Source
<i>member-name-1</i> ELSE <i>member-name-2</i> (members have the same length)	02 <i>data-name-1</i> 02 <i>data-name-2</i> REDEFINES <i>data-name-1</i>
<i>member-name-1</i> ELSE <i>member-name-2</i> (length of <i>member-name-1</i> is greater) (Unless tailored: see note 1)	02 <i>data-name-1</i> 02 <i>data-name-2</i> FILLER REDEFINES <i>data-name-1</i> . 04 <i>data-name-2</i> 04 FILLER
<i>member-name-1</i> ELSE <i>member-name-2</i> (length of <i>member-name-2</i> is greater) (see note 1)	02 <i>data-name-1</i> FILLER. 04 <i>data-name-2</i> 04 FILLER 02 <i>data-name-2</i> REDEFINES <i>data-name-1</i> 04 FILLER
(<i>n</i>) <i>member-name-1</i> (see note 2 and note 3) (<i>item-name-1</i> version) <i>member-name-2</i> (see note 2 and note 3) (version is optional)	02 <i>data-name-1</i> OCCURS <i>n</i> TIMES. 02 <i>data-name-2</i> OCCURS <i>n1</i> TO <i>n2</i> TIMES DEPENDING ON <i>data-name-1</i>
(<i>n</i>) <i>member-name-1</i> ELSE <i>member-name-2</i> (see note 4)	02 <i>data-name-1</i> FILLER. 04 <i>data-name-1</i> OCCURS <i>n</i> TIMES. 02 <i>data-name-2</i> REDEFINES <i>data-name-1</i> FILLER
KEYS <i>member-name</i> ASCENDING (see note 7 to note 10)	ASCENDING KEY IS <i>data-name</i>
KEYS <i>member-name</i> DESCENDING (see note 7 to note 10)	DESCENDING KEY IS <i>data-name</i>
INDEXED-BY <i>index-name</i> (see note 5)	INDEXED BY <i>index-name</i>
ALIGNED (see note 6)	SYNCHRONIZED

Notes

1. The installation macro can be tailored so as not to pad unequal redefining members with fillers. Storage is equalized, irrespective of tailoring, where *member-name-1* has the shorter length.
2. In standard COBOL, the depth of nesting of OCCURS must not exceed three. COBOL II seven-dimensional arrays are permitted. If an array having an illegal number of levels is encountered in the repository, generation continues and a warning message is given. An attempt to generate COBOL from a variable array defined with an ELSE clause would generate non-standard COBOL (an OCCURS DEPENDING ON clause in the redefined or redefining item). If this is encountered, the coding is generated and a warning message is given. An attempt to generate COBOL from an array where the array element is a GROUP that contains redefining members would produce illegal COBOL. If this is encountered the coding is generated and a warning message is given.
3. *n*, *n1*, and *n2* are evaluated from RANGE or IS clauses in *item-name-1*, or from its length if there is no RANGE or IS clause. The default for *n1* is 1. If the upper bound exceeds 32767, then 32767 is defaulted. No checks are made on whether *data-name-1* is defined in the COBOL code being generated.
4. The code shown would be generated if the lengths of the array and *member-name-2* were the same, or if the length of *member-name-2* were shorter than that of the array and the installation macro had been tailored so as not to pad unequal redefining members with fillers.
5. *index-name* is subject to any name editing specified. A maximum of twelve *index-names* may be generated.
6. SYNCHRONIZED is generated for any directly contained ITEM that has BINARY or FLOATING-POINT form. Alignment is computed from the start of the record description (assumed to be on a double-word boundary) with the generation of any slack-byte fillers as required.
7. The PRODUCE output reflects the key information, as long as the definitions being produced are correctly set up to allow for ascending or descending keys.
8. The KEYS clause only appears in the PRODUCE output if defined in conjunction with the OCCURS clause.
9. If a key is not reflected as expected in the PRODUCE output, this may be because of the group level, or of the position of the particular item in the chain, and the way in which it is nested. The level at which you are producing must be one different from the level which contains the key.

10. This example shows how SLG produces concatenated keys:

```
RECORD R
  °
  GROUP GA
    °
    ITEM IA
      °
      GROUP GB
        °
        ITEM IB
```

If IB has a key and GB occurs several times in GA, then as long as GA has an index, either:

```
PRODUCE COBOL FROM GA
```

or

```
PRODUCE COBOL FROM R
```

reflects the keys. However, the command:

```
PRODUCE COBOL FROM GB
```

will not reflect the key, because it needs to be one group level removed.

If IB has a key, but GB does not occur several times in GA, then the command:

```
PRODUCE COBOL FROM GA
```

will not reflect the key, because of the absence of an OCCURS clause in GB.

COBOL Generation from ITEMS

COBOL Generated from ITEM's Form-description

ITEM Member Definition in Repository	COBOL Source
ALPHABETIC q	PIC A(q)
ALPHANUMERIC q ALPHAMERIC q CHARACTER q	PIC X(q)
LEFT-JUSTIFIED	JUSTIFIED
RIGHT-JUSTIFIED	JUSTIFIED RIGHT
BITS q (see note 1)	PIC X(($q+7$)/8)
CHARACTER q USAGE GRAPHIC	PIC G(q) DISPLAY-1 for COBOL II
CHARACTER 4 USAGE INDEX	INDEX for COBOL II
BINARY 8 USAGE POINTER	POINTER for COBOL II
HEXADECIMAL q (see note 1)	PIC X(($q+1$)/2)
BINARY $n.m$ (Notes 2, 3, 4, 6)	PICS9(n)V9(m) COMP or BINARY for COBOL II
PACKED-DECIMAL $n.m$ DECIMAL-PACKED $n.m$ (see note 2 and note 6)	PICS9(n)V9(m) COMP-3 or PACKED DECIMAL for COBOL II
NUMERIC-CHARACTER $n.m$ (see note 2 and note 6)	PIC 9(n)V9(m)
SIGNED NUMERIC-CHARACTER $n.m$ (see note 2 and note 6)	PIC S9(n)V9(m)
WITH LEADING SIGN	SIGN IS LEADING
WITH TRAILING SIGN	SIGN IS TRAILING
WITH SEPARATE LEADING SIGN	SIGN IS LEADING SEPARATE CHARACTER
WITH SEPARATE TRAILING SIGN	SIGN IS TRAILING SEPARATE CHARACTER
FLOATING-POINT n $n = 1$ to 6 . (see note 4)	COMP-1
FLOATING-POINT n $n = 7$ to 16 . (see note 4)	COMP-2
PICTURE	(Note 5)

q is the number of characters. Where an ITEM is defined as variable length, the maximum length is used.

n and m represent the number of decimal digits before and after the decimal point respectively.

Repetition symbols are used in generated pictures if there would otherwise be four or more identical adjacent symbols; for example, 9999 would be replaced by 9(4). This limit of four is tailorable, but a repetition factor of (1) may not be generated.

Notes

1. An ITEM whose *form-description* is BITS or HEXADECIMAL generates a character field of the length that is required to contain the number of bits that constitute the item. If BITS items are defined as UNALIGNED or NOT-ALIGNED in their containing GROUP or FILE members, or RNDBIT=NO is specified in the installation macro, then BITS items are still generated in the source code as CHARACTER fields, but any record layouts that are generated are not be consistent with the source code.
2. A sign indicator (S) is generated from BINARY and PACKED-DECIMAL items unless UNSIGNED appears in the ITEM member definition (unless tailored). For NUMERIC-CHARACTER items, a sign indicator is generated only if SIGNED appears in the ITEM member definition (unless tailored).
3. Unless tailored, storage allocations for BINARY items are rounded up, using the same algorithm as a COBOL compiler; thus:
 - If $(n + m) = 1$ to 4, 2 bytes (half word) are allocated
 - If $(n + m) = 5$ to 9, 4 bytes (full word) are allocated
 - If $(n + m) = 10$ to 18, 8 bytes ($2 \times$ full word) are allocated

If tailored (RNDBIN = NO), then unaligned BINARY item members containing 1 or 2, and 5 or 6 decimal digits—corresponding to minimum storage requirements of one and three bytes respectively—are not rounded up to an even number of bytes. Instead they are translated into PICTURE X and PICTURE X(3) clauses, together with a warning message. However, even if RNDBIN=NO has been defined, BINARY items which are aligned (see [note 4](#)) or which have over ten decimal digits are rounded as stated above.

The rounding of binary items is defaulted for COBOL, but not for Assembler, PL/I, or language-independent record layouts. Thus the computed storage allocations for one and three byte unaligned BINARY item members vary depending on whether or not the generation is for COBOL.

4. Processing of COBOL binary items (COMPUTATIONAL, COMPUTATIONAL-1, and COMPUTATIONAL-2), may be optimized by use of the COBOL SYNCHRONIZED clause. Where a repository BINARY or FLOATING-POINT ITEM is defined as "aligned" (by direct reference from its containing GROUP or FILE member which includes an ALIGNED keyword in its definition), then a SYNCHRONISED clause is generated for it.
5. A COBOL PICTURE clause that is equivalent to the ITEM's PICTURE clause is generated. A T or R symbol in an ITEM's picture is replaced by a 9 symbol in the generated COBOL picture. If the first symbol of an ITEM's picture is T, then an S symbol is added to the left-hand side of the generated COBOL picture and a SIGN IS LEADING clause is added to the whole data description. If the last symbol of an ITEM's picture is T, then an S symbol is added to the left-hand side of the COBOL picture.

6. If $n+m$ is greater than the COBOL maximum (18), then PICTURE X (b) is generated, where b is the number of bytes of storage needed by the item.

COBOL Generated from ITEM's CONTENTS Clause

ITEM Member Definition in Repository	COBOL Source
CONTENTS IS <i>value</i> ... or	<i>data-description</i>
CONTENTS RANGE <i>value</i> TO <i>value</i>	VALUE <i>value</i>
(The CONTENTS clause does not include an associated CONDITION-NAME or IF clause.)	
<i>value</i> is a keyword (note 1) or a literal (see note 2). If a list or range of values is specified, the first value is generated (see note 3).	
If CONTENTS clause is not as above or is omitted, and form-description is ALPHABETIC, ALPHANUMERIC, ALPHAMERIC, or CHARACTER, or is an alphabetic or alphanumeric or numeric edited PICTURE (see note 3).	<i>data-description</i> VALUE SPACE(S)
If CONTENTS clause is not as above or is omitted, and form-description is BINARY, NUMERIC-CHARACTER, PACKED-DECIMAL, DECIMAL-PACKED, FLOATING-POINT, or is a numeric PICTURE that is not edited (see note 3).	<i>data-description</i> VALUE ZERO(S)
CONTENTS... CONDITION-NAME <i>condition-name</i>	88 <i>condition-name</i> 88 <i>condition-name</i>
CONTENTS... IS <i>value</i> ... CONDITION-NAME <i>condition-name</i>	VALUE <i>value</i> ... VALUE <i>value</i> ...
CONTENTS... IS/RANGE <i>value-1</i> TO <i>value-2</i> CONDITION-NAME <i>condition-name</i>	88 <i>condition-name</i> VALUE <i>value-1</i> THROUGH <i>value-2</i>

condition-name is subject to name editing.

Unless tailored, level-88 data entries are indented one column relative to the level number of the preceding conditional variable.

See ["COBOL Generated from ITEM's NOTE and DESCRIPTION Clauses" on page 17](#) for an alternative way of generating level-88 data entries.

Notes

1. The *value* keyword may be SPACES, ZEROS, ZEROES, LOW-VALUES, HIGH-VALUES, QUOTES, or ALL literal. These will not be produced if the ITEM has no CONTENTS clause and if the VALUE=NO DGCOB macro definition is used.

2. Literals generated from delimited character strings are enclosed in double quotation marks ("), unless tailored. Where a member's character string contains an embedded double quotation mark, it is replaced with two double quotation marks in the generated literal. Where a character string contains more than the maximum of 40 characters permitted for literals generated by the export from COBOL function, the generated literal is truncated to this limit and a warning message is given.
3. In standard COBOL a VALUE clause is not generated for any elementary data description that is subordinate to a COBOL REDEFINES or OCCURS clause or is subsequent to an OCCURS DEPENDING ON clause. In COBOL II a VALUE clause can be generated for an OCCURS clause.
4. A VALUE is not generated for an ITEM with more than 18 numeric digits.

COBOL Generated from ITEM's NOTE and DESCRIPTION Clauses

Comments can be generated from NOTE and DESCRIPTION clauses in ITEM (and GROUP and FILE) members. Their generation is governed by values in the installation macro, subject to output form specification in the generation panel or to GIVING and OMITTING clauses in the PRODUCE command. See ["Comments" on page 5](#) for details of comment generation, or ["Specifying the Format and Contents of Output" on page 63](#) and ["Suppressing Specified Generation Options" on page 65](#).

Level-88 data entries can be generated from NOTE and/or DESCRIPTION clauses in ITEM members. The preferred practice is to generate level-88 data entries from CONDITION-NAME clauses; this alternative of generating them from NOTE and DESCRIPTION clauses is provided for compatibility with early ASG-DataManager (herein called DataManager) releases. (See ["COBOL Generated from ITEM's CONTENTS Clause" on page 16](#) for generation from CONDITION-NAME.)

The generation of level-88 data entries from NOTE and/or DESCRIPTION clauses is governed by values in the installation macro, subject to output form specification in the generation panel or to GIVING and OMITTING clauses in the PRODUCE command. If generation is specified, then the NOTE and/or DESCRIPTION clauses of ITEM members are scanned for lines that start with the characters 88 immediately after the opening delimiter, and these are taken as level-88 data entries. Each entry must be wholly contained within one line.

Comment lines may be interspersed with level-88 lines. For example:

```
NOTE
'THE FOLLOWING COMMENTS ARE LEVEL-88 STATEMENTS '
'88 AMERI                VALUE IS 1. '
'88 EUROPE              VALUE IS 2. '
'ORDINARY COMMENTS MAY INTERSPERSE 88S. '
'88 ENGLAND             VALUE IS 3. '
```

A level-88 data entry is output immediately after the source code generated from the member. It is output in the same format as comments generated from NOTE or DESCRIPTION lines, but without an asterisk in column 7.

If generation of both level-88 data entries and comments from NOTE and/or DESCRIPTION clauses is specified, lines commencing with 88 are not treated as comments. If generation of level-88 data entries is not specified, no distinction is made between lines in NOTE and DESCRIPTION clauses that commence with 88 and those that do not.

Level-88 data entries generated from NOTE or DESCRIPTION clauses are not subject to editing.

Generation of COBOL Fillers and Dummy Names

These are the forms of filler names, dummy names, and associated data descriptions that may be generated:

FILLER PICTURE X(<i>p</i>)	Generated when a protected member is encountered or when a filler data description to equalize or pad storage or to align binary items is required.
FILLER <i>nnnnn</i>	Generated when a blank data name results from editing operations or from the automatic deletion of illegal characters in the name. <i>nnnnn</i> is the running total of fillers generated by the current panel or PRODUCE command.
FILLER	Generated when a member name is encountered which corresponds to the rules for filler names generated by the import from COBOL function, except when the member name is that of a non-redefining redefined member. (See "Import from COBOL Function Filler Name Conversion" on page 50 for filler names generated by the import from COBOL function.)
<i>data-name</i> -FILLER	Generated as a group name when fillers have been generated to equalize storage (for unequal redefining members, for fixed arrays that are redefined, for alignment of binary items by inserting "slack bytes," or for padding floating-point items). <i>data-name</i> -FILLER is derived from the name of the member being padded, after any editing, shortened to 23 characters if necessary by dropping the middle characters.
<i>data-name</i> PICTURE X	Generated when a member is encountered with a dummy data entries record. <i>data-name</i> is derived from the member name defined in the containing member. PICTURE X reserves storage of one byte.
PICTURE X	Generated when an ITEM is encountered with no length definition.

See ["Fillers" on page 4](#) for general details of the generation of fillers.

Level Numbers

Unless otherwise specified by tailoring or in the command, COBOL level numbers are generated in this sequence:

01, 02, 04, 06, ... 48

This can be tailored by two keywords of the DGCOB installation macro:

- INCLEV specifies a level number increment in the range 1 to 99. The default is 2.
- INCLEV0 specifies whether level number increments are to begin from zero. The default YES value gives level numbers as above. The alternative NO value gives level numbers 01, 03, 05, 07 ... (Assuming INCLEV is not tailored.)

Thus, if INCLEV is set to a value i , level numbers will be generated in this sequence:

01, i , $2i$, $3i$, ...

if INCLEV0 is untailored; or in this sequence:

01, $1+i$, $1+2i$, $1+3i$, ...

if the value of INCLEV0 is tailored to NO.

The generation of level numbers can be further controlled from the PRODUCE COBOL command, by including this clause:

LEVEL nn

LEVEL nn specifies the initial level-number of the generated data descriptions. nn is an unsigned integer in the range 1 to 49 inclusive. (See [Chapter 7, "Command Specifications," on page 53.](#))

If nn is a number in whichever of the above sequences applies, then the subsequent level numbers follow in that sequence. If nn is not a number in the appropriate sequence, then this is the sequence of level numbers generated:

nn , $nn+i-1$, $nn+2i-1$, $nn+3i-1$, ...

if INCLEV0 is untailored; or in this sequence:

nn , $nn+i$, $nn+2i$, $nn+3i$, ...

if the value of INCLEV0 is tailored to NO.

Levels are automatically indented.

The generation of level-88 data entries is described in ["COBOL Generated from ITEM's CONTENTS Clause" on page 16.](#)

3

Generation of PL/I Data Descriptions

This chapter explains how to generate PL/I data descriptions and contains these sections:

Section	Page
How to Generate PL/I Data Descriptions	21
Introduction to PL/I Source Generation	22
Storage Attribute Declarations in PL/I	22
PL/I Structures and Level Numbers	22
Based Structures	23
PL/I Generation from Arrays	24
Generating PL/I Elementary Items	25
Generating PL/I INITIAL Attributes	28
Generation of PL/I Fillers and Dummy Names	29
Pointer Variables	29

How to Generate PL/I Data Descriptions

To generate PL/I data descriptions, use the export PL/I function provided by the MethodManager panel TE13000 or the PRODUCE PL/I command.

How to generate PL/I using panel TE13000 is described in that panel's Help.

For the specifications of the PRODUCE command, see [Chapter 7, "Command Specifications," on page 53](#).

Introduction to PL/I Source Generation

PL/I source data descriptions can be generated from repository FILE, GROUP, and ITEM members, or members of user-defined member types based on FILE, GROUP, or ITEM. In the case of FILES, generation only takes place if no ITEMS are declared in the CONTAINS clause of the FILE definition. If any ITEMS are declared in the FILE's CONTAINS clause, then a message is issued, generation is suppressed and processing is abandoned for that member-name.

PL/I source data descriptions can be generated with the storage alignments of either the check-out/optimizer compilers or the F level compiler.

The PL/I 60-character set is generated unless tailored by the installation macro.

Storage Attribute Declarations in PL/I

PL/I storage attribute keywords (ALIGNED, UNALIGNED, STATIC, EXTERNAL, BASED) are generated under these conditions:

- PL/I structures are generated as ALIGNED or UNALIGNED depending on the presence of the ALIGNED or UNALIGNED/NOT-ALIGNED in the GROUP (or FILE) member definition. If neither is given, the UNALIGNED attribute is generated.
- Structures generated from members defined as alternatives by use of ELSE clauses in the CONTAINS clause of their containing GROUP are generated as BASED.
- Where the keyword STATIC or EXTERNAL is included in the PRODUCT PL/I command, then all major structures are generated as STATIC or EXTERNAL respectively.
- Where the BASED clause is included in the PRODUCE PL/I command, for example:

```
PRODUCE PL/I BASED pointer-name FROM DTR009;
```

then all major structures are generated as BASED on the named pointer.

- When a variable array is defined in the CONTAINS clause of a GROUP member, then it is generated as a self-defining data structure using the PL/I REFER option.

PL/I Structures and Level Numbers

PL/I structures are generated from GROUP members. Unless declared otherwise using the LEVEL clause, the major structure level number is 1, and unless tailored, successive level-numbers are increased by two each time (1, 3, 5, 7, ...) up to a maximum level number of 99.

A PL/I restriction is that the number of levels must not exceed 15.

See ["Generation of Data Structures" on page 3](#) for details of the generation of data structures.

Based Structures

PL/I based structures are generated in two formats. This is the first format:

```
DCL
1 structure-name BASED (pointer-name),
  . . . i
```

where:

structure-name is the name of the structure to be overlaid.

pointer-name is the pointer variable name as given in the BASED clause of the PRODUCE PL/I command. It is assumed that *pointer-name* will be defined by the user.

When BASED is specified, all major based structures are generated in this format.

The second format is generated from alternative members defined in the CONTAINS clause of a GROUP member. This is the second format:

```
DCL
level-number structure-name_BASEDn BASED (structure-name_PTR),
  . . . i
```

```
DCL structure-name_PTR POINTER;
structure-name_PTR=ADDR (structure-name);
```

where:

structure-name is the name of the structure to be overlaid.

*structure-name*_BASED*n* is the generated name identifying the structures generated in this format in the current run. The suffix _BASED may be tailored.

*structure-name*_PTR is the generated name of the pointer variable. The suffix _PTR may be tailored.

Where a structure of this type is generated and its length differs from that being overlaid, a filler is generated to pad the shorter of the two structures. This is the form of the filler:

```
FILLERnnnnn CHAR (p)
```

where:

nnnnn is the running total of fillers generated during the run (1-99999).

p is the padded storage in bytes.

PL/I Generation from Arrays

PL/I structure arrays are generated from definitions of fixed arrays in the CONTAINS clause of a GROUP member.

For example, if GROUP1 and its contained member GROUP2 are defined in this way:

```
GROUP CONTAINS (3) GROUP2, ITEM1
.
GROUP CONTAINS (2) ITEM2, ITEM3
.
```

This output would be generated:

```
DCL
1 GROUP1,
  3 GROUP 2 (3),
  5 ITEM2 (2) ...,
  5 ITEM3 ...,
  3 ITEM1 ...,
```

PL/I self-defining data structures are generated from definitions of variable arrays in the CONTAINS clause of a GROUP member. This must be the form of the variable array definition:

```
CONTAINS ( item-name-a version ) item-name version
```

where *version* may be omitted for either or both of the items.

Note the PL/I restriction that *item-name-a* must appear in the same structure as the self-defining data structure.

The PL/I source code generated takes this form:

```
level-number data-name (data-name_REFER REFER (data-name-a) )
```

where:

data-name-a identifies an element-variable (derived from *item-name-a*) that is the object of the REFER option.

data-name is the data name (derived from *item-name*).

data-name-REFER identifies an element-expression defining the upper bound of the array. *data-name* is shortened, if necessary, by dropping the middle characters. The suffix *_REFER* may be tailored. The element-expression must be supplied by the user.

Note the PL/I restriction that the structure containing the self-defining data must be BASED.

Any one major PL/I structure can only contain one self-defining data structure.

Generating PL/I Elementary Items

PL/I elementary items are generated from repository ITEM members as shown in this table:

Repository ITEM Definition		PL/I Data Description
Form-description	Length Definition	Generated
ALPHABETIC ALPHANUMERIC ALPHAMERIC CHARACTER	q	CHARACTER (q)
ALPHABETIC ALPHANUMERIC ALPHAMERIC CHARACTER	p TO q	CHARACTER (q) VAR
BITS	q	BIT (q) (see note 1)
HEXADECIMAL	q	CHARACTER ($\lfloor q+1 \rfloor / 2$)
NUMERIC-CHARACTER	$n1 . n2$	PICTURE ($n1$) 9V ($n2$) 9 (see note 2)
PACKED-DECIMAL	$n . m$ ($n+m$) = 1 to 15 ($n+m$) = 16 to 31	FIXED DECIMAL ($n+m, m$) FIXED DECIMAL (15, m) (see note 3)
BINARY	$n . m$	FIXED BINARY (s, r) (see note 4)
FLOATING POINT (unless tailored)	p $p = 1$ to 6 p = 7 to 16 $p =$ 17 to 33	FLOAT BINARY (21) FLOAT BINARY (53) FLOAT BINARY (109) (see note 5)
FLOATING POINT (if tailored)	p $p = 1$ to 6 p = 7 to 16 $p =$ 17 to 33	FLOATING DECIMAL (6) FLOAT DECIMAL (16) FLOAT BINARY (33) (see note 5)
PICTURE	-	PICTURE (see note 6)

where:

q is the number of characters.

n is the number of decimal digits before the decimal point.

m is the number of decimal digits after the decimal point.

s represents the total number of digits in a fixed binary item.

r represents the number of digits after the binary point in a fixed binary item.

p represents the number of decimal digits in the mantissa.

Floating point precision shown for $p = 7$ to 16 is the maxima for the VSE compiler.

Floating point precision shown for $p = 17$ to 33 is the maxima for the VM/CMS compiler.

Notes

1. An item that is defined in the repository as a BITS ITEM may be defined as aligned by specifying the keyword **ALIGNED** in the definition of the ITEMS containing **GROUP** or **FILE** members. Fillers are not generated for padding of BITS fields.
2. The PL/I picture that is generated for a **NUMERIC-CHARACTER** ITEM will also include a sign symbol if the ITEM's form description includes these:
 - The **SIGNED** keyword
 - The **WITH SEPARATE LEADING SIGN** or **WITH SEPARATE TRAILING SIGN** clauses

This table shows the different ways in which sign symbols (T or S) are included in generated PL/I pictures:

ITEM's Form Description	Generated PL/I Pictures
NUMERIC-CHARACTER 3	'999'
SIGNED NUMERIC-CHARACTER 3	'99T'
SIGNED NUMERIC-CHARACTER 3 WITH LEADING SIGN	'T99'
SIGNED NUMERIC-CHARACTER 3 WITH TRAILING SIGN	'99T'
SIGNED NUMERIC-CHARACTER 3 WITH SEPARATE LEADING SIGN	'S999'
SIGNED NUMERIC-CHARACTER 3 WITH SEPARATE TRAILING SIGN	'999S'

3. Up to 15 decimal digits are allowed in a PL/I **FIXED DECIMAL** item. Where a repository **DECIMAL-PACKED** or **PACKED-DECIMAL** ITEM member definition of between 16 and 31 digits is encountered, padding is added in this form:

FILLERnnnnn CHAR (q)

where:

nnnnn is the running total of fillers generated in the run.

q is between 1 and 8 to reserve storage.

4. A maximum precision of 31 binary digits is allowed in a PL/I FIXED BINARY item. Thus, where a repository BINARY item of 10 to 31 decimal digits is encountered, padding is added to the containing generated structure in this form:

FILLER n CHAR (4) .

The value of s is calculated from $(n + m)$ in the same way that r is calculated from m . The corresponding values of s and $(n + m)$ are shown in this table:

$(n + m)$ or m	1	2	3	4	5	6	7	8	9	10 - 31
s or r	0	4	7	10	15	17	20	24	27	31

A repository BINARY ITEM member should be defined as ALIGNED (in its containing member), otherwise the storage allocation computed from it will differ from that computed by a PL/I compiler, as shown in the table below. This is important where binary items are defined in based structures, as Manager Products generate fillers to equalize storage of the redefined and redefining structures.

Repository BINARY $n . m$ Definition	PL/I Storage Unaligned	Allocation Aligned
$(n + m) = 1,2$	1 byte	2 bytes (half word)
$(n + m) = 3,4$	2 bytes	2 bytes (half word)
$(n + m) = 5,6$	3 bytes	4 bytes (full word)
$(n + m) = 7,8,9$	4 bytes	4 bytes (full word)

The rounding up of binary item lengths from 1 to 2 bytes and from 3 to 4 bytes is controlled by the installation macro keyword RNDBIN. RNDBIN=YES is defaulted for COBOL and IMS, and RNDBIN=NO is defaulted for PL/I, BAL, MARK IV, and record layouts only (that is, RECORD-LAYOUTS clause with no language clause).

5. PL/I floating point items are generated as FLOAT BINARY, unless tailored. When tailored, FLOAT DECIMAL will be generated instead.

6. A PL/I PICTURE is the same as the repository PICTURE it is generated from, except for these points:
 - Repetition factors are written before the symbols to which they refer; for example, A(5) becomes (5)A.
 - Where a string of *n* or more identical picture symbols is encountered in a definition, it is converted for PL/I to a single symbol and a repetition factor; for example, ZZZZ becomes 4(Z). *n* is four unless tailored. This limit of 4 is tailorable, but a repetition factor of (1) may not be generated.

Generating PL/I INITIAL Attributes

If tailored or specified in the generation panel or in the PRODUCE command, PL/I INITIAL attributes are generated for elementary data descriptions in this way:

- If an ITEM member has a CONTENTS clause that contains an IS clause or a RANGE clause without an associated CONDITION-NAME or IF clause, then the literal value (or the first literal value of a list or range of values) is generated as:

elementary-data-description INIT *literal*

- If an ITEM member has a CONTENTS clause that does not fulfill the above conditions, or has no CONTENTS clause at all, or has more than the maximum number of allowed numeric digits, or the ITEM is a dummy, then either INIT (' ') or INIT (0) is generated according to the form-description of the form and version of the ITEM being processed:

ITEM Form-description	PL/I INITIAL Clause
ALPHABETIC ALPHANUMERIC ALPHAMERIC CHARACTER	INIT (' ')
BITS	INIT ('0'B)
BINARY NUMERIC-CHARACTER PACKED-DECIMAL DECIMAL-PACKED FLOATING-POINT	INIT (0)
PICTURE	Depends on picture type:
alphabetic	INIT (' ')
alphanumeric	INIT (' ')
numeric	INIT (0)
edited	No INITIAL clause is generated.

Non-numeric literals are generated enclosed in single quotes (' '). Where a member's character string contains an embedded single quote, it is replaced with two single quotes in the generated literal.

The maximum length of a PL/I non-numeric literal is 62 characters. Where a member's character string contains more than this number of characters, the corresponding generated PL/I literal is truncated to this limit.

Generation of PL/I Fillers and Dummy Names

In the table below, *nnnnn* is the running total of fillers generated by the current generation panel or PRODUCE command.

Filler or Dummy Name	Description
FILLER <i>nnnnn</i> CHAR(<i>p</i>)	Generated when a protected member is encountered or when a filler data description to equalize or pad storage is required.
FILLER <i>nnnnn</i>	Generated when a blank data name results from editing operations or from the automatic deletion of illegal characters in the name. Also generated when a member name is encountered which corresponds to the rules for export from COBOL function filler names.
<i>data-name</i> CHAR(1)	Generated when a member is encountered with a dummy data entries record. <i>data-name</i> is derived from the member name defined in the containing member. CHAR(1) reserves storage of one byte.
CHAR(1)	Generated when a member is encountered with no length definition.

See ["Fillers" on page 4](#) for general details of the generation of fillers.

Pointer Variables

PL/I pointers can be generated from ITEM members whose HELD-AS form is defined as a binary full word with a USAGE POINTER clause; for example:

```
ITEM HELD-AS BINARY 8 USAGE POINTER
;
```

The generated output has this form:

```
DCL n item-name PTR
```

where *n* is the level number, and DCL is present only if *n* = 1.

4

Generation of Assembler Data Descriptions

This chapter explains how to generate Assembler data descriptions and contains these sections:

Section	Page
How to Generate Assembler Data Descriptions	31
Introduction to Assembler Source Generation	31
Assembler Generation from GROUPs	32
Assembler Generation from Arrays	33
Assembler Generation from ITEMS	33
Assembler Edit Patterns	35
Generation of Assembler EQU Statements	37
Generation of Assembler DC Statements	38
Generation of Assembly Fillers and Dummy Names	40

How to Generate Assembler Data Descriptions

To generate Assembler data description, use the export Assembler function provided by MethodManager panel TE12000 or the PRODUCE ASSEMBLER command.

How to generate Assembler using panel TE12000 is described in that panel's Help.

For specifications of the PRODUCE command, see [Chapter 7, "Command Specifications," on page 53](#).

Introduction to Assembler Source Generation

Assembler source data descriptions can be generated from repository GROUP and ITEM members or members of user-defined member types based on GROUP or ITEM.

A basic subset of the Assembler language is generated, comprising:

- DC
- DS
- EQU
- ORG

Statements are generated with single operands only.

If tailored, or if specified in the generation panel or the PRODUCE command, Assembler edit patterns are generated from numeric edited PICTUREs.

Assembler Generation from GROUPs

Generation from a GROUP member is of a named storage area (without any actual storage allocation) using a zero duplication factor in a DS statement; thus:

```
group-name DC OCLn
```

where *n* is the size of the area, up to a maximum of 65,535 characters. Where *n* is greater than 65,535 characters, a DS *n*C statement is generated.

Alternative GROUP members defined by ELSE clauses in the CONTAINS clause of the containing GROUP rename the storage area; for example:

If this was the GROUPA member definition:

```
GROUP CONTAINS GROUPB ELSE GROUPC  
;
```

and Assembler was generated from it, this would be the output:

```
GROUPA DS OCLn  
GROUPB DS OCLm  
GROUPC DS OCLp
```

or, if interspersed with DS statements generated from contained ITEM members:

```
GROUPA DS OCLn  
GROUPB DS OCLm  
    . . .  
    . . .  
    ORG GROUPA  
GROUPC DS OCLp
```

The ORG statement ensures that GROUPC renames the start of the notational area GROUPA.

The next section discusses Assembler generation from arrays.

Assembler Generation from Arrays

Where the CONTAINS clause of a GROUP member defines a fixed array, as:

```
(n) member-name
```

then action depends on whether the member is an ITEM or GROUP.

- Where *member-name* is an ITEM, generation is of a DS statement with a duplication of *n*. For example, generation from:

```
(10) ITEM1
```

where ITEM1 has a form-description of CHARACTER 4, would be:

```
ITEM1 DS 10CL4
```

- Where *member-name* is a GROUP, generation is of the Assembler definition for that group followed by a DS statement in this form:

```
blank DS CLm
```

where *m* is the number of bytes necessary to allocate storage sufficient for the number of times the group recurs; that is, $m = (n - 1) \times (\text{the length of the group})$.

Generation from variable arrays is not supported. Where a variable array definition:

```
( item-name ) member-name
```

or

```
( item-name version ) member-name
```

is encountered in the CONTAINS clause of a GROUP member, then the Assembler description for a single occurrence of *member-name* is generated preceded by this comment:

```
THE FOLLOWING GROUP OCCURS item-name TIMES
```

Assembler Generation from ITEMS

The elements of repository member definitions that generate particular elements of Assembler code are set out in the following table. Where a variable length ITEM definition is encountered, the maximum length is used.

If tailored to give initial values, or if so specified in the generation panel or command, DC statements may be generated in place of DS statements. See "[Generation of Assembler DC Statements](#)" on page 38.

Repository ITEM Definition		Assembler Data Description			
Form-description	Length Definition	Generated			
ALPHABETIC ALPHANUMERIC ALPHAMERIC CHARACTER	q	DS	CL	q	
BITS	q	DS	BL	(q)	(see note 1)
HEXADECIMAL NUMERIC-CHARACTER	q	DS	XL	$(\lfloor (q+1)/2 \rfloor)$	
SIGNED	$n.m$	DS	ZL	$(n + m)$	
UNSIGNED	$n.m$	DS	CL	$(n + m)$	
PACKED-DECIMAL DECIMAL-PACKED	$n.m$	DS	PL	$(\lfloor (n+m+2)/2 \rfloor)$	
BINARY	$n.m$		<u>Aligned</u>		<u>Unaligned</u>
	$n+m = 1, 2$	DS	H		DS BL1
	$n+m = 3, 4$	DS	H		DS HL2
	$n+m = 5, 6$	DS	F		DS FL3
	$n+m = 7, 8, 9$	DS	F		DS FL4
	$n+m = 10$ to 31	DS	DL8		DS DL8
					(see note 2)
FLOATING POINT	n				(see note 3)
	$n = 1$ to 6	DS	EL4		
	$n = 7$ to 16	DS	DL8		
PICTURE	-	DS	CL	p	

where:

q is the number of characters.

n represents the number of decimal digits before the decimal point.

m represents the number of decimal digits after the decimal point.

p represents the number of characters implied by the picture.

Notes

1. An ITEM defined as BITS is generated as a bit length specification field. The length of the field would be the number of whole bytes required to hold the contents of the BITS field.

If BITS ITEMS are defined as UNALIGNED or NOT-ALIGNED in their containing GROUP or FILE members, or if RNDBIT=NO is specified in the installation macro DGBAL, BITS ITEMS are still generated in the source code as bit length specification fields, but any record layouts generated will not be consistent with the source code.
2. When an aligned binary ITEM is defined with a length of 10 through 31, it is generated as DL8 and is forced onto a full-word boundary. This allows for the same record layouts to be used for both COBOL and PL/I as well as Assembler.
3. FLOATING-POINT ITEM members generate aligned Assembler definitions, regardless of whether they are defined in their containing GROUP members as ALIGNED or NOT-ALIGNED. When an alignment attribute is defined in the containing GROUP, only BINARY ITEM members are affected, as shown in the table. If tailored, unaligned BINARY ITEM definitions that would otherwise generate the operand BL1 (see the table on page 34), instead map to HL2.

Assembler Edit Patterns

If tailored, or if so specified in the Assembler generation panel or command, Assembler decimal edit patterns are generated from numeric edited PICTURE definitions. If generated, edit patterns are output in a block immediately after the source code generated for the member named in the panel or command.

Edit patterns are output in this way:

```
item-nameEP DC X'ffssxxxx...'
```

where:

*item-name*EP is the item name after any editing, shortened by dropping middle characters if necessary and suffixed with EP (unless tailored)

ffssxxxx is the generated edit pattern as an unpunctuated list of hexadecimal values, in which:

ff is the fill character.

ss is a significance starter or digit selector.

xx is a message character, digit selector, or significance starter.

Where the edit pattern is more than seven hexadecimal values, successive DC statements are generated.

Assembler edit patterns are generated from the ITEM PICTURE symbols shown in this table:

PICTURE Symbol	Hexadecimal Edit Pattern	Comments
9 I R T	20 (digit selector)	If PICTURE starts with 9, I, R, or T, <i>SS</i> is set to 21; otherwise to 20. I, R, and T lose conditional meanings. (see note 2)
Y Z	20 or 21 (digit selector or significance starter, depending on context)	
S + - \$ / .,	@E (message char) 4E 60 5B 61 (message char) 4B 6B	S, +, - and \$ lose conditional meanings (see note 2). See note 3 for symbols in float strings.
B CR DB 0	40 (message char) C3D9 C4C2 F0	B, CR, DB, and 0 lose conditional meanings (see note 2).
P V *	F0 (message char) - 20 or 21 (digit selector or significance starter, depending on context)	P loses its meaning. V is ignored.

Notes

- Edit patterns generated from these PICTURE symbols:

9 Y /) X , *

form a complete and compatible subset in terms of the repository definition.

2. Edit patterns generated from these PICTURE symbols:

I R T B CR DB 0 + - \$ E

are compatible with the repository definition, but are incomplete in that alternative edit patterns will need to be coded to meet conditional data-dependent requirements (for example, to replace CR with spaces if the value to be output is positive).

3. Edit patterns generated from these PICTURE symbols:

\$ * - £

are not compatible with their use in repository float strings. Where a string of the same symbol is encountered (for example, \$\$\$\$), the pattern generated is the same as would be produced by a string of Zs (for example, ZZZZ).

4. Assembler edit patterns operate on packed decimal numbers packed two digits per byte. If an edit pattern is generated with an odd number of digit selectors following the significance starter, a message is given that an additional source character is needed, and the significance starter is preceded by a digit selector to ignore the unused half byte.

Examples of edit patterns generated from an ITEM named ITEMA with the PICTURE clauses shown:

- 1 PICTURE '9(2)/(1)9(2)/(1)9(2)' generates
 ITEMAEP DC X'40212020612020'
 DC X'612020'
- 2 PICTURE 'E(1)Z(2)9(2)' generates
 ITEMAEP DC X'40205B20212020'
- 3 PICTURE '* (2)9(2)V(1)9(2)' generates
 ITEMAEP DC X'5C202021202020'
 DC X'20'
- 4 PICTURE 'E(3)9(2)' generates
 ITEMAEP DC X'40202020212020'
 * FLOATING CHARACTER LOST

Generation of Assembler EQU Statements

Unless tailored, or unless specified otherwise in the PRODUCE command, EQU statements are generated from CONDITION-NAME clauses in the CONTENTS clause of the ITEM definition. They immediately follow the DS statement generated from that ITEM.

This is the format:

```

item-name          DS    ...
condition-name-1  EQU   value
condition-name-2  EQU   value

```

where *condition-name-1* and *condition-name-2* are names, after any editing, declared in a CONDITION-NAME clause. (See ["Name Editing Options Overview" on page 68](#), ["Replacing Names or Name Elements" on page 69](#), ["Dropping Names or Name Elements" on page 69](#), ["Inserting Characters Into Names" on page 70](#), and ["Conditional Editing" on page 70](#) of the PRODUCE command specifications for name editing options.)

Each condition name must have an associated IS clause for the EQU statement to be fully defined. If the CONDITION-NAME clause is specified with a RANGE clause or a list of values, or no value at all, no EQU statement is generated.

Generation of Assembler DC Statements

If tailored, or if so specified in the Assembler generation panel or command, DC statements are generated in place of DS statements in this way:

- If an ITEM definition has a CONTENTS clause that contains an IS clause or a RANGE clause, without an associated IF clause, then:
 - The literal value (or the first literal value of a list or range of values) is generated as:

DC data-description literal
 - For a BINARY form-description with one or two decimal characters, a data description of HL1 is generated instead of B, followed by the literal.
 - For a BINARY, PACKED-DECIMAL, DECIMAL-PACKED, or NUMERIC-CHARACTER form-description, where the length declaration indicates decimal digits after an implied decimal point, the literal is generated with any preceding or following zeros needed to align the decimal point.

- If an ITEM definition has a CONTENTS clause that does not fulfill the above conditions, has no CONTENTS clause, or if the ITEM is a dummy, then either a zero or a space literal is generated according to the form-description of the form and version of the ITEM being processed.

Form-description	Assembler Literal Generated
ALPHABETIC ALPHANUMERIC ALPHAMERIC CHARACTER	DC <i>data-description</i> ' '
BINARY PACKED-DECIMAL DECIMAL-PACKED FLOATING-POINT BITS HEXADECIMAL	DC <i>data-description</i> '0'
NUMERIC-CHARACTER	DC <i>data-description</i> '0...' (Note)
PICTURE	Depends on picture type:
alphabetic	DC <i>data-description</i> ' '
alphanumeric	DC <i>data-description</i> ' '
numeric	DC <i>data-description</i> '0'
edited	DC <i>data-description</i>

Note: _____

For NUMERIC-CHARACTER, a zero is generated for each character specified in the data-description.

Non-numeric literals are generated enclosed in single quotes (' '). Where a member's character string contains an embedded single quote, it is replaced with two single quotes in the generated literal.

The maximum length of an Assembler non-numeric literal is eight characters. Where a character string in an ITEM contains more than eight characters, it is truncated and a warning message is given.

Generation of Assembly Fillers and Dummy Names

These are the forms of filler and dummy names and data descriptions that may be generated for Assembler:

(blank) DS CLp	Generated where a protected member is encountered, or where a filler is required to equalize or pad storage.
(blank)	Generated where a <i>member-name</i> is encountered that duplicates a member name already processed, or that corresponds to the rules for filler names generated by the import from COBOL function.
<i>data-name</i> DC CL1	Generated where a member with a dummy data entries record is encountered, <i>data-name</i> is derived from the member name defined in the containing member, and the data description reserves storage of one byte.
DMFILLER	Generated where a member name declared in the generation panel or command is edited (whether by the use of editing options or by the automatic deletion of illegal characters from the name) to leave a blank name.
DS CL1	Is the data description generated where a member with no length definition is encountered.

See ["Fillers" on page 4](#) for general details of the generation of fillers.

5

Generation of Record Layouts

This chapter explains how to generate record layouts and contains these sections:

Section	Page
How to Generate Record Layouts	41
Record Layouts: Overview and Example	42
Fields in Record Layouts	43
Format of the Generated Layout	45

How to Generate Record Layouts

To generate language-independent record layouts, use the export function provided by MethodManager panel TE14000, or issue a PRODUCE RECORD-LAYOUTS command.

Language-dependent record layouts can also be generated, alone or in conjunction with COBOL, PL/I, or Assembler source data descriptions. For this, the expert mode panel E1E3000 or a PRODUCE RECORD-LAYOUTS command with additional keywords is used.

How to generate record layouts using panels TE14000 or E1E3000 is described in those panels' Help.

For the specifications of the PRODUCE command, see [Chapter 7, "Command Specifications," on page 53](#).

Record Layouts: Overview and Example

Record layouts are tables containing these details:

- The levels and names of the GROUPs and ITEMs from which the record layout was generated
- The decimal and hexadecimal offsets of the storage fields for the groups and items in the record
- The length, type, and alignment of the groups and items
- Remarks giving additional information about the member

The layouts can be tailored to your own installation requirements by using installation macros.

[Figure 1](#) shows an example of a record layout generated from the group EMP-IDENT on the DEMO repository.

Figure 1.. Example Record Layout Generated from Group EMP-IDENT

```

*****
*
*      DESCRIPTION OF EMP-IDENT
*
*****
*      *      *      *      *      *
* DEC  *HEX  *      *      *      *      *
* OFFSET*OFFSET* LEVEL & NAME      *LENGTH* TYPE * ALIGN*      REMARKS
*      *      *      *      *      *
*****
*      0 *    0 * 1 * EMP-IDENT      * 102 * GROUP*      *
*-----*-----*-----*-----*-----*
*      0 *    0 * 2 * EMP-NAME      * 30 * CHAR *      *
*-----*-----*-----*-----*-----*
*     30 *   1E * 2 * EMP-ADDR      * 50 * CHAR *      *
*-----*-----*-----*-----*-----*
*     80 *   50 * 2 * TELEPHONE      * 22 * GROUP*      *
*-----*-----*-----*-----*-----*
*     80 *   50 * 3 * HOME-TEL-NO    * 11 * NUM *      * 11 DIGITS
*-----*-----*-----*-----*-----*
*     91 *   58 * 3 * OFF-TEL-NO     * 11 * NUM *      * 11 DIGITS
*****

```

Record layouts can be generated alone or together with source language data descriptions. When generated together with source language data descriptions, the record layouts are output first; data names are edited if necessary to conform to the rules of the specified language.

These are the names produced in a record layout:

- Member names
- If tailored or if specified in the generation panel or command, language specific aliases, for those members that have them
- If tailored or if specified in the generation panel or command, context specific local names, for those members for which such names are declared in KNOWN-AS clauses in the containing member

If so specified in the generation panel or command, or if tailored by the installation macros, the horizontal and vertical boxing lines may be omitted, additional line spacing may be inserted between members, and the number of printing lines per page may be adjusted.

Fields in Record Layouts

These are the fields in the record layouts:

Field	Description
DEC/HEX OFFSETS	Are the offsets (unless tailored to start positions) in decimal and hexadecimal, of the storage fields computed for the members constituting a storage block (or record).
LEVEL AND NAME	Is the hierarchical level of the field, followed by the generated name of the field. Level numbers begin at one (the member from which record layouts are being produced), and are increased by one for successively lower levels.
LENGTH	Is the computed length of the storage field in bytes. For a GROUP member, the length is the sum of the lengths of the ITEMS directly or indirectly contained by it. For variable length items, the maximum length is used. For fixed arrays, the length is the repetition factor multiplied by the array element length. For variable arrays the length is that of a single occurrence of the array element. For dummy members, a length of one byte is defaulted. Computation of field lengths for binary items is source language dependent. (See " COBOL Generation from ITEMS " on page 14, " Generating PL/I Elementary Items " on page 25, or " Assembler Generation from ITEMS " on page 33.)
TYPE	Is the member's form-description, if the member is an ITEM; otherwise the member type.
ALIGN	Shows whether the ITEM is aligned in storage and on what word boundary (half word, full word, or double word), as declared by the keyword ALIGNED in its containing member.

REMARKS

The remarks column gives additional information about the member, including:

- The member's length definition for numeric ITEM members (for example 2 BITS, 16.2 DIGITS).
- The sign of numeric ITEM members (for example 10 DIGITS SIGNED, 14.1 DIGITS UNSIGNED). This information is only given when record layouts are generated in association with COBOL.
- The member's PICTURE definition for ITEM members (unless tailored) DUMMY ENTRY, indicating that the member is a dummy.
- The repetition factor of arrays declared in the CONTAINS clause of a GROUP or FILE member (for example OCCURS 10 TIMES, OCCURS VAR TIMES).
- The length of a variable length array.
- NOTE and/or DESCRIPTION clauses defined for the member. This information is only given if selected in the generation panel or command, or if tailored.

Format of the Generated Layout

There are three options for the format of the layout generated; the option is chosen using the keyword WIDEFMT in the installation macro DGREC.

- With WIDEFMT=NO existing layouts are enforced, even if the structure is too long to display within the layout (see [Figure 1 on page 42](#)). This may lead to an error condition if the offsets or lengths are too large to display.
- With WIDEFMT=YES a wide format layout is selected, irrespective of the size of the structure (see [Figure 2](#)).
- With WIDEFMT=AUTO the wide format is selected only if the structure is large enough to require it.

Figure 2.. Example Record Layout in Wide Format

```

*****
*
*   DESCRIPTION OF EMP-IDENT
*
*****
*   *   *   *   *   *   *
* DEC  * HEX  *   *   *   *   *
* OFFSET * OFFSET * LEVEL & NAME * LENGTH * TYPE * ALIGN *   REMARKS
*   *   *   *   *   *   *   *
*****
*   0 *   0 * 1 * EMP-IDENT-TABLE * 27372 * GROUP *   *
*-----*-----*-----*-----*-----*-----*-----*
*   0 *   0 * 2 * EMP-NAMES * 30 * CHAR *   * OCCURS 910 TIMES
*-----*-----*-----*-----*-----*-----*-----*
* 27300 * 6AA4 * 2 * EMP-ADDR * 50 * CHAR *   *
*-----*-----*-----*-----*-----*-----*-----*
* 27350 * 6AD6 * 2 * TELEPHONE * 22 * GROUP *   *
*-----*-----*-----*-----*-----*-----*-----*
* 27350 * 6AD6 * 3 * HOME-TEL-NO * 11 * CHAR *   * 11 DIGITS
*-----*-----*-----*-----*-----*-----*-----*
* 27361 * 6AE1 * 3 * OFF-TEL-NO * 11 * NUM *   * 11 DIGITS
*****

```

6

Tailoring Source Language Generation

This chapter explains how to tailor source language generation and contains these sections:

Section	Page
Installation Macros	47
Source Library Dataset Control	48
Record Layouts Tailoring	48
Source Language Output Format Tailoring	49
Import from COBOL Function Filler Name Conversion	50
Output Source Language Tailoring	51

Installation Macros

These are the relevant installation macros for tailoring programming source language and record layouts generation:

- DGCOD for COBOL
- DGPLI for PL/I
- DGBAL for Basic Assembler Language
- DGREC for language-independent record layouts

Each of these macros has a series of keywords with default values, for which alternative values can be declared. The way in which these macros are applied is described in the Manager Products installation manual applicable to your environment. That manual also contains complete definitions of all the keywords and their default and alternative assignable values.

This chapter summarizes the available keywords, grouped by function. Many of the keywords are common to all four macros.

Source Library Dataset Control

These keywords are defined in DGCOB, DGPLI, and DGBAL:

Keyword	Specifies
ACSMETH	Type of file, BPAM, or QSAM.
CONCARD	Whether a catalog control card is generated.
DDNAME	Default library file name.
LIBCC	Default catalog control card.
MEMLEN	Maximum length of library name.

Record Layouts Tailoring

These keywords are defined in DGCOB, DGPLI, DGBAL, and DGREC unless otherwise stated. If language-independent record layouts are generated, the DGREC keyword values apply. If record layouts are generated for or with a programming source language, then the keyword values of DGCOB, DGPLI, or DGBAL apply, depending on the source language specified.

Keyword	Specifies
ALIAS	Output language-specific alias instead of member name (not in DGREC).
DESC	Maximum number of DESCRIPTION strings to be output as remarks or comments.
MAXLEN	Maximum length of any data structure.
KNOWNAS	Output local name instead of member name.
NOTE	Maximum number of NOTE strings to be output as remarks or comments.
RECBOX	Insert horizontal and vertical boxing lines.
RECPGSI	Number of lines per page.
RECPIC	Whether to include PICTUREs in record layouts.
RECPOS	Output offsets or start positions of fields.
RECSP	Line spacing between members.
RNDBIN	Whether binary items are to be rounded up.
RNDBIT	Whether bit string fields are generated with byte alignment.

Note: _____
Where both apply, KNOWNAS takes precedence over ALIAS.

Source Language Output Format Tailoring

Installation macro keywords controlling the output format of generated source programming language are listed below:

Keyword	Specifies
In DGCOB, DGPLI, and DGBAL:	
COLMAIN	Start column for COBOL PICTURE and VALUE clauses, PL/I attributes or Assembler operation codes.
COLSUBS	Start column for subsequent COBOL or PL/I statement elements or Assembler operands.
COLNOTE	Start column for comments.
COLSEQ	Start column for sequence number.
INCRSEQ	Sequence number increment.
LENSEQ	Length of sequence number field.
SEQNOQ	Whether sequence numbering is required.
In DGCOB and DGPLI:	
COL01	Starting column for 01 level number.
INCLEV	Level number increment.
OFFSUBS	Offset for subsequent level numbers.
SPACING	Number of spaces between statement elements.
In DGCOB only:	
COL2ND	Starting column for second level number.
COLCOND	Whether level-88 statements generated from CONDITION-NAME clauses are to be output in a fixed position or in a position relative to the generated conditional variable.
COLCPOS	As determined by COLCOND, the starting column or the offset of level 88 statements.
INCLEV0	Whether level numbering increments are to begin from zero or from 01.
RECFMGEN	Whether the RECORD FORMAT clause is generated unconditionally.
In DGPLI only:	
MEMBLEN	Maximum length of binary item.
In DGBAL only:	
NAMEMAX	Maximum length of generated data names.
In DGREC only:	
WIDEFMT	Record layout generation formatting option.

Import from COBOL Function Filler Name Conversion

Unless the value of the AUTOCHK keyword is tailored, ITEM and GROUP member names processed by an export function are checked against a set of pre-specified names defined by installation macro keywords ATRUNK to GFNL (see below). This set of names represents filler names that may have been inserted into the repository via the import from COBOL function from COBOL source code. Where a GROUP or ITEM with such a filler name is found (before any name editing is applied in the generation process) it is converted according to these rules:

COBOL. Names of members representing fillers are converted back to FILLER.

PL/I. Names of members representing fillers are converted back to FILLER $nnnnn$ (where $nnnnn$ is the running total of fillers generated by the current generation panel or command).

Assembler. Names of members representing fillers are converted to blanks.

These keywords are defined in DGCOB, DGPLI, and DGBAL:

Keyword	Specifies
AUTOCHK	Whether to check for and convert fillers.
	The character part of the item filler name for the indicated types:
ATRUNK	ALPHABETIC
BTRUNK	BINARY
CTRUNK	CHARACTER
DTRUNK	DECIMAL-PACKED
FTRUNK	FLOATING-POINT
NTRUNK	NUMERIC-CHARACTER.
IFNL	The length of the number part of item filler names.
GTRUNK	The character part of group filler names.
GFNL	The length of the number part of group filler names.

Output Source Language Tailoring

These are the installation macro keywords controlling the generated source programming language output:

Keyword	Specifies
In DGC0B, DGPLI, and DGBAL:	
ACHAR	Additional characters to those in the source language character set, to be accepted for output in names.
ACHAR2	Any additional valid characters, beyond those specified for ACHAR.
INITVAL	Whether VALUE clauses (COBOL), INITIAL attributes (PL/I), or DC statements are to be generated from ITEM members.
In DGC0B and DGPLI:	
MAXSYM	Maximum number of PICTURE symbols before a repetition factor is used.
KEYABB	Whether keyword abbreviation is required.
In DGC0B only:	
BINSIGN	Whether BINARY ITEMS are to be signed in the PICTURE clause.
COBOL2	Whether the generated code is COBOL II or VS COBOL.
COMP	In COBOL II whether computational or binary/packed decimal keywords are generated.
COND88	Whether level-88 statements are to be generated from CONDITION-NAME clauses in ITEMS.
DDESC88	Whether level-88 statements are to be found in DESCRIPTION clauses in ITEMS.
DECOMMA	Whether the decimal point is represented by a full stop (period) or a comma.
DNOTE88	Whether level 88 statements are to be found in NOTE clauses in ITEMS.
FILESUF	Suffix for file name when 01 data-name is automatically generated.
GEN	Whether both FD and 01 levels, or FD only, or 01 only, are to be generated from FILES.
NEZEROS	Whether to assume a default value for numeric edited items of ZEROS (YES) or SPACES (NO). This option is only relevant where the item has a picture clause including editing symbols, no contents clause is present and initial values are required.
NUMSIGN	Whether NUMERIC ITEMS are to be signed in the PICTURE clause.
PCKSIGN	Whether PACKED-DECIMAL ITEMS are to be signed in the PICTURE clause.
VALUE	Whether, on giving initial values, blank or zero contents are generated when no contents are specified in the item.

Keyword	Specifies
QUOTES	Whether generated non-numeric literals are to be enclosed within single or double quotes.
REDFILL	Whether unequal redefining members are to be padded with fillers.
ULABNAM	Name for use in FD LABEL-RECORDS clause when USER-LABELS is specified.
In DGPLI only:	
BSDSUF	Suffix for BASED structure names.
CHARSET	Whether to use a 60- or 48-character set.
FLOATYP	Whether BINARY or DECIMAL FLOAT is generated.
PNTRSUF	Suffix for POINTER names.
REFSUF	Suffix for PL/I REFER option.
In DGBAL only:	
EPATSUF	Edit pattern suffix.
EPPROD	Whether edit patterns are to be generated.
EQUATE	Whether EQU statements are to be generated from CONDITION-NAME clauses in ITEM members.

Some of the keywords listed in earlier sections of this chapter also impact the output source language generated.

7

Command Specifications

This chapter describes the Manager Products commands relevant to source language generation. The commands are documented in alphabetical order of command name. These are the commands:

Section	Page
PRODUCE Command	53
SHOW PRODUCE-OPTIONS	76

PRODUCE Command

The PRODUCE command is used to generate record layouts and/or programming source language statements or database management system language statements from members of the repository. The generation of record layouts and COBOL, PL/I, and Assembler data descriptions in conventional file environments is described here. For the use of the PRODUCE command in other environments, see "[What IS Source Language Generation?](#)" on page 1. Refer to "[PRODUCE Syntax](#)" on page 72 for the syntax of the PRODUCE command.

Generic Overview of the PRODUCE Command

This is the general form of the PRODUCE command:

```
PRODUCE context qualifier FROM member-name-list control-options ;
```

where:

context is a keyword identifying the system context (record layouts, programming source language, database management system, file management system) in which the command is being used.

qualifier is a context-dependent keyword or clause, or a number of such keywords and/or clauses, that determine(s) the type of output produced. Some contexts do not require a qualifier.

member-name-list identifies from one to sixteen encoded members from which generation is to take place, and optionally defines names to be given to the generated library members in the output file.

control-options are optional keywords or clauses that control the operation of the command. They can be sub-divided into:

- Output control options
- Generation control options
- Name editing options

COBOL Generation

Enter this command to generate standard COBOL data descriptions:

```
PRODUCE COBOL LEVEL nn FROM member-name-list ;
```

Enter this command to generate COBOL II data descriptions:

```
PRODUCE COBOL2 EXTERNAL/GLOBAL/LEVEL nn FROM member-name-list ;
```

LEVEL *nn* is optional. It specifies the initial level-number of the generated data descriptions. *nn* is an unsigned integer in the range 1 to 49 inclusive. The default is 1.

member-name-list is the names of one or more encoded FILEs, GROUPs, or ITEMs (or members of a user-defined member type based on a FILE, GROUP, or ITEM) from which the data description is to be generated. A maximum of sixteen names may be specified. The names must be separated by commas. You must have sufficient authority to access the members.

Each member name in *member-name-list* may be followed by this statement:

```
AS library-name
```

in which case the output generated from the member is catalogued under *library-name* in the output source library dataset. For example:

```
PRODUCE COBOL FROM FILE-EMP-TRANS, FILE-SORT-TRANS AS QLIB;
```

catalogs the data description generated from FILE-SORT-TRANS under the name QLIB in the output source library dataset. *library-name* must not be more than eight characters long (unless tailored in the installation macro). The first character must be alphabetic, @, the local currency symbol with the internal code hexadecimal 5B, %, or @.

If the AS clause is omitted, the output is catalogued under the same name as *member-name* provided that *member-name* is not too long (by default eight characters or the length tailored by the installation macro). If *member-name* is too long, no generation takes place in respect of that *member-name*.

Data names in the output generated by the command are edited, if necessary, to conform to the rules of COBOL by:

- Removing any characters that are illegal in COBOL
- Shortening any names that are longer than the maximum permitted in COBOL by removing middle characters

Validation and truncation take place after the name editing options described below have been executed.

Library-names, whether declared or defaulted, are not subjected to any name editing.

You can specify *control-options* after *member-name-list*. The *control-options* you can include in the command comprise:

1. Output control options; for example, to specify the dataset to which the output is to be written.
2. Generation control options; for example, to tailor the output generated by the command.
3. Name editing options to tailor the data names generated by the command.

These are described in the subsections that follow.

EXTERNAL is optional. It specifies that the EXTERNAL keyword is to be generated on the 01 level data description.

GLOBAL is optional. It specifies that the GLOBAL keyword is to be generated on the 01 level data description.

The EXTERNAL, GLOBAL, and LEVEL keywords are mutually exclusive.

PL/I Generation

Enter this command to generate PL/I data descriptions conforming to the storage alignment of the PL/I check-out/optimizer compilers:

```
PRODUCE PL/I state LEVEL nn FROM member-name-list ;
```

To generate PL/I data descriptions conforming to the storage alignment of the PL/I F level compiler, enter:

```
PRODUCE PL/IF state LEVEL nn FROM member-name-list ;
```

PL/I can alternatively be PLI, PL/1, or PL1.

PL/IF can alternatively be PLIF, PL/1F, or PL1F.

where:

state is optional and relates only to data descriptions produced from GROUP and ITEM members. It can be any of these:

```
EXTERNAL          STATIC          BASED name
```

where *name* is a PL/I pointer variable indicating the address on which the generated data description is based.

LEVEL *nn* is optional. When present, *nn* is an integer specifying the initial level of the generated data descriptions. *nn* is an unsigned integer in the range 1 to 99 inclusive. If LEVEL *nn* is not specified, an initial level of 1 is assumed. If LEVEL *nn* is included and *nn* is not 1, then the final symbol of each generated PL/I structure is a comma (unless tailored); otherwise it is a semi-colon.

member-name-list is the names of one or more encoded FILEs, GROUPs, or ITEMs (or members of a user-defined member type based on a FILE, GROUP, or ITEM) from which the data descriptions are to be generated. A maximum of sixteen names may be specified. The names must be separated by commas. You must have sufficient authority to access the members.

Each member name in *member-name-list* may be followed by:

AS *library-name*

in which case the output generated from the member is cataloged under *library-name* in the output source library dataset. For example:

```
PRODUCE PL/I FROM FILE-EMP-TRANS, FILE-SORT-TRANS AS QLIB;
```

catalogs the data description generated from FILE-SORT-TRANS under the name QLIB in the output source library dataset. *library-name* must not be more than eight characters long (unless tailored in the installation macro). The first character must be alphabetic, @, the local currency symbol with the internal code hexadecimal 5B, %, or @.

If the AS clause is omitted, the output is cataloged under the same name as *member-name* provided that *member-name* is not too long (by default eight characters or the length tailored by the installation macro). If *member-name* is too long, no generation takes place in respect of that *member-name*.

Data names in the output generated by the command are edited, if necessary, to conform to the rules of PL/I by:

- Removing any characters that are illegal in PL/I
- Shortening any names that are longer than the maximum permitted in PL/I by removing middle characters

Validation and truncation take place after the name editing options described below have been executed.

Library-names, whether declared or defaulted, are not subjected to any name editing.

You can specify *control-options* after *member-name-list*. The *control-options* you can include in the command comprise:

1. Output control options; for example, to specify the dataset to which the output is to be written.
2. Generation control options; for example, to tailor the output generated by the command.

3. Name editing options to tailor the data names generated by the command.

These are described in the subsections below.

Assembler Generation

Enter this command to generate Assembler data descriptions:

```
PRODUCE ASSEMBLER FROM member-name-list ;
```

ASSEMBLER can alternatively be coded as BAL, ASSEMBLY, or ALC.

member-name-list is the names of one or more encoded GROUPs or ITEMs (or members of a user defined member type based on a GROUP or ITEM) from which the data description is to be generated. A maximum of sixteen names may be specified. The names must be separated by commas. You must have sufficient authority to access the members.

Each member name in *member-name-list* may be followed by:

```
AS library-name
```

in which case the output generated from the member is cataloged under *library-name* in the output source library dataset. For example:

```
PRODUCE BAL FROM FILE-EMP-TRANS, FILE-SORT-TRANS AS QLIB;
```

catalogs the data description generated from FILE-SORT-TRANS under the name QLIB in the output source library dataset. *library-name* must not be more than eight characters long (unless tailored in the installation macro). The first character must be alphabetic, @, the local currency symbol with the internal code hexadecimal 5B, %, or @.

If the AS clause is omitted, the output is catalogued under the same name as *member-name* provided that *member-name* is not too long (by default eight characters or the length tailored by the installation macro). If *member-name* is too long, no generation takes place in respect of that *member-name*.

Data names in the output generated by the command are edited, if necessary, to conform to the rules of Assembler by:

- Removing any characters that are illegal in Assembler
- Shortening any names that are longer than the maximum specified in the DGBAL installation macro, by removing middle characters

Validation and truncation take place after the name editing options mentioned below have been executed.

Library-names, whether declared or defaulted, are not subjected to any name editing.

You can specify *control-options* after *member-name-list*. The *control-options* you can include in the command comprise:

1. Output control options; for example, to specify the dataset to which the output is to be written.
2. Generation control options; for example, to tailor the output generated by the command.
3. Name editing options to tailor the data names generated by the command.

These are described in the subsections below.

Record Layouts Generation

Enter this command to generate language-independent record layouts:

```
PRODUCE RECORD-LAYOUTS FROM member-name-list ;
```

member-name-list is the names of one or more encoded FILEs, GROUPs, or ITEMs (or members of a user defined member type based on a FILE, GROUP, or ITEM) from which the record layouts are to be generated. A maximum of sixteen names may be specified. The names must be separated by commas. You must have sufficient authority to access the members.

You can specify *control-options* after *member-name-list*. *control-options* can be *generation-control-options* and/or *name-editing-options*. (See ["Generation Control Options Overview" on page 61](#) and ["Name Editing Options Overview" on page 68](#).)

To produce record layouts with data names edited if necessary to conform to the rules of COBOL, PL/I, or Assembler, enter:

```
PRODUCE RECORD-LAYOUTS FOR language FROM member-name-list ;
```

language can be any of these:

```
COBOLCOBOL2PL/IPL/IFBAL _  
  
COBOLI IPL/1PL/1FASSEMBLER _  
  
PLI PLIFASSEMBLY  
  
PL1 PL1FALC
```

If PL/I or an alternative form is specified, the generated record layout fields will conform to the storage alignment of the PL/I checkout/optimizer compilers.

If PL/IF or an alternative form is specified, the generated record layout fields will conform to the storage alignment of the PL/I F level compiler.

You can specify *control-options* after *member-name-list*, as for language-independent record layouts generation.

To produce record layouts in addition to programming source language data descriptions, enter:

```
PRODUCE RECORD-LAYOUTS AND language FROM member-name-list ;
```

For each listed *member-name* in turn, a record layout is generated followed by the data descriptions in the specified language. Data names are edited if necessary to conform to the rules of the specified language.

member-name-list can include AS clauses to specify library names for the generated source language. See ["COBOL Generation" on page 54](#), ["PL/I Generation" on page 55](#) and ["Assembler Generation" on page 57](#) for details in the context of COBOL, PL/I or Assembler.

You can specify *control-options* after *member-name-list*. *control-options* can include output control options. (See ["Output Control Options Overview" on page 59](#).)

Output Control Options Overview

The destination of output from the PRODUCE command depends on the parameter values specified in the installation macros as amended by the output control options in the PRODUCE command itself. These output control options are available:

1. You can specify the dataset or file to which the data descriptions are to be written, and:
 - Specify a control card for the dataset in DOS environments.
 - Specify the type of dataset and a control card in OS environments.
2. You can suppress output to a source library dataset.
3. You can specify whether or not the data descriptions are to be displayed or printed.

Specifying the Output Dataset

To specify the library dataset to which the source language data descriptions are to be written, add this statement to the PRODUCE command:

```
ONTO file-name
```

For example:

```
PRODUCE COBOL FROM OFF-NO ONTO OFFDATA;
```

writes the data descriptions generated from OFF-NO to the dataset OFFDATA. If the ONTO clause is omitted, then unless generation is suppressed, output is written to the dataset GENLIB (unless tailored in the installation macro). (For the suppression of generation, see ["Suppressing Output to a Source Library Dataset" on page 61](#).)

file-name is the logical file name (ddname or dtfname) used in job control statements to indicate the external dataset name (physical file name) of the dataset to which the generated program source data descriptions are to be written. *file-name* must not be:

- MPRACWF
- MPRDIAG
- MPAID, MPAIDR, MPAIDV, or the name of any concatenated MP-AID
- The name of the repository, or the repository name with a suffix of: B, C, D, E, F, G, H, I, J, K, L, M, N, R, S, or V
- MPRPOST

Once a library dataset has been opened, then:

- If it is a partitioned dataset, it is closed at the end of the PRODUCE command processing.
- If it is a sequential dataset, it remains open until the end of the Manager Products run, unless it is explicitly closed by a CLOSE DATASET command. Thus successive PRODUCE commands may be given either to add to an already opened dataset, or to create a new one. Any number of datasets may be simultaneously open, subject to the availability of sufficient virtual storage. However, you can only process the output held in an external dataset if the dataset has been closed. The CLOSE command allows you to close individual datasets, so that they can be used for other functions, while keeping others open.

In OS environments, you can specify the type of output dataset to be either a QSAM sequential file or a BPAM partitioned dataset, by following the ONTO *file-name* clause with the keyword SEQUENTIAL or PARTITIONED respectively. The default is a BPAM partitioned dataset, unless tailored in the installation macro. In DOS environments output is always written to a sequential dataset: if PARTITIONED is stated in the command, it is converted to SEQUENTIAL.

Unless tailored or specified in the command, the standard IBM library update control card is output for sequential files:

- Under OS, the IEBUPDTE control card:

```
' ./ ADD LIST=ALL,NAME=? '
```
- Under DOS/VSE, the MAINT control card:

```
' CATALS x.?' '
```
- Under DOS/VSE/SP, the LIBR control card:

```
' CATALOG x.?' '
```

where ? indicates the point at which the generated library-name is to be inserted in the control card and x is C, P, or A for COBOL, PL/I, and Assembler respectively. The control card is written to the output dataset immediately before the generated source language data description.

You can specify control cards for alternative source library maintenance systems to be written to the output dataset. This can be done by tailoring the installation macro, or by including in the command:

```
ONTO file-name SEQUENTIAL 'control-card'
```

where *control-card* is a string of up to 72 characters, being a library system control card image. Trailing spaces are implied. A single question mark (?) character must be used to indicate the point at which the generated library member name is to be inserted in the control card.

If PARTITIONED '*control-card*' is stated in the command, '*control-card*' is ignored with no message.

Suppressing Output to a Source Library Dataset

To suppress output to a source library dataset, add the keyword NOGENERATION to the PRODUCE command. For example:

```
PRODUCE COBOL FROM OFF-NO NOGENERATION;
```

NOGENERATION can alternatively be NO-GENERATION.

Generation of source language data descriptions and/or record layouts, and their output to a terminal or a printer, is not inhibited by the NOGENERATION keyword.

It is not necessary to state NOGENERATION when record layouts are produced without source language data descriptions.

Controlling Output During Source Language Generation

To specify explicitly that the generated source language data descriptions are or are not to be displayed or printed, add PRINT or NOPRINT respectively to the command. For example:

```
PRODUCE COBOL FROM OFF-NO PRINT;
```

NOPRINT can alternatively be NO-PRINT.

In interactive mode, source language data descriptions are not displayed or printed on the terminal unless PRINT is stated in the command.

In batch mode, source language data descriptions are printed as they are produced, unless NOPRINT is stated in the command.

PRINT and NOPRINT have no effect on the printing of record layouts or of messages.

Generation Control Options Overview

The output generated by the PRODUCE command depends on the parameter values specified in the installation macros as amended by the generation control options in the PRODUCE command itself. These generation control options are available:

1. You can derive data names from aliases taken from the relevant members' ALIAS clauses.

2. You can override the parameter values in the installation macros specifying the format and contents of the output.
3. You can override the parameter values in the installation macros and suppress output that would otherwise be generated.
4. You can state which form and version of ITEM members are to be used by the PRODUCE command.

Deriving Data Names from Aliases

To derive data names wherever possible from aliases taken from the relevant members' ALIAS clauses instead of from the members' names, add to the PRODUCE command:

```
ALIAS n
```

or

```
ALIAS alias-type
```

For example:

```
PRODUCE COBOL FROM OFF-NO ALIAS 3 ;
```

ALIAS can alternatively be WITH-ALIAS.

If ALIAS *n* is specified, each generated data name is derived if possible from the *n*th general alias in the member's ALIAS clause. *n* is an unsigned integer.

If ALIAS *alias-type* is specified, each generated data name is derived if possible from the specific alias of the specified type in the member's ALIAS clause.

If a source language context keyword is present in the command, ALIAS can be specified without an associated variable. The context keyword is then applied as a default alias-type variable. For example, if:

```
PRODUCE PL/I FROM OFF-NO WITH-ALIAS ;
```

is specified, then PL/I specific aliases will be used wherever possible for the generation of data names. If ALIAS is specified without an associated variable when there is no source language context keyword in the command, a message is output and the ALIAS keyword is subsequently ignored.

If a member has no alias of the specified number or alias-type, a message is output and the data name generated in respect of that member is derived from its member name.

Note:

To ensure that specific aliases are always detected when required, the installation macro DALIAS should be tailored to declare all alternative versions of source language context keywords as sublists when specifying the alias-type keyword list. For example, all of the keywords PL/I, PLI, PL/1, PL1, PL/IF, PLIF, PL/1F, and PL1F should be declared in one sublist. See your Manager Products installation manual.)

You can also specify that data names are to be derived from local names in the containing members' KNOWN-AS clauses instead of from members' names. If the derivation of data names from local names and from aliases are both specified, then the derivation from KNOWN-AS clauses, where present, takes precedence. See "[Specifying the Format and Contents of Output](#)" on [page 63](#).

Specifying the Format and Contents of Output

To specify particular requirements for output generation, overriding the parameter values in the installation macros, add to the PRODUCE command:

GIVING *output-form-1*

For example:

```
PRODUCE ASSEMBLER FROM REC-UPD-DATA GIVING INITIAL-VALUES ;
```

output-form-1 is a list of one or more of these

NOTES	or	<i>s</i> NOTES
DESCRIPTIONS	or	<i>s</i> DESCRIPTIONS
<i>n</i> LINE-SPACING	or	BOXING
PAGE-LENGTH <i>n</i>		
FD-ONLY	or	RECORDS-ONLY or ALL-FILE
CONDITION-NAMES	or	CONDITION-NAMES FROM NOTES
	or	CONDITION-NAMES FROM DESCRIPTIONS
EQUATES		
INITIAL-VALUES		
EDIT-PATTERNS		
KNOWN-AS		
OFFSETS	or	START-POSITIONS
TERMINATOR		

If two or more options are specified, they must be separated by commas. Options must not be repeated in the clause. Not more than one from each set of alternative options—indicated by or—may be specified.

Details of the effects of the options are set out below. The third column in the table states the equivalent parameter in the relevant installation macro.

Options Available in the GIVING Clause

Option	Effect	Parameter
NOTES	Contents of NOTE clauses are listed as source language comments and/or record layout remarks.	NOTE=ALL
<i>s</i> NOTES	First <i>s</i> delimited strings in the NOTE clauses are listed.	NOTE= <i>s</i>
DESCRIPTIONS	Contents of DESCRIPTION clauses are listed as source language comments and/or record layout remarks.	DESC=ALL
<i>s</i> DESCRIPTIONS	First <i>s</i> delimited strings in the DESCRIPTION clauses are listed.	DESC= <i>s</i>
<i>n</i> LINE-SPACING	<i>n</i> blank lines are inserted between consecutive data elements in record layouts.	RECSP= <i>n</i>
BOXING	Boxing lines are inserted in record layouts. This option must not be specified if LINE-SPACING is specified.	RECBOX=YES
PAGE-LENGTH <i>n</i>	Record layouts contain <i>n</i> print lines per page.	RECPGSI= <i>n</i>
FD-ONLY	Only FDs are generated from FILE members (COBOL only). This option must not be specified if RECORDS-ONLY or ALL-FILE is specified.	GEN=FD
RECORDS-ONLY	Only record descriptions are generated from FILE members (COBOL only). This option must not be specified if FD-ONLY or ALL-FILE is specified.	GEN=01
ALL-FILE	Both FDs and record descriptions are generated from FILE members (COBOL only). This option must not be specified if FD-ONLY or RECORDS-ONLY is specified.	GEN=ALL
CONDITION-NAMES	Level-88 statements are generated from CONDITION-NAME clauses of ITEM members (COBOL only).	COND88=YES
CONDITION-NAMES FROM NOTES	Level-88 statements are generated from NOTE clauses of ITEMS (COBOL only).	DNOTE88=YES
CONDITION-NAMES FROM DESCRIPTIONS	Level-88 statements are generated from DESCRIPTION clauses of ITEMS (COBOL only).	DDESC88=YES
EQUATES	EQU statements are generated from CONDITION-NAMES clauses (with associated IS clauses) of ITEMS (Assembler only).	EQUATE=YES

Options Available in the GIVING Clause

Option	Effect	Parameter
INITIAL-VALUES	VALUE clauses are generated for COBOL elementary data descriptions or INITIAL attributes are generated for PL/I elementary data items or DC statements are generated (in place of DS statements) for Assembler data descriptions.	INITVAL=YES
EDIT-PATTERNS	Assembler edit patterns are produced.	EPPROD=YES
KNOWN-AS	Generated data names are derived wherever possible from local names in the containing members' KNOWN-AS clauses, instead of from the members' names or aliases.	KNOWNAS=YES
OFFSETS	The first two columns of generated record layouts contain decimal and hexadecimal offsets. This option must not be specified if START-POSITIONS is specified.	RECPOS=NO
START-POSITIONS	The first two columns of generated record layouts contain decimal and hexadecimal start positions. This option must not be specified if OFFSETS is specified.	RECPOS=YES
TERMINATOR	Each generated structure ends with a semicolon irrespective of whether or not a LEVEL <i>nn</i> clause has been included (PL/I only).	

Suppressing Specified Generation Options

To suppress particular options in output generation, overriding the parameter values in the installation macros, add to the PRODUCE command:

```
OMITTING output-form-2
```

output-form-2 is a list of one or more of these options:

```
NOTESEQUATES
DESCRIPTIONSINITIAL-VALUES
BOXINGEDIT-PATTERNS
CONDITION-NAMESALIAS
CONDITION-NAMES FROM NOTESKNOWN-AS
CONDITION-NAMES FROM DESCRIPTIONSTERMINATOR
```

If two or more options are specified, they must be separated by commas; for example:

```
PRODUCE ASSEMBLER FROM FILE-HIST-MASTER
OMITTING NOTES, DESCRIPTIONS;
```

Options must not be repeated in the OMITTING clause. The three CONDITION-NAMES options are mutually exclusive.

Details of the effects of the options are set out below. The third column in the table states the equivalent parameter in the relevant installation macro.

Options Available in the OMITTING Clause

Option	Effect	Parameter
NOTES	Suppresses generation of comments from NOTE clauses.	NOTE=0
DESCRIPTIONS	Suppresses generation of comments from DESCRIPTION clauses.	DESC=0
BOXING	Omits boxing lines from record layouts.	RECBOX=NO
CONDITION-NAMES	Suppresses generation of COBOL level-88 statements from CONDITION-NAME clauses of ITEMS.	COND88=NO
CONDITION-NAMES FROM NOTES	Suppresses generation of COBOL level-88 statements from NOTE clauses of ITEMS.	DNOTE88=NO
CONDITION-NAMES FROM DESCRIPTIONS	Suppresses generation of COBOL level-88 statements from DESCRIPTION clauses of ITEMS.	DDESC88=NO
EQUATES	Suppresses generation of Assembler EQU statements from CONDITION-NAME clauses of ITEMS.	EQUATE=NO
INITIAL-VALUES	Suppresses generation of VALUE clauses from COBOL elementary data descriptions or of INITIAL attributes for PL/I elementary data items or of DC statements in place of DS statements for Assembler.	INITVAL=NO
EDIT-PATTERNS	Suppresses generation of edit patterns in Assembler.	EPPROD=NO
ALIAS	Generated data names are not derived from specific aliases. Except where generation from KNOWN-AS clauses applies, generated data names are derived from member names.	ALIAS=NO
KNOWN-AS	Generated data names are not derived from local names in containing members' KNOWN-AS clauses. Except where generation from KNOWN-AS clauses applies, generated data names are derived from member names.	KNOWNAS=NO
TERMINATOR	The final line of each generated structure ends in a comma, irrespective of whether or not a LEVEL <i>nn</i> clause has been included (PL/I only).	

If, for PL/I generation, neither GIVING TERMINATOR nor OMITTING TERMINATOR is included in the command, then:

- If a LEVEL *nn* clause is included, and *nn* is not 1, the final symbol is a comma
- Otherwise, the final symbol is a semi-colon

If OMITTING and GIVING clauses are both present and contain contradictory output-form specifications (for example, ... GIVING NOTES... OMITTING NOTES...), then, of the contradictory specifications, the one from whichever of these clauses was last input prevails.

Selecting a Form or Version of an ITEM Member

To specify a particular form and version of an ITEM to be used for generation, add to the PRODUCE command:

```
USING form VERSION version
```

For example:

```
PRODUCE COBOL FROM EMP-NAME USING HELD-AS VERSION 2;
```

where:

USING may alternatively be USE.

form is one of these:

- ENTERED-AS
- HELD-AS
- REPORTED-AS
- DEFAULTTED-AS

version is an unsigned integer in the range 1 to 15. VERSION *version* can be omitted in which case version 1 is assumed. VERSION *version* is not applicable for the DEFAULTTED-AS form.

If the form and version specified are not present in the ITEM from which generation is taking place, a message is output, and the lowest-numbered version of the first form encountered in the search sequence:

```
DEFAULTTED-AS  
HELD-AS  
ENTERED-AS  
REPORTED-AS
```

is used.

If the form is not specified, but a version is specified, then if the required DEFAULTTED-AS version is not present, the search will continue through the other forms from version 1, irrespective of the specified version number.

If the USING clause is omitted, the form and version of ITEM members used are those defined for their containing GROUPs or FILEs. If the containing member does not state a version, the lowest numbered version of the relevant form is assumed. If the containing member does not state a form, DEFAULTED-AS is assumed.

To avoid excessive output of warning messages when generating from GROUP or FILE members, you are recommended to include a form keyword in those members' definitions. If there is only one form of a contained ITEM, then containing GROUPs or FILEs should be defined as also of that form.

Name Editing Options Overview

The data names output by the PRODUCE command may be generated from member names, aliases or local names, depending on parameter values specified in the installation macros, unless overridden by generation control options in the PRODUCE command itself. These data names may be edited before they are output in source language data descriptions and/or record layouts, by including editing clauses in the command. (For a description of the relevant generation control options, see ["Generation Control Options Overview" on page 61](#), ["Deriving Data Names from Aliases" on page 62](#), ["Specifying the Format and Contents of Output" on page 63](#), and ["Suppressing Specified Generation Options" on page 65](#).)

The name editing options allow you to:

1. Replace the whole of each data name, or specified strings within each data name, by a specified string.
2. Drop the whole of each data name, or specified strings within each data name.
3. Insert specified strings within each data name.
4. Specify conditions, individually for each editing clause specified within the name editing options, under which the clause is to become effective.

Any editing clause you specify will (subject to any specified conditions) apply to all data names output, but not to library names. Any number of editing clauses may be specified, in any order; but they must not be interspersed with non-editing clauses. If the generation control option USING is present in the command, then all editing clauses must immediately follow the USING clause; subject to this proviso, control options clauses may be in any order and may precede or follow the FROM clause.

For each data name, editing clauses are applied, in the sequence in which they are presented, to the data name as left by the preceding editing clause. Each editing clause is applied only once to each data name; so if you want to change more than one occurrence of a group of characters within data names, you must repeat the relevant editing clause the requisite number of times.

Names can be expanded up to a maximum of 96 characters during editing. If an editing clause would cause this length to be exceeded, editing of the name concerned is discontinued, and the name is reduced to the maximum permitted for the language being generated, by the removal of middle characters.

After all editing specified by editing clauses has been completed, or if no editing clauses are specified, the final automatic editing of data names to ensure conformity with the rules of the language being generated is performed. Names are modified if necessary by removing any illegal characters and shortening long names by removing middle characters.

Replacing Names or Name Elements

To replace each occurrence of a data name output by the PRODUCE command with a given string, add to the command:

```
REPLACING ALL WITH 'string'
```

For example:

```
PRODUCE COBOL FROM TRANS-DATA REPLACING ALL WITH 'CONV-ITEM';
```

REPLACING can alternatively be REPLACE.

To replace the first occurrence of *string-1* in each name with *string*, add to the command:

```
REPLACING 'string-1' WITH 'string'
```

string and *string-1* are strings of not more than 32 printable characters. A space (hexadecimal 40) is considered to be a printable character.

To replace the first *p* characters in each name with *string*, add to the command:

```
REPLACING (p) WITH 'string'
```

To replace *p* characters, starting at character position *m* in each name, add to the command:

```
REPLACING m (p) WITH 'string'
```

p and *m* are unsigned integers in the range 1 to 96. The sum of *p* and *m* must not exceed 97.

Dropping Names or Name Elements

To remove all the characters of each data name output by the PRODUCE command, add to the command:

```
DROPPING ALL
```

To remove the first occurrence of *string* in each name, add to the command:

```
DROPPING 'string'
```

For example:

```
PRODUCE COBOL FROM REC-EMP-TRANS DROPPING 'REC';
```

removes the first occurrence of the string REC from any data name generated by the command.

string is a string of not more than 32 printable characters. A space (hexadecimal 40) is considered to be a printable character.

To remove the first *p* characters in each name, add to the command:

```
DROPPING (p)
```

To remove *p* characters, starting at character position *m* in each name, add to the command:

```
DROPPING m (p)
```

p and *m* are unsigned integers in the range 1 to 96. The sum of *p* and *m* must not exceed 97.

Inserting Characters Into Names

To insert a given string at the start of each data name output by the PRODUCE command, add to the command:

```
INSERTING 'string' BEFORE ALL
```

For example:

```
PRODUCE COBOL FROM TRANS-DATA INSERTING 'IT-' BEFORE ALL;
```

To insert *string* before the first occurrence of *string-1* in each data name output, add to the command:

```
INSERTING 'string' BEFORE 'string-1'
```

string and *string-1* are strings of not more than 32 printable characters. A space (hexadecimal 40) is considered to be a printable character.

To insert *string* before character position *n* of each data name output, add to the command:

```
INSERTING 'string' BEFORE n
```

n is an unsigned integer in the range 1 to 96.

You can specify AFTER instead of BEFORE in each of the above formats, in which case *string* is inserted, respectively, at the end of each data name, or immediately after the first occurrence of *string-1*, or immediately after character position *n* in each data name.

Conditional Editing

Each editing clause of the name editing options operates on every data name generated, unless it is made conditional. To make an editing clause conditional, you add a subordinate WHEN clause. The WHEN clause states a condition that must be satisfied by a data name if the editing clause is to operate on that data name.

The condition stated in a WHEN clause tests each data name, or a any part of each data name, or a specified part of each data name, for equality or inequality with a specified string. For example:

```
PRODUCE COBOL FROM OFF-NO  
REPLACING 1 (3) WITH 'REC' WHEN ANY EQ 'DMR' ;
```

To limit the operation of an editing clause to those names which are the same as a given string, add:

```
WHEN ALL EQ 'string'
```

To limit the operation of an editing clause to those names that contain a given string, add:

```
WHEN ANY EQ 'string'
```

To limit the operation of an editing clause to those names containing *string* in the *p* character positions starting at character position *m*, add:

```
WHEN m (p) EQ 'string'
```

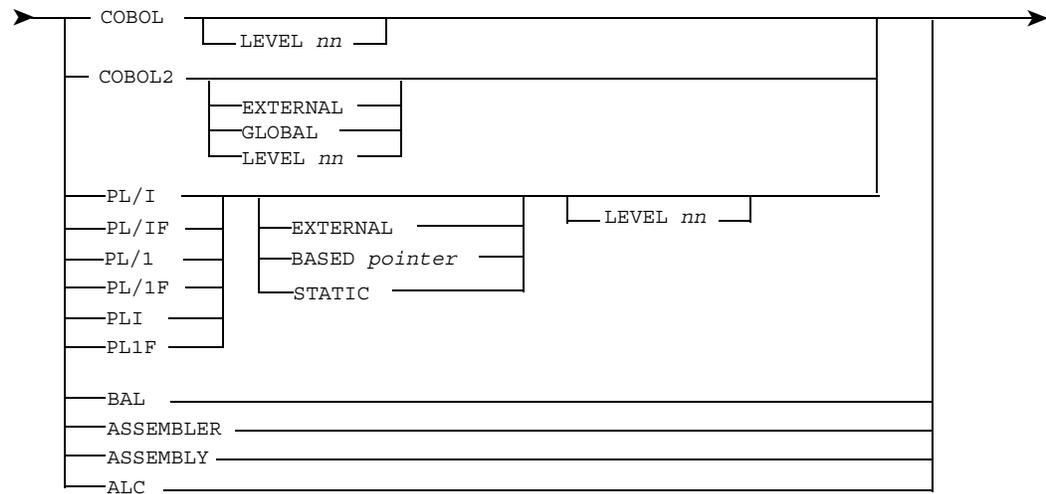
string is a string of not more than 32 printable characters. A space (hexadecimal 40) is considered to be a printable character.

p and *m* are unsigned integers in the range 1 to 96. The sum of *p* and *m* must not exceed 97.

EQ can alternatively be = .

To make the WHEN clause test for inequality, instead of equality, substitute NE (not equal) for EQ in the above specifications.

where *language* is:



nn is an assigned integer in the range 1 to 49 inclusive for COBOL, or 1 to 99 inclusive for PL/I, being the initial level of the generated data description.

pointer is a PL/I pointer available name indicating the address on which the generated data description is based.

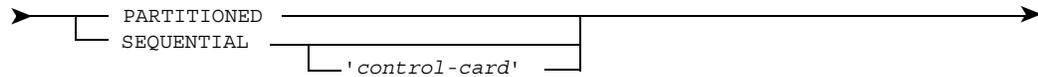
member-name is the name of an encoded repository member from which a record layout and/or a source language dataset description is to be produced. Up to a maximum of 16 member-names may be declared.

library-name is the name to be given to the generated library member in the output dataset. The name must not be more than eight characters. The first character must be alphabetic or @, £ (or a local currency symbol with the internal code hexadecimal 5B), %, or @.

file-name is the logical file name (ddname or dtfname) used in job control statements to indicate the external dataset name (physical file name) of the dataset to which the generated program source data descriptions are to be written. *file-name* must not be:

- MPRACWF
- MPRDIAG
- MPAID, MPAIDR, MPAIDV, or the name of any concatenated MP-AID
- The name of the repository, or the repository name with a suffix of: B, C, D, E, F, G, H, I, J, K, L, M, N, R, S, or V
- MPRPOST

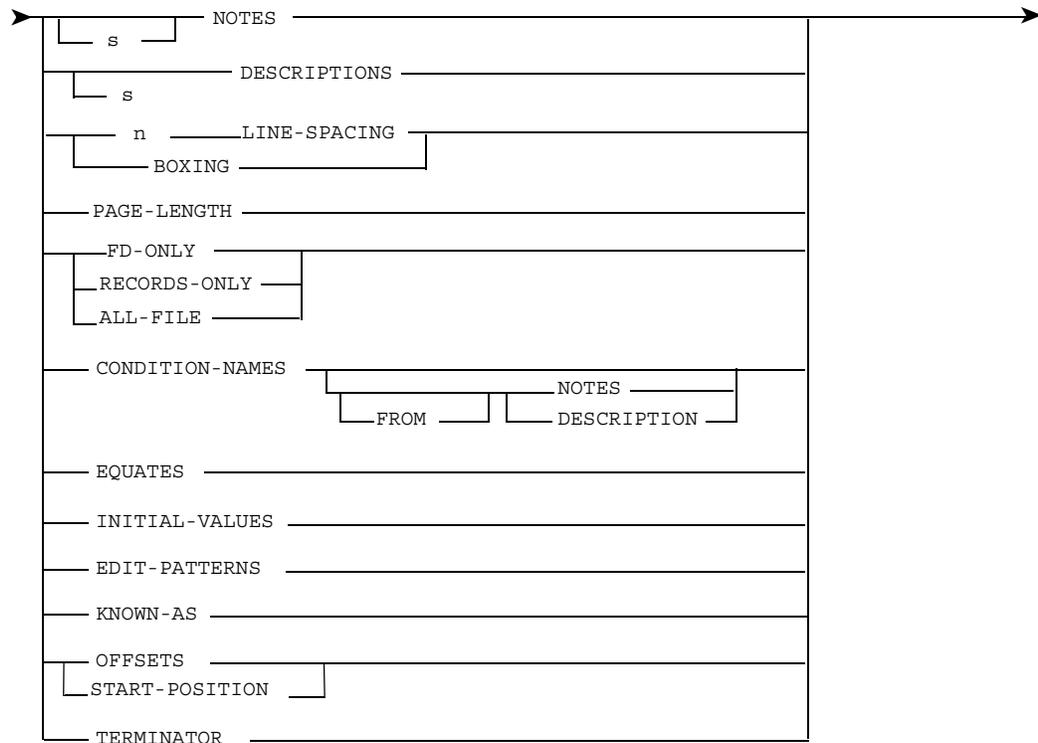
variable-a is:



where:

control-card is a character string of up to 72 characters.

output-form-1 is:

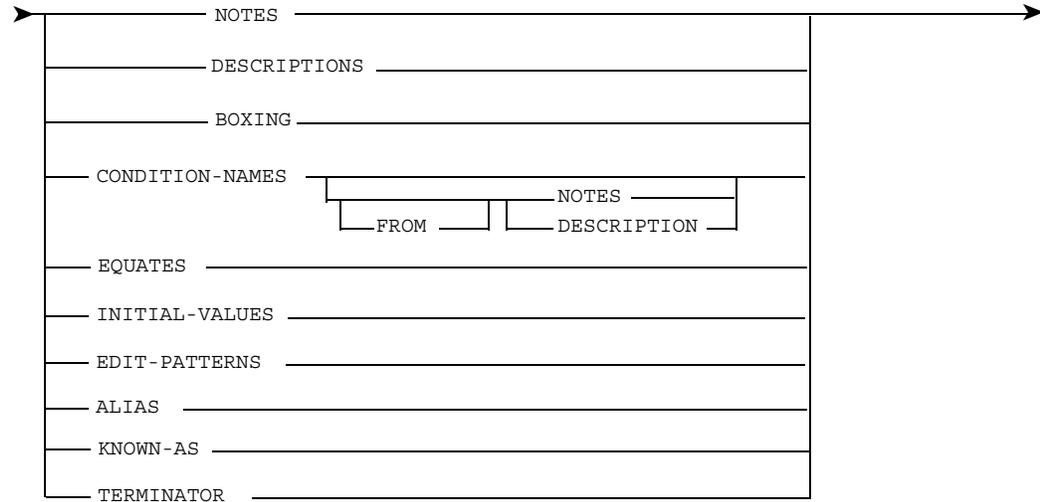


where:

s is an unsigned integer specifying a number of consecutive delimited character strings starting with the first delimited character string of the stated clause.

n is an unsigned integer specifying a number of print lines.

output-form-2 is:



where:

number is an unsigned integer identifying a general alias. The integer must be in the range 1 to g , where g is the number of *ALIAS n* keywords of the *DALIAS* macro that have been implemented with empty values. The maximum possible value of g is 16.

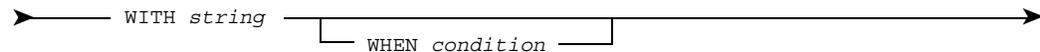
alias-type is a keyword from the alias-type keyword list of the repository.

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

m and p are unsigned integers in the range 1 to 96, specifying a generated data name, starting at character position m of the name and including p characters. If m is omitted, a value of 1 is defaulted. The sum of m and p must not exceed 97. A space or spaces must separate m and (p).

string is a character string of not more than 32 printable characters. A space (hexadecimal 40) is considered to be a printable character.

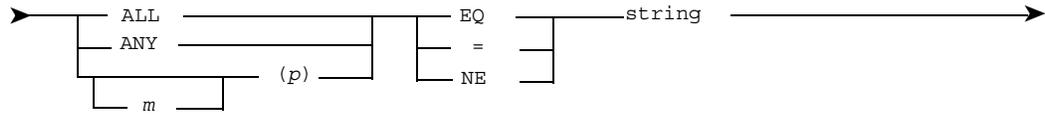
variable-b is:



where:

string is as defined above.

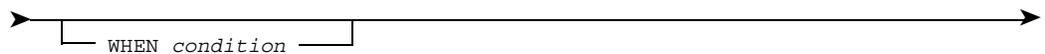
condition is:



where:

m, *p*, and *string* are as defined above.

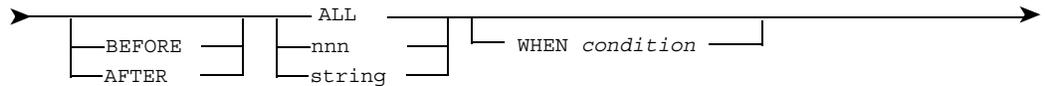
variable-c is:



where:

condition is as defined above.

variable-dis:



where:

nnn is an unsigned integer in the range 1 to 96, specifying a character position in a generated data name.

condition is as defined above.

string is as defined above.

SHOW PRODUCE-OPTIONS

Use the SHOW PRODUCE-OPTIONS command to display the currently active settings for each control option of the PRODUCE command.

The command has the form:

```
SHOW PRODUCE-OPTIONS FOR language ;
```

where *language* is any one of the programming languages, database management system languages, or file management system languages supported by Manager Products' source language generation capabilities.

For example:

```
SHOW PRODUCE-OPTIONS FOR COBOL ;
```

or

```
SHOW PRODUCE-OPTIONS FOR IMS PSBGEN ;
```

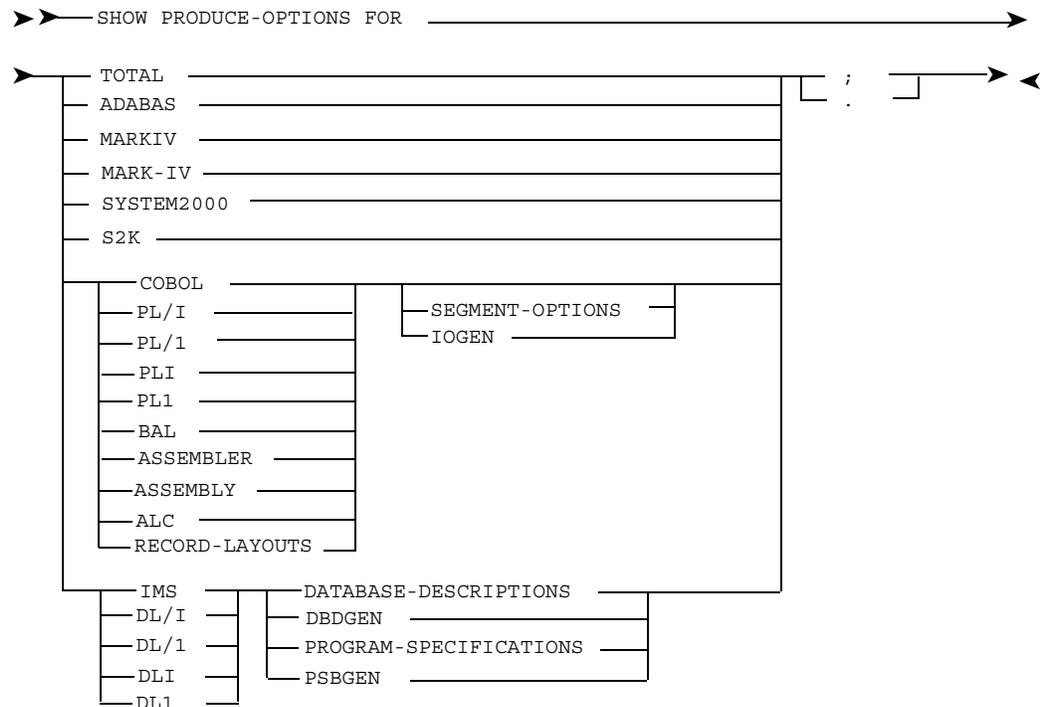
The settings displayed are those currently specified in the installation macro relevant to *language* (DGCOB, DGPLI, or DGBAL for programming languages, DGREC for language-independent record layouts, or the corresponding macros for database or file management system languages). These settings may be overridden by control options in the PRODUCE command.

If *language* is a programming language or is RECORD-LAYOUTS, it may be qualified by SEGMENT-OPTIONS or IOGEN. Either of these keywords relates to the tailoring of data descriptions output for segment input/output areas for IMS (DL/I). The settings displayed are then those currently specified in the macro DGSCOB, DGSPLI, DGSBAL, or DGSREC, depending on *language*. For example:

```
SHOW PRODUCE-OPTIONS FOR PL/I IOGEN;
```

displays the settings of the parameters in DGSPLI.

Syntax



A

- aliases 62
 - derivation of data names from 62
 - precedence of KNOWN-AS clause 63
 - tailoring of installation macro 63
- Assembler generation 31
 - DC statements 38
 - dummy names 40
 - ECU statements 37
 - edit patterns 35
 - fillers 40
 - from Arrays 33
 - from GROUPs 32
 - from ITEMs 33
 - PRODUCE command 57
- Assembler language
 - subset generated 32

B

- BAL
 - subset generated 32
- BAL generation 31
 - DC statements 38
 - dummy names 40
 - ECU statements 37
 - edit patterns 35
 - from Arrays 33
 - from GROUPs 32
 - from ITEMs 33
 - PRODUCE command 57

C

- CLOSE command 60
- COBOL generation 7
 - constructs supported 8
 - dummy names 18
 - fillers 18
 - from FILEs 9
 - from GROUPs and arrays 11
 - from ITEM's CONTENTS Clause 16
 - from ITEM's form-description 14

- from ITEM's NOTE and DESCRIPTION clauses 17
- PRODUCE command 54
- standards 8
- comments 5
- concatenated keys 13
- conditional editing 70
- contradictory output-form specifications 67
- control card 60
- control options 55
 - displaying 76
 - generation 61
 - name editing 68
 - output 59

D

- DALIAS installation macro 63
- data structures 3
- DC statements 38
- ddname 60
- DGBAL installation macro 47
- DGCOB installation macro 47
 - level number tailoring 19
- DGPLI installation macro 47
- DGREC installation macro 47
- displaying output 59
- DROPPING clause 69
- dtfname 60
- dummy data entries record 4
- dummy names
 - Assembler 40
 - COBOL 18
 - PL/I 29

E

- ECU statements 37
- edit patterns
 - Assembler 35

F

- filler name conversion 50
- fillers 4

- Assembler 40
 - COBOL 18
 - PL/I 29
 - form of ITEM 67
- G**
- generation control options 61
 - aliases 62
 - format and content of output 63
 - suppressing 65
 - GIVING clause 63
- I**
- IEBUPDTE control card 60
 - INSERTING clause 70
 - installation macros 47
 - DALIAS 75
 - keywords
 - filler name conversion 50
 - output source language 51
 - record layout 47–48
 - source language output
 - format 49
 - source library dataset control 48
 - overriding values 62
 - GIVING 63
 - OMITTING 65
 - interrogating control options settings 76
 - ITEM form and version 67
- L**
- level numbers
 - COBOL 19
 - level-88 data entries
 - generation from CONDITION-NAME
 - clauses 16
 - generation from NOTE and DESCRIPTION clauses 17
 - LIBR control card 60
 - library dataset
 - closing 60
 - specifying 59
 - suppressing output 59, 61
 - type 60
 - library update control card 60
 - logical file name 60
- M**
- MAINT control card 60
- N**
- name editing 68
 - automatic 69
 - conditional 70
 - DROPPING clause 69
 - INSERTING clause 70
 - REPLACING clause 69
 - WHEN clause 71
 - NOGENERATION keyword 61
 - NOPRINT keyword 61
- O**
- OMITTING clause 65
 - output control options 59
 - output dataset
 - closing 60
 - specifying 59
 - suppressing output 59, 61
 - type 60
 - output form specification
 - contradictory 67
 - GIVING 63
 - OMITTING 65
 - output source language tailoring 51
- P**
- PL/I generation 21
 - based structures 23
 - binary items 27
 - character set 22
 - dummy names 29
 - fillers 29
 - final symbol 67
 - from arrays 24
 - from GROUPs 22
 - from ITEMs 22
 - INITIAL attributes 28
 - level numbers 22
 - overview 22
 - pointer variables 29
 - self-defining data structures 22
 - storage attributes 22
 - pointer variables 22
 - PL/I 29
 - PRINT keyword 61
 - printing output 61
 - PRODUCE command 53
 - general form 53
 - syntax 72
 - protected members 4
- R**
- record layouts 41
 - data name 42
 - data names 42
 - definition 42

- example 42
- fields in 43
- format of 45
- generation 41
 - PRODUCE command 58
- tailoring
 - capabilities 43
 - installation macro
 - keywords 47–48
- recursive reference 4
- reference paths 4
- repetition factors
 - COBOL 15
 - PL/I 28
- REPLACING clause 69

S

- self-defining data structure 22
- SHOW PRODUCE-OPTIONS command 76
- sign symbols in PL/I pictures 26
- source language generation
 - benefits 2
 - definition 1
- source language output format tailoring 49
- source library dataset tailoring 48
- suppressing output
 - to printer or terminal 61
 - to source library dataset 61

T

- tailoring capabilities
 - installation macros 42
 - interrogation 5
 - SHOW PRODUCE-OPTIONS command 76
- overview 5

U

- USING clause 67

V

- version of ITEM 67

W

- WHEN clause 70

ASG Worldwide Headquarters Naples Florida USA | asg.com