

CA IDMS™ DBOMP

Transparency

User Guide

Release 18.5.00



This Documentation, which includes embedded help systems and electronically distributed materials, (hereinafter referred to as the "Documentation") is for your informational purposes only and is subject to change or withdrawal by CA at any time. This Documentation is proprietary information of CA and may not be copied, transferred, reproduced, disclosed, modified or duplicated, in whole or in part, without the prior written consent of CA.

If you are a licensed user of the software product(s) addressed in the Documentation, you may print or otherwise make available a reasonable number of copies of the Documentation for internal use by you and your employees in connection with that software, provided that all CA copyright notices and legends are affixed to each reproduced copy.

The right to print or otherwise make available copies of the Documentation is limited to the period during which the applicable license for such software remains in full force and effect. Should the license terminate for any reason, it is your responsibility to certify in writing to CA that all copies and partial copies of the Documentation have been returned to CA or destroyed.

TO THE EXTENT PERMITTED BY APPLICABLE LAW, CA PROVIDES THIS DOCUMENTATION "AS IS" WITHOUT WARRANTY OF ANY KIND, INCLUDING WITHOUT LIMITATION, ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NONINFRINGEMENT. IN NO EVENT WILL CA BE LIABLE TO YOU OR ANY THIRD PARTY FOR ANY LOSS OR DAMAGE, DIRECT OR INDIRECT, FROM THE USE OF THIS DOCUMENTATION, INCLUDING WITHOUT LIMITATION, LOST PROFITS, LOST INVESTMENT, BUSINESS INTERRUPTION, GOODWILL, OR LOST DATA, EVEN IF CA IS EXPRESSLY ADVISED IN ADVANCE OF THE POSSIBILITY OF SUCH LOSS OR DAMAGE.

The use of any software product referenced in the Documentation is governed by the applicable license agreement and such license agreement is not modified in any way by the terms of this notice.

The manufacturer of this Documentation is CA.

Provided with "Restricted Rights." Use, duplication or disclosure by the United States Government is subject to the restrictions set forth in FAR Sections 12.212, 52.227-14, and 52.227-19(c)(1) - (2) and DFARS Section 252.227-7014(b)(3), as applicable, or their successors.

Copyright © 2013 CA. All rights reserved. All trademarks, trade names, service marks, and logos referenced herein belong to their respective companies.

CA Technologies Product References

This document references the following CA product:

- CA IDMS/DB

Contact CA Technologies

Contact CA Support

For your convenience, CA Technologies provides one site where you can access the information that you need for your Home Office, Small Business, and Enterprise CA Technologies products. At <http://ca.com/support>, you can access the following resources:

- Online and telephone contact information for technical assistance and customer services
- Information about user communities and forums
- Product and documentation downloads
- CA Support policies and guidelines
- Other helpful resources appropriate for your product

Providing Feedback About Product Documentation

If you have comments or questions about CA Technologies product documentation, you can send a message to techpubs@ca.com.

To provide feedback about CA Technologies product documentation, complete our short customer survey which is available on the CA Support website at <http://ca.com/docs>.

Contents

Chapter 1: Introduction	9
Syntax Diagram Conventions	9
Chapter 2: Introduction to the CA IDMS DBOMP Transparency	13
Overview	13
Functions and Modules	14
Functions	14
Modules	15
Data Description Guidelines.....	17
Programming Requirements	17
Installation.....	18
Chapter 3: The Transparency Environment	21
Overview.....	21
DBOMP Macros Supported	21
Macros Supported Unconditionally.....	22
Macros That Require Program Modification and Reassembly.....	22
Macros Not Supported	23
Macros Processed Independently of the Transparency.....	24
DBOMP Process Indicators Supported	24
Process Indicators Fully Supported	24
Process Indicators Supported with Exceptions	25
Process Indicators Not Supported	25
DBOMP Routines Supported	26
CA IDMS DML Statements Supported in Bridged Programs.....	26
How to Include CA IDMS DML Statements	27
Chapter 4: Transparency Programs and Macros	29
Overview	29
IMBS Customizing Macro.....	30
Control Statement.....	31
Set Identification Statement	33
File/Record Type Description Statement	33
Pointer/Set Relationship Statement	35
Delimiter Statement	35

Output From IMBS Macro— IMBSTAB	36
IMBSTAB Error Messages.....	39
Sample IMBS and IMBSTAB	40
IMBSPROC Database Procedure.....	46
IMBSBRDG program module.....	48
Converting DBOMP Calls to CA IDMS/DB Statements	49
Converting Records Retrieved from CA IDMS/DB	51
IMBSEQ macro	55

Chapter 5: Converting DBOMP to CA IDMS/DB **59**

Overview	59
Converting Data	60
Converting DBOMP Load and Maintenance Programs	62
DBOMP Process Indicators and Corresponding DML	64
DBOMP Commands and Corresponding DML.....	69
Sequence of Logic in Converted Programs	71
Converting DBOMP Retrieval and Update Programs.....	72
DBOMP Error Codes With CA IDMS/DB Equivalentents	72

Chapter 6: Using the Transparency as a Bridge to CA IDMS/DB **75**

Overview	75
Preparing DBOMP Assembler Programs.....	75
Executing DBOMP Assembler Programs.....	76
Assembling and Executing Under z/OS	76
Assembling and Executing Under Z/VSE	79
Diagnosing Errors	81
What to Look For When Errors Occur During Program Processing.....	81
What to Look For When Inaccurate Data is Returned	83
Where to Find Values During Debugging.....	83

Appendix A: PL/I Considerations **85**

Overview	85
Transparency Support For DBOMP PL/I Commands	85
IMBSPL1 Interface Macro	87
DBOMP PL/I Program Preparation and Execution	88

Appendix B: COBOL Considerations **91**

Overview	91
Transparency Support For DBOMP COBOL Commands	91

IMBSCOBL Interface Macro	93
DBOMP COBOL Program Preparation and Execution	94

Appendix C: Sample Application and Procedures **97**

Overview	97
IMBSBILL Sample Application.....	97
IMBSMJ01 Sample JCL for z/OS	99
IMBSMJ02 Sample JCL for z/OS	100

Appendix D: Setting Up CA IDMS/DBOMP Transparency Under z/OS **101**

Overview	101
Customizing and Executing IMBSMJ01 and IMBSMJ02	102
Explanation of EXEC Statements in IMBSMJ01 Procedure	103
Customizing IMBSMJ01.....	103
IMBSMJ01 (z/OS).....	104
Explanation of EXEC Statements in IMBSMJ02 Procedure	105
Customizing IMBSMJ02.....	106
Executing IMBSMJ01 and IMBSMJ02	107

Appendix E: Setting Up CA IDMS DBOMP Transparency under Z/VSE **109**

Customizing and Executing IMBSVJ01 and IMBSVJ02	109
Explanation of EXEC Statements in IMBSVJ01 Procedure	109
Explanation of EXEC Statements in IMBSVJ02 Procedure	110
Running IMBSVJ01	111
Running IMBSVJ02	111

Index **113**

Chapter 1: Introduction

This manual is intended for:

- Database administrators who are converting DBOMP databases to CA IDMS/DB databases
- Application programmers who are using existing DBOMP application programs to access CA IDMS/DB databases

This section contains the following topics:

[Syntax Diagram Conventions](#) (see page 9)

Syntax Diagram Conventions

The syntax diagrams presented in this guide use the following notation conventions:

UPPERCASE OR SPECIAL CHARACTERS

Represents a required keyword, partial keyword, character, or symbol that must be entered completely as shown.

lowercase

Represents an optional keyword or partial keyword that, if used, must be entered completely as shown.

italicized lowercase

Represents a value that you supply.

lowercase bold

Represents a portion of the syntax shown in greater detail at the end of the syntax or elsewhere in the document.

←

Points to the default in a list of choices.

▶—————

Indicates the beginning of a complete piece of syntax.

—————▶

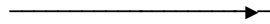
Indicates the end of a complete piece of syntax.

—————→

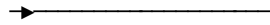
Indicates that the syntax continues on the next line.

▶—————

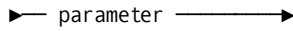
Indicates that the syntax continues on this line.



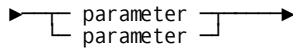
Indicates that the parameter continues on the next line.



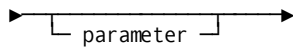
Indicates that a parameter continues on this line.



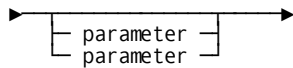
Indicates a required parameter.



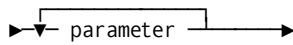
Indicates a choice of required parameters. You must select one.



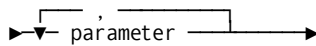
Indicates an optional parameter.



Indicates a choice of optional parameters. Select one or none.



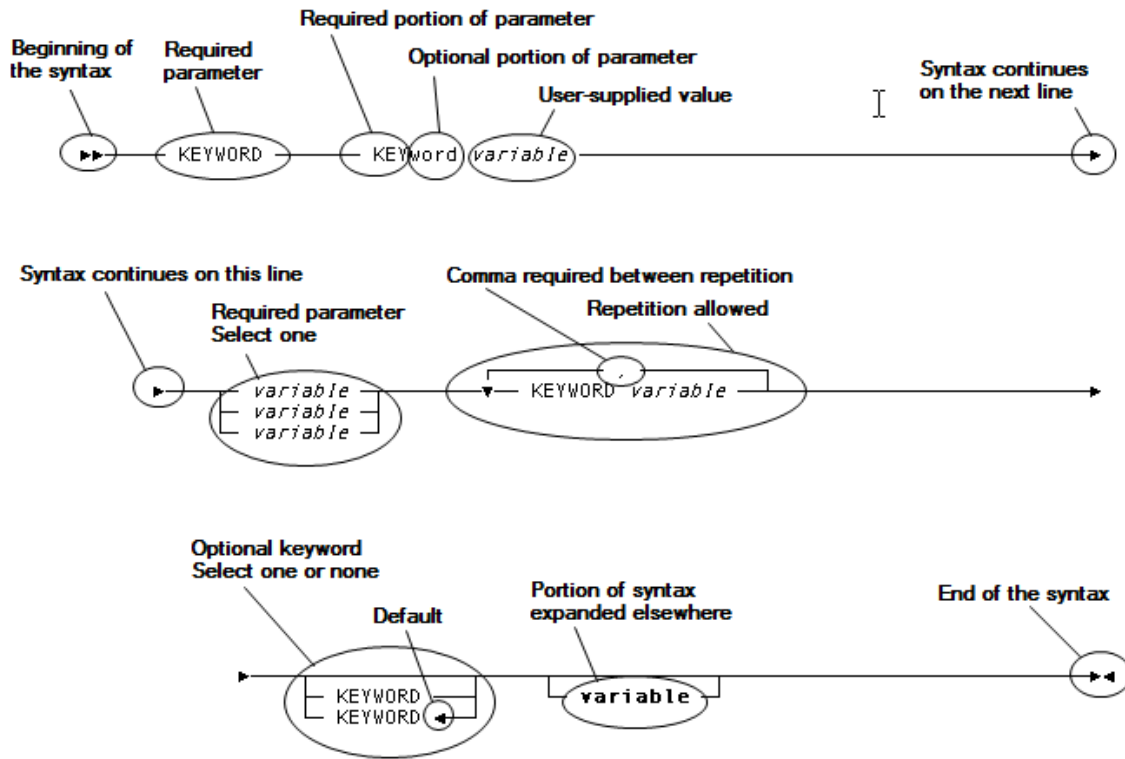
Indicates that you can repeat the parameter or specify more than one parameter.



Indicates that you must enter a comma between repetitions of the parameter.

Sample Syntax Diagram

The following sample explains how the notation conventions are used:



Chapter 2: Introduction to the CA IDMS DBOMP Transparency

This section contains the following topics:

[Overview](#) (see page 13)

[Functions and Modules](#) (see page 14)

[Data Description Guidelines](#) (see page 17)

[Programming Requirements](#) (see page 17)

[Installation](#) (see page 18)

Overview

The CA IDMS DBOMP Transparency facilitates conversion from DBOMP or its Z/OS equivalent, CFMS, to CA IDMS/DB. By simulating the DBOMP environment, the transparency allows you to run existing DBOMP application programs after the DBOMP files have been converted to CA IDMS/DB database files. This allows for a gradual conversion from DBOMP to CA IDMS/DB.

Minimal User Involvement

The CA IDMS DBOMP Transparency is usually transparent to the DBOMP retrieval and update programs that it bridges, requiring little or no program alteration and usually no reassembly.

Conversion Tool

To aid you in converting DBOMP load and maintenance programs, the transparency package includes a prototype CA IDMS/DB bill-of-materials application program. This program shows the logic required to add records to and delete records from a CA IDMS/DB database.

This program is in [Sample Application and Procedures](#) (see page 97).

System Requirements

The transparency requires no operating system facilities other than those necessary for CA IDMS/DB.

Two of the CA IDMS DBOMP Transparency modules, IMBSBRDG and IMBSTAB, require 5Kb memory in addition to that needed for standard CA IDMS/DB processing. Disk storage and all other memory requirements are the same as for CA IDMS/DB. The transparency operates under the CA IDMS/DB central version or in local mode.

The remainder of this chapter discusses the following topics:

- CA IDMS DBOMP Transparency functions and modules
- Data description guidelines
- Programming restrictions

Functions and Modules

This section describes what the CA IDMS DBOMP Transparency does and the modules it uses to do it.

Functions

The transparency acts as a bridge between the DBOMP application program and CA IDMS/DB, as follows:

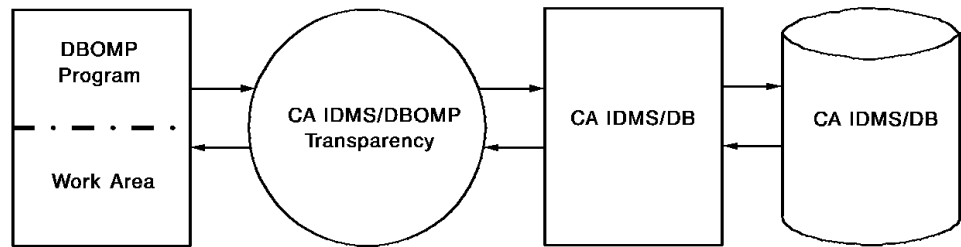
- Accepts data and processing requests from the calling program
- Converts the data to CA IDMS/DB record formats
- Converts the processing requests to CA IDMS/DB commands
- Passes the converted information to the CA IDMS/DB database management system

Conversely, the transparency also:

- Retrieves data from the CA IDMS/DB database
- Converts the data to DBOMP record formats
- Returns the converted data, along with CA IDMS/DB control information, to the calling program

All communication occurs between the DBOMP program and the transparency or between transparency and CA IDMS/DB. The transparency does not interface directly with the operating system.

The following figure illustrates the CA IDMS DBOMP Transparency processing sequence.



Modules

The two central transparency modules are IMBSBRDG and IMBSTAB:

- **IMBSBRDG**—handles all application program requests for database services
- **IMBSTAB (user-customized bridge module)**—supplies IMBSBRDG with the CA IDMS/DB and DBOMP record descriptions necessary to simulate DBOMP processing

IMBSBRDG and IMBSTAB are discussed briefly below. These and other transparency components are described in detail in [Transparency Programs and Macros](#) (see page 29).

IMBSBRDG

The IMBSBRDG module replaces the DBOMP modules:

- BM\$PIO
- AP\$SEQ

IMBSBRDG simulates DBOMP **retrieval processing** and **update processing** at the BM\$PIO and AP\$SEQ entry points, as shown in the following table.

Simulation of:	Description
Retrieval processing	<ul style="list-style-type: none"> ■ Accepts a DBOMP call to entry point BM\$PIO or AP\$SEQ ■ Validates the DBOMP file name and process indicator ■ Converts the process indicator to a CA IDMS/DB call ■ Retrieves the requested record from the CA IDMS/DB database ■ Converts the retrieved CA IDMS/DB record to a DBOMP record ■ Returns the converted record to the calling program ■ Converts the CA IDMS/DB error status to the appropriate DBOMP error code ■ Updates the work area prefix ■ Returns control to the calling program
Update processing	<ul style="list-style-type: none"> ■ Accepts a DBOMP call to entry point BM\$PIO or AP\$SEQ ■ Validates the DBOMP file name and process indicator ■ Converts the process indicator to a CA IDMS/DB call ■ Reconstructs a CA IDMS/DB record from the updated DBOMP record ■ Returns the reconstructed record to the CA IDMS/DB database ■ Converts the CA IDMS/DB error status to the appropriate DBOMP error code ■ Updates the work area prefix ■ Returns control to the calling program

IMBSTAB

The IMBSTAB customized bridge module is generated by the user-coded customizing macro, IMBS. IMBSTAB consists entirely of buffers and tables that describe the DBOMP files and their equivalent CA IDMS/DB record types and set relationships. IMBSTAB provides IMBSBRDG with the environmental information necessary to build DBOMP records to be returned to the calling program and to reconstruct CA IDMS/DB records to be returned to the CA IDMS/DB database.

Data Description Guidelines

Adhere to the data description guidelines presented below when you describe the parts of the CA IDMS/DB database that will be accessed by bridged DBOMP programs:

- Make sure there is one CA IDMS/DB record type for every DBOMP file to be simulated.
- Check the schema description of the CA IDMS/DB record types. Make sure the description allows the generation of a subschema view that represents the data exactly as it appears on the DBOMP files, with the exception of disk addresses, which are not part of the schema description.
- Define record types that are members of more than one set in the schema with next, prior, and owner pointers, so that an end-of-set condition can be detected by the transparency and communicated to the calling program.
- Store DBOMP master files as CALC or DIRECT (for sequential processing) record types on the CA IDMS/DB database.
- Store DBOMP chain files as VIA record types on the CA IDMS/DB database; however, these member records can also be described as owners of other sets.

Programming Requirements

You must do the following for any DBOMP application program you want to bridge with the transparency:

- Make all database service requests using the following Assembler macros:
 - CA\$LL
 - CHA\$E
 - GE\$T

- PUS\$T
- ST\$KY
- ST\$DA

For PL/I equivalents of these macros, see [PL/I Considerations](#) (see page 85). For COBOL equivalents of these macros, see [COBOL Considerations](#).

- Remove MF\$SQ and FI\$LE macros from the application program; replace them with the transparency macro IMBSEQ.

For more information on IMBSEQ, see [Transparency Programs and Macros](#) (see page 29).

- [COBOL Considerations](#) (see page 91) Use an index for the logical sequential ordering of master records.

Note: For more information on indexing, see the *CA IDMS Database Administration Guide*.

- Make sure that the application program does not combine DBOMP calls with CA IDMS/DB calls.

For more information on using CA IDMS/DB verbs in a bridged DBOMP program, see [How to Include CA IDMS DML Statements](#) (see page 27).

- Convert any application program that performs structural maintenance functions to CA IDMS/DB.

For more information on converting maintenance programs, see [Converting DBOMP Load and Maintenance Programs](#) (see page 62).

Installation

Use the CA IDMS installation media to install the CA IDMS DBOMP Transparency software.

Note: For more information about installation, see the *CA IDMS Installation Guide* for your operating system.

The following three tables list the object, source, and load modules placed in CA IDMS DBOMP Transparency or CA IDMS/DB libraries at the time of installation.

Object and Load Modules Placed During Installation

Items listed in the following table exist as both object and load modules.

Module	Description
IMBSBRDG	Bridge program

Module	Description
IMBSPROC	Database procedure

Source Modules Placed During Installation

Modules listed in the following table exist as source only.

Module	Description
BRDGSAMP	Z/OS JCL for BRDGSAMP procedure (for more information, see Sample Application and Procedures (see page 97))
IMBS	Customizing macro
IMBSBILL	Sample CA IDMS/DB COBOL manufacturing application program
IMBSBRDG	Assembler source code for IMBSBRDG object module
IMBSCOBL	IMBS COBOL interface macro
IMBSDBMP	Sample COBOL DBOMP program (to be bridged)
IMBSDMCL	Sample DMCL description module
IMBSINP1	Sample input to IMBSBILL
IMBSINP2	Sample input to IMBSDBMP
IMBSPL1	CA IDMS DBOMP Transparency PL/I interface macro
IMBSPROC	Source code for database procedure object module
IMBSAMP	Z/OS JCL for IMBSAMP procedure (for more information, see Sample Application and Procedures (see page 97))
IMBSCHM	Sample CA IDMS/DB schema description
IMBSUBS	Sample CA IDMS/DB subschema description
IMBSTAB	Sample input to IMBS customizing macro

Chapter 3: The Transparency Environment

This section contains the following topics:

[Overview](#) (see page 21)

[DBOMP Macros Supported](#) (see page 21)

[DBOMP Process Indicators Supported](#) (see page 24)

[DBOMP Routines Supported](#) (see page 26)

[CA IDMS DML Statements Supported in Bridged Programs](#) (see page 26)

[How to Include CA IDMS DML Statements](#) (see page 27)

Overview

CA IDMS DBOMP Transparency for DBOMP Transparency functions include:

- Simulation of the logic generated by DBOMP retrieval and update macros and process indicators
- Limited maintenance of the Run Activity Control Number (RACN)
- Support of a limited number of CA IDMS verbs issued from bridged programs

This chapter discusses support for the following entities in the transparency environment:

- DBOMP macros
- DBOMP process indicators
- Special DBOMP routines
- CA IDMS DML statements

DBOMP Macros Supported

The transparency supports, to varying degrees, DBOMP programs that issue retrieval and update macros. Support of DBOMP programs that issue macros to entry point BM\$PIO is unconditional and requires no program modification; Support of DBOMP programs that issue macros to AP\$SEQ requires that the programs be modified and reassembled. To modify these programs, you replace DBOMP macros that provide logic routines for sequential and consecutive processing with the transparency's macros.

This section describes the following categories of DBOMP Assembler macros in the transparency environment:

- Macros supported unconditionally by the transparency
- Macros requiring program modification and reassembly
- Macros not supported by the transparency
- Macros processed independently of the transparency

For more information on PL/I equivalent macros, see [PL/I Considerations](#) (see page 85). For more information on COBOL equivalent macros, see [COBOL Considerations](#) (see page 91).

Macros Supported Unconditionally

The transparency simulates unconditionally the processing generated by macros issued to entry point BM\$PIO. Programs that issue macros only to this entry point need not be altered or reassembled. The transparency interprets these macros as follows:

- **CA\$LL** (issued directly or as part of the CHA\$E macro expansion)— Establishes linkage with the transparency by passing the work area prefix to the bridge program
- **CHA\$E**— Walks a set

Macros That Require Program Modification and Reassembly

The transparency requires that programs issuing macros to entry point AP\$SEQ be altered and subsequently reassembled before interfacing with the bridge. The transparency can simulate the following macros only if you remove the prerequisite MF\$SQ and FI\$LE macros from the issuing program and replace them with the transparency macro IMBSEQ (see [Transparency Programs and Macros](#) (see page 29)):

- **GE\$T**— Sequential retrieval
- **PU\$T**— Sequential update
- **ST\$KY**— Skip-sequential retrieval using logical key
- **ST\$DA**— Skip-sequential retrieval using disk address

Transparency support of the sequential processing logic generated by the ST\$KY and ST\$DA macros assumes the use of indexing. Indexing allows the transparency to support logical sequential dependencies in DBOMP programs. If indexing hasn't been defined for the database, all programs using ST\$KY and ST\$DA must be altered to remove logical sequential dependencies before interfacing with the bridge.

The transparency handles GE\$T, PU\$T, ST\$KY, and ST\$DA as follows:

- **GE\$T**—The transparency retrieves the first record in the logical or physical sequence of the named file and returns it to the work area. Subsequent GE\$T macros issued for the same file cause the transparency to retrieve records in logical sequential order from that point if the record type is indexed, or in physical sequential order from that point if the record type is not indexed. Each retrieved record becomes current of run unit and current of its record type.
- **PU\$T**—The transparency verifies that the named record is current of the transaction, updates the record with the information in the user work area, and returns the record to the CA IDMS/DB database. If the record is not current of run unit when PU\$T is issued, CA IDMS DBOMP Transparency performs a direct read to establish currency.
- **ST\$KY**—The transparency retrieves a record by the key specified in the work area prefix for the named file and returns the record to the work area. Currency for the file (record type) is set at the retrieved record. Subsequent GE\$T macros cause the transparency to retrieve records in logical sequential order from that point if the record type is indexed, or in physical sequential order from that point if the record type is not indexed.
- **ST\$DA**—The transparency retrieves a record by the disk address specified in the work area prefix for the named file and returns the record to the work area. Currency for the file (record type) is set at the retrieved records in logical sequential order from that point if the record type is indexed, or in physical sequential order from that point if the record type is not indexed.

Macros Not Supported

The following list shows the DBOMP macros you should remove from your bridged programs and what to replace them with.

Remove this macro:	Replace it with:
MF\$SQ	IMBSEQ
FI\$LE	IMBSEQ
CF\$RT	IMBSEQ
CGE\$T	GE\$T
CPU\$T	PU\$T

Macros Processed Independently of the Transparency

The following macros are executed independently of the transparency. Do not alter them or remove them from bridged programs:

- **BM\$DS**— Generates dummy sections
- **BM\$WA**— Generates the work area prefix
- **EQ\$RG**— Equates registers to a symbol
- **MO\$VE**— Moves a variable number of bytes from one field to another
- **MSG**— Displays a message on the console
- **TY\$PE**— Displays data on the console

DBOMP Process Indicators Supported

The transparency supports most DBOMP process indicators that request retrieval and update functions. That support is achieved when the transparency does the following:

1. Accepts DBOMP process indicators that are passed in the work area prefix when a CA\$LL macro is issued.
2. Converts those process indicators to CA IDMS/DBB calls.

Note: The transparency does not support any DBOMP process indicators that request structural maintenance functions.

Process Indicators Fully Supported

The following process indicators are supported by the transparency in the same manner they are supported by DBOMP:

- **MRAN**— Reads master file record by key and return data
- **MRKY**— Reads master file record by key
(positioning only)
- **MDIR**— Reads master file record by disk address and return data
- **MRDR**— Reads master file record by disk address
(positioning only)
- **MUPD**— Updates current master file record
- **CDIR**— Reads chain file record by disk address and return data

- **CRDR**— Reads chain file record by disk address (positioning only)
- **CUPD**— Updates current chain file record

Process Indicators Supported with Exceptions

The following process indicators are supported by the transparency but are handled in a manner that is different from DBOMP:

- **CMPR**—The transparency moves the disk address from the work area prefix, simulating compression. Since CA IDMS/DBB uses only 4-byte relative addresses, actual compression is unnecessary. This operation is transparent to the calling program, and no program changes need be made.
- **EXPN**—The transparency moves the disk address to the work area prefix, simulating expansion. Since CA IDMS/DBB uses only 4-byte relative addresses, actual expansion is unnecessary. This operation is transparent to the calling program, and no program changes need be made.
- **OPEN**—The first CA\$LL issued by the DBOMP program moves an OPEN process indicator to the work area prefix of each file. The first OPEN encountered by the transparency opens the entire CA IDMS/DBB database: BINDs are issued for the run unit and all record types, and database areas are READYed. In addition, the OPEN process indicator for the first and all other files causes the transparency to determine, for future processing purposes, how the corresponding record type is stored on the CA IDMS/DB database (CALC or DIRECT for master files; VIA for chain files). OPEN also causes the transparency to determine from information in IMBSTAB whether the file named in the CA\$LL is the one for which RACN processing has been requested. If so, the transparency returns the file control record to the work area for that record (for information about the transparency's support of RACN, see [DBOMP Routines Supported](#) (see page 26).
- **CLOS**—The first CLOS encountered by the transparency closes the entire CA IDMS/DB database: the transparency updates the file control record if RACN processing has been requested for a file, and then issues a FINISH command.

Process Indicators Not Supported

The following DBOMP retrieval and update process indicators are *not* supported by the transparency. Remove them from bridged programs:

- **MWRT**
- **CWRT**
- **CCHG**
- **CCSR**

DBOMP Routines Supported

The transparency provides the logic for limited maintenance of the Run Activity Control Number (RACN). If you want to retain RACN logic in bridged programs, modify RACN processing within each program to accommodate the limited support provided by the transparency.

Note: The transparency does not acknowledge low-level code logic or chain count logic. The presence of low-level code or chain count fields in a DBOMP file does not necessitate program modification. These fields are ignored.

The transparency supports RACN logic as follows:

- RACN processing is maintained for only one DBOMP file
- OPEN processing causes the transparency to return to the calling DBOMP program the file control record for the file for which RACN has been specified
- CLOS processing causes the transparency to MODIFY the file control record, thereby returning it to the database

Once the file control record has been made available to the program, the transparency ignores it until a CLOS process indicator is issued. All RACN logic is executed independently of the transparency so the contents of the file control record can be manipulated by the executing program as you wish. When the transparency encounters a CLOS process indicator, it modifies the file control record, whether or not the DBOMP program has updated that record.

You are responsible for storing (in the CA IDMS/DB database) one occurrence of the record for which RACN processing is specified. The database key for this record must be initialized to binary zeros.

CA IDMS DML Statements Supported in Bridged Programs

The transparency supports certain CA IDMS DML statements issued from a DBOMP program. These DML statements (for Assembler) are as follows:

- @BIND PROC
- @COMMIT(ALL)
- @ROLLBAK(CONTINUE)
- @ACCEPT(STATS/PROC)

How to Include CA IDMS DML Statements

For each CA IDMS DML statement you want to include in a bridge program, do the following:

1. Build a three-field argument in the program variable storage of the bridged DBOMP program.
2. Pass the arguments to the bridge program. The transparency converts the values in the arguments to CA IDMS DML statements.

Step 1— Build the argument

Use the information in the following table to build the three-field argument for the DML statement.

Field	Usage	Length	Contents
1	Character	8	The literal value of the CA IDMS verb issued by the bridged program. Acceptable values are: <ul style="list-style-type: none"> ■ @BIND ■ @COMMIT ■ @ROLLBAK ■ @ACCEPT PL/I or COBOL equivalents are also acceptable.
2	Character	8	The literal value of the CA IDMS keyword associated with the CA IDMS verb entered in field 1. Acceptable values are as shown in the list following this table.
3	Character	1-256	The variable data passed by : <ul style="list-style-type: none"> ■ @BIND PROC ■ @ACCEPT PROC or <ul style="list-style-type: none"> ■ @ACCEPT STATS This field is necessary only if one of these DML statements is issued.

The acceptable values for field 2 (shown in the preceding table) are:

- Name of the database procedure, if @BINDing to or @ACCEPTing from a data procedure
- STATS, if @ACCEPTing database statistics
- ALL, if issuing the @COMMIT verb and releasing locks on current records; enter spaces if issuing an unqualified @COMMIT verb
- CONTINUE, if issuing the @ROLLBAK verb and terminating the run unit; enter spaces if issuing an unqualified @ROLLBAK verb

In the following example, the bridged DBOMP Assembler program builds the argument IDMSREQ to issue the CA IDMS DML statement @ACCEPT STATS:

```
IDMSREQ      DS      0D
IDMSVERB     DC      CL8 '@ACCEPT '
IDMSKEY      DC      CL8 'STATS  '
IDMSAREA     DS      CL256
```

Step 2— Pass the argument to the bridge program

Bridged DBOMP Assembler program

Include this statement in a bridged DBOMP Assembler program to pass the CA IDMS DML statement argument to the bridge program:

```
CA$LL BMP$I0, argument-name
```

Bridged DBOMP PL/I program

Include this statement in a bridged DBOMP PL/I program to pass the CA IDMS DML statement argument to the bridge program:

```
CALL CA$LL(argument_name, 'END.')
```

Bridged DBOMP COBOL program

Include this statement in a bridged DBOMP COBOL program to pass the CA IDMS DML statement argument to the bridge program:

```
CALL BMPCALL USING argument-name.
```

Chapter 4: Transparency Programs and Macros

This section contains the following topics:

- [Overview](#) (see page 29)
- [IMBS Customizing Macro](#) (see page 30)
- [Output From IMBS Macro— IMBSTAB](#) (see page 36)
- [Sample IMBS and IMBSTAB](#) (see page 40)
- [IMBSPROC Database Procedure](#) (see page 46)
- [IMBSBRDG program module](#) (see page 48)
- [IMBSEQ macro](#) (see page 55)

Overview

This chapter provides information on the transparency components that are described briefly in the following table.

Component	Brief description
IMBS customizing macro	Describes the DBOMP files and the equivalent CA IDMS/DB database. The IMBS macro generates IMBSTAB.
IMBSTAB	Contains (in tabular format) the data that the bridge program uses to convert CA IDMS/DB records to DBOMP records.
IMBSPROC database procedure	Moves pointers from the subschema table into a CA IDMS/DB dummy record.
IMBSBRDG program module	Simulates DBOMP records and processing using IMBSTAB, IMBSPROC, IMBSEQ (or equivalent COBOL or PL/I macros), and CA IDMS/DB.
IMBSEQ macro	Supports the DBOMP GE\$T, PU\$T, ST\$KY, and ST\$DA macros in Assembler programs and replaces the MF\$SQ, FI\$LE, and CF\$RT macros. For more information on equivalent PL/I and COBOL macros, see PL/I Considerations (see page 85) and see COBOL Considerations (see page 91).

IMBS Customizing Macro

IMBS is an Assembler macro that describes DBOMP files and the CA IDMS/DB database that replaces them.

Input statements for IMBS are as follows:

- Control
- Set identification
- File/record type description
- Pointer/set relationship
- Delimiter

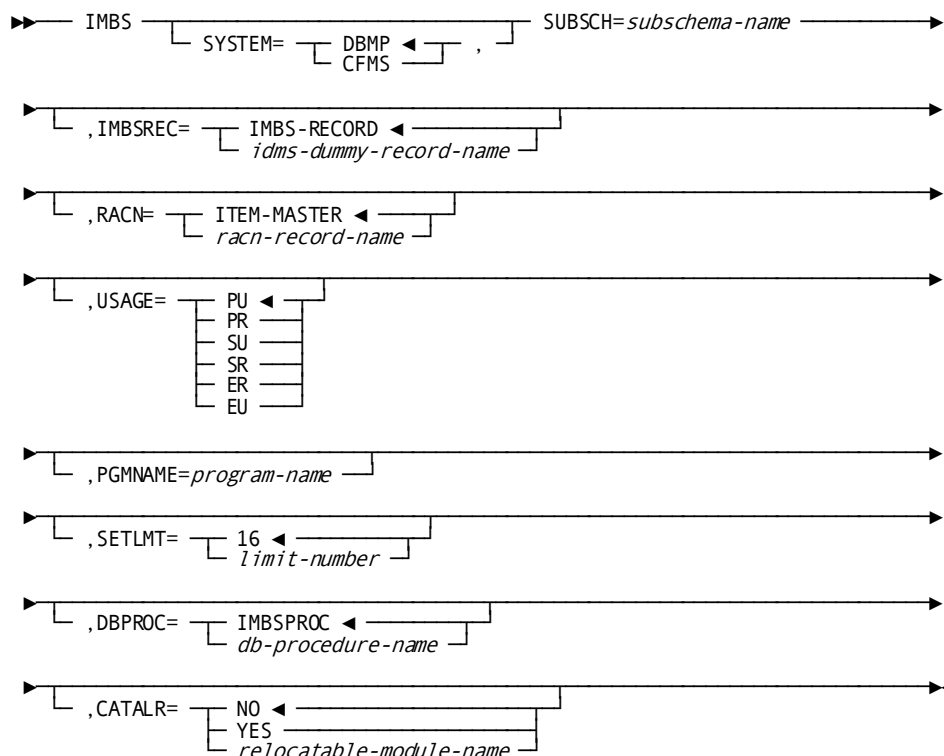
These statements require set names, file names, record types, logical record length, and pointer displacement in DBOMP records. To get this information, use the IDMSRPTS utility (see the *CA IDMS Utilities Guide*), running these reports:

Report name	Gives information on:
RECDES	Record types defined in a schema
SETDES	Sets defined in a schema
DATDIR	Record types copied into a subschema (general)
SUBREC	Record types copied into a subschema (comprehensive)
SUBAREA	Areas copied into a subschema
SUBSET	Sets copied into a subschema

Syntax for the input statements is provided in the following sections.

Control Statement

The control statement specifies control information for the run, including usage mode and required names.



IMBS

Constant; Code anywhere after column one.

SYSTEM=DBMP/CFMS

Specifies DBMP or CFMS, as appropriate. The default value is DBMP.

SUBSCH=*subschema-name*

Specifies the subschema name as it is known to CA IDMS/DB.

IMBSREC=IMBS-RECORD/*idms-dummy-record-name*

Specifies the name of the CA IDMS/DB dummy record as defined in the schema. The default value is IMBS-RECORD.

RACN=ITEM-MASTER/*racn-record-name*

Specifies the name of the record for which RACN processing is requested. The default value is ITEM-MASTER.

USAGE=

Specifies the CA IDMS/DB usage mode in which all areas named in the subschema are to be READYed.

PU

Protected update (the default)

PR

Protected retrieval

SU

Shared update

SR

Shared retrieval

ER

Exclusive retrieval

EU

Exclusive update

PGMNAME=*program-name*

Specifies the name of the program to be bridged. This parameter defaults to IDMSDBMP if DBMP is indicated in the SYSTEM= parameter, or to IDMSCFMS if CFMS is indicated in the SYSTEM= parameter.

SETLMT=*limit-number*

Sets the maximum number of sets that can be defined in a single IMBSTAB. The default is 16. The largest allowed number is 255.

DBPROC=IMBSPROC/*db-procedure-name*

Specifies the name of a database procedure that passes pointers from the subschema table to the CA IDMS/DB dummy record. The default value, IMBSPROC, should be used unless a database procedure by that name already exists.

CATALR=

Specifies the CATALR option (Z/VSE only).

NO

Specifies that a CATALR card is not to be provided at the front of the object deck. NO is the default.

YES

Specifies that a CATALR card is to be provided at the front of the object deck, naming IMBSTAB as the relocatable module.

relocatable-module-name

Specifies the relocatable module to be named on the CATALR card placed at the front of the object deck.

Set Identification Statement

The set identification statement names a CA IDMS/DB set. One set identification statement must exist for each set type to be accessed by the bridged program.

▶— IMBS SET=(*set-number*, *set-name*) —▶

IMBS

Constant; Code anywhere after column one.

set-number

Specifies a 2-digit number indicating the set number. Set identification statements must be entered in sequence by this number.

Set-number cannot exceed the value of the SETLMT parameter in the control statement.

set-name

Specifies the name of the set as it appears in the subschema.

File/Record Type Description Statement

The file/record type description statement describes the characteristics of the DBOMP file and names the CA IDMS/DB record type to which it corresponds. There must be one file/record type description statement for each DBOMP file referenced by the bridged program.

This statement must be followed by a pointer/set relationship statement for each pointer that is established for the record type and that is to be passed to the calling program by the database procedure.

```

▶— IMBS RECNAME=(dbomp-file-name, idms-record-type-name) —————▶
▶— ,TYPE= 

|   |
|---|
| M |
| C |
| S |

 ,KEYL=key-length —————▶
▶— ,LRECL=record-length —————▶
    
```

IMBS

Constant; Code anywhere after column one.

dbomp-file-name

Specifies the 7-character name of the DBOMP file.

idms-record-type-name

Specifies the name of the corresponding CA IDMS/DB record type as it appears in the subschema.

TYPE=

Specifies the type of DBOMP file.

M

Master file

C

Chain file linked to more than one master file; note that if C is specified, the corresponding record type must have next, prior, and owner pointers.

S

Chain file linked to only one master file; any file/record type description statement specifying TYPE=S must be preceded by a file/record type description statement for the master file to which it is linked.

KEYL=*key-length*

Specifies the length of the record key as it is specified in the work area prefix of the DBOMP file. *Key-length* must be between 0 and 256; specify 0 for all chain files except those with product-structure characteristics where the master-record key length is used.

LRECL=*record-length*

Specifies the length, in bytes, of the record as it appears on the DBOMP record layout. The length of the work area prefix should *not* be included in this value.

Pointer/Set Relationship Statement

Pointer/set relationship statements provide CA IDMS DML with information about the pointers established for each record type that is to be passed from the database to the user work area. One pointer/set relationship statement must exist for each pointer that is to be passed for the record type described in the preceding file/record type description statement.

▶▶— IMBS POINTER=(*pointer-number*,*pointer-type*,*pointer-displacement-number*) ▶▶

IMBS

Constant; Code anywhere after column one.

pointer-number

Specifies the two-digit number corresponding to the sequential number in the set identification statement (see above) for the set to which the pointer links the record.

pointer-type

Specifies the type of pointer, as follows:

- **N**— Next pointer
- **P**— Prior pointer
- **O**— Owner pointer
- **X**— Dummy pointer; causes the constant END to be moved to the specified pointer position in the simulated DBOMP record

pointer-displacement-number

Specifies the displacement of the pointer in the DBOMP logical record, where the record begins at byte 1.

Delimiter Statement

The delimiter statement indicates the end of the input statement entries. Code the constant IMBS anywhere after column one.

▶▶— IMBS END —————▶▶

Output From IMBS Macro— IMBSTAB

IMBSTAB is an Assembler program module generated by the IMBS macro. It consists of storage (DS) and storage constants (DC), in the form of tables and buffers. IMBSTAB:

- Supplies IMBSPROC with information needed to move pointers for current records from the CA IDMS/DB subschema table into the dummy CA IDMS/DB record
- Provides IMBSBRDG with information needed to build DBOMP records from retrieved CA IDMS/DB records
- Supplies IMBSBRDG with the information needed to return updated records from the user work area to the CA IDMS/DB database

The IMBSTAB module contains the following four tables:

Table	Contains:
Control table	Control information
Set table	An entry for each set described to the IMBS macro
Pointer table	Pointers for each set described to the IMBS macro; the groups of pointers are in the same order as the corresponding sets in the set table.
File table	A group of entries for each file described to the IMBS macro

The control table, set table, pointer table, and file table layouts are shown in the figures on the following pages.

Control Table

Displacement	Field Contents	Field Length
0	System name	4
4	Addresses of other tables and logical record buffer	16
20	Advantage CA-IDMS/DB dummy record name	16
36	Database procedure name	8
44	Subschema name	8
52	Program name	8
60	RACN record name	16
76	Usage mode	4
80	Advantage CA-IDMS/DB communications block - SSCTRL	

Set Table

The set table contains one entry for each set described to the IMBS macro.

Displacement	Field Contents	Field Length
0	<u>set-name-1</u>	16
16	<u>set-name-2</u>	16
	• • •	
	<u>set-name-n</u>	16

Pointer Table

The pointer table contains one group of pointers (owner, prior, current, and next) for each set described to the IMBS macro, in the same order as the sets to which they correspond are named in the set table.

Displacement	Field Contents	Field Length
0	<u>set-name-1</u> pointers	16
0	owner pointer	4
4	prior pointer	4
8	current pointer	4
12	next pointer	4
16	<u>set-name-2</u> pointers	16
16	owner pointer	4
20	prior pointer	4
24	current pointer	4
28	next pointer	4
	• • •	
	<u>set-name-n</u> pointers	16
	owner pointer	4
	prior pointer	4
	current pointer	4
	next pointer	4

File Table

The file table contains one group of entries for each DBOMP file and corresponding CA IDMS/DB record type described to the IMBS macro.

Displacement	Field Contents		Field Length
0	next-entry PTR		4
4	DBOMP file name		7
11	File type		1
12	Advantage CA-IDMS/DB record type name		16
28		+ Switch	1
29	Key length		1
30	Record length		2
32	Filler		4
36	Current of record type		4
40	Pointer		8
40	Address of entry in dummy record		4
44		Disp. in DBOMP record	2
46		PTR type	1
47		+ Filler	1
	<ul style="list-style-type: none"> • 1 pointer entry for each • pointer established for • the Advantage CA-IDMS/DB record type 		
	end-of-pointer entries indicator PTR type = FF (HEX)		8

The IMBS macro generates a CA IDMS/DB logical record buffer from which the bridge program constructs the DBOMP logical record. The size of this buffer is equivalent to the size of the largest CA IDMS/DB record described in the file table.

Assembling and Linking IMBSTAB

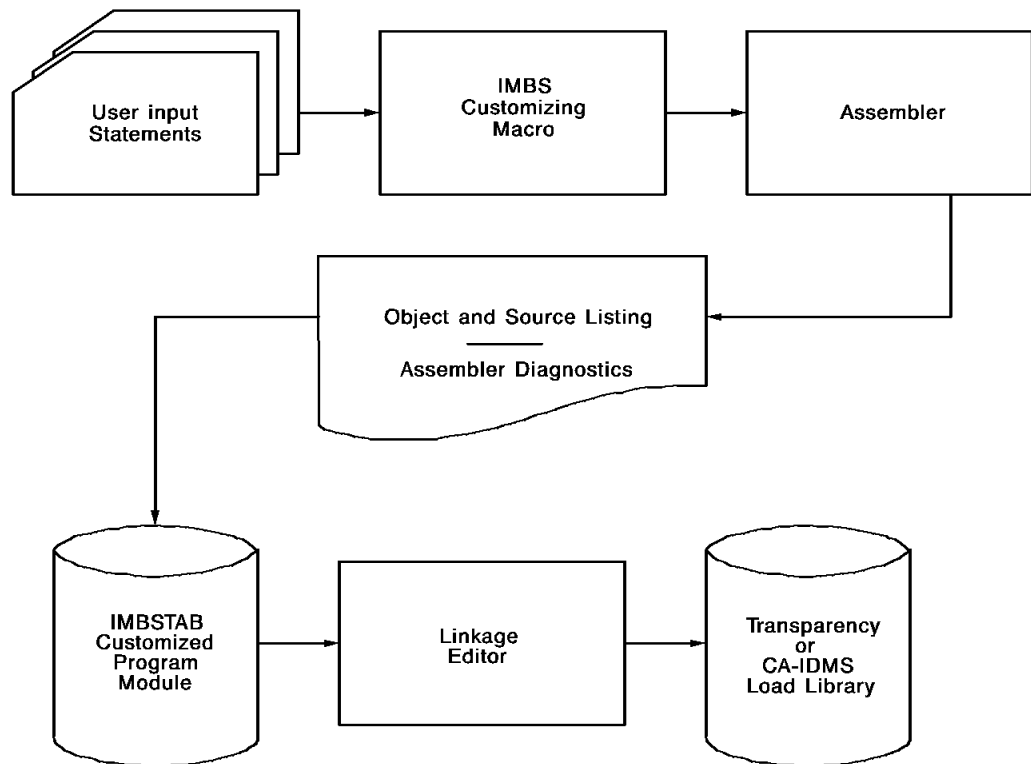
You can reassemble IMBSTAB as often as you like. This allows you to change control information and accommodate the requirements of multiple DBOMP applications. The information most likely to vary is the program name, the usage mode, the name of the record for which RACN is to be maintained, and the CATALR option (Z/VSE only).

Each time you change any input statements, do the following:

1. Submit all of the IMBS input statements.
2. Link edit IMBSTAB to the library containing IMBSBRDG.

For the JCL you use to assemble and link edit the IMBSTAB module, see [Using the Transparency as a Bridge to CA IDMS/DB](#) (see page 75).

The following flowchart illustrates IMBSTAB assembly and linkage.



IMBSTAB Error Messages

Error messages that are issued during the assembly of the IMBSTAB customized bridge program are called MNOTES. An MNOTE appears in the source code listing directly below the input statement to which it applies.

Note: The line number of an MNOTE appears on the Assembler Diagnostics and Statistics page of the Assembler output listing.

MNOTES (and their descriptions) are as follows:

- **INCORRECT USAGE MODE SPECIFIED**

There is an invalid usage mode in the USAGE= parameter of the control statement.

- **SET SPECIFIED OUT OF SEQUENCE**

A set identification statement is not in numeric sequence by the set number parameter.

- **SUBSCHEMA NOT SPECIFIED**

The SUBSCH= parameter is missing from the control statement.

- **SET TABLE LIMIT EXCEEDED**

The number of sets defined in the IMBS macro has been exceeded.

- **UNRECOGNIZED KEYWORD PARAMETER**

The Assembly program has encountered an unrecognizable keyword parameter.

You must correct input statements that are flagged by MNOTES, then resubmit the statements to the IMBS macro for assembly of IMBSTAB. Repeat the process until all user input statements are free of errors.

The error-detection capabilities of the IMBS macro are limited, and it is recommended that you check all input statements for errors not covered by MNOTES. In particular, check:

- The subschema name
- File and record type names
- File types
- Linkage options
- Pointer displacement
- CA IDMS/DB set names

If errors exist in the above values and are not detected when you generate and assemble IMBSTAB, the bridge program will encounter discrepancies between information requested by the calling program and information supplied by IMBSTAB. The results are unpredictable.

Sample IMBS and IMBSTAB

Sample Input to IMBS

The following is a sample of statements input to the IMBS macro.

```
IMBS  SYSTEM=DBMP, SUBSCH=IMBSSUBS

IMBS  SET=(01, ITEM-STRUCTURE)
IMBS  SET=(02, ITEM-WHERE-USED)
IMBS  SET=(03, WORK-ROUTING)
IMBS  SET=(04, ITEM-ROUTING)
```



```

IMBS  RECNAME=(ITEMFLE,ITEM-MASTER),TYPE=M,KEYL=5,LRECL=68
IMBS  POINTER=(01,X,1)
IMBS  POINTER=(01,N,10)
IMBS  POINTER=(02,N,14)
IMBS  POINTER=(04,N,18)
IMBS  POINTER=(04,P,22)

IMBS  RECNAME=(PRODSTR,PROD-STRUCTURE),TYPE=C,KEYL=5,LRECL=36
IMBS  POINTER=(01,0,1)
IMBS  POINTER=(01,N,5)
IMBS  POINTER=(02,0,9)
IMBS  POINTER=(02,N,13)
IMBS  POINTER=(02,P,17)
IMBS  RECNAME=(WORKCTR,WORK-CENTER),TYPE=M,KEYL=5,LRECL=32
IMBS  POINTER=(01,X,1)
IMBS  POINTER=(03,N,10)
IMBS  POINTER=(03,P,14)

IMBS  RECNAME=(ROUTING,ROUTINGS),TYPE=C,KEYL=0,LRECL=84
IMBS  POINTER=(04,0,1)
IMBS  POINTER=(04,N,5)
IMBS  POINTER=(03,0,9)
IMBS  POINTER=(03,N,13)
IMBS  POINTER=(03,P,17)

IMBS  END
END

```

Sample Output from IMBS

The following is a sample IMBSTAB source listing, the output from the IMBS macro.

LOC	OBJECT CODE	ADDR1	ADDR2	STMT	SOURCE	STATEMENT
				1	IMBS	SYSTEM=DBMP,SUBSCH=IMBSSU
000000				2+	IMBSTAB	CSECT
000000	47F0 E000		00000	3+	BC	15,0(,14)
000004	00000020			4+	DC	A(IMBSCNTL)
000008	5C5CC9D4C2E240E3			5+	DC	C'**IMBS TABLE V12.0**'
				6+*		
000020				7+	IMBSCNTL	DS 0D
000020	C4C2D4D7			8+	DC	CL4'DBMP'
000024	00000298			9+	DC	A(R1)
000028	000003F0			10+	DC	A(BUFFER)

00002C 00000148	11+	DC	A(SETABLE)
000030 00000190	12+	DC	A(PTRTAB)
000034 C9D4C2E260D9C5C3	13+	DC	CL16' IMBS-RECORD '
000044 C9D4C2E2D7D9D6C3	14+	DC	CL8' IMBSPROC '
00004C C9D4C2E2E2E4C2E2	15+	DC	CL8' IMBSSUBS '
000054 C9C4D4E2C4C2D4D7	16+	DC	CL8' IDMSDBMP '
00005C C9E3C5D460D4C1E2	17+	DC	CL16' ITEM-MASTER '
00006C 000000F5	18+	DC	A(SSCIDBCM+38-1)
000070	19+	DS	0D
000070	20+SSCTRL	DS	0CL200
000070 4040404040404040	21+PGMNAME	DC	CL8' '
000078 F1F4F0F0	22+ERRSTAT	DC	C'1400'
00007C 00000000	23+DBKEY	DC	F'0'
000080 4040404040404040	24+RECNAME	DC	CL16' '
000090 4040404040404040	25+AREANAME	DC	CL16' '
0000A0 4040404040404040	26+ERRORSET	DC	CL16' '
0000B0 4040404040404040	27+ERRORREC	DC	CL16' '
0000C0 4040404040404040	28+ERRAREA	DC	CL16' '
0000D0	29+SSCIDBCM	DS	0F
0000D0 0000000000000000	30+IDBMSCOM	DC	25F'0'
000134 00000000	31+DIRDBKEY	DC	F'0'
000138	32+DBSTATUS	DS	0CL7
000138 4040	33+DBSTMTCD	DC	CL2' '
00013A 40404040404040	34+DBSTATCD	DC	CL5' ',CL1' '
000140 00000000	35+RECOCCUR	DC	F'0'
000144 00000000	36+DMLSEQ	DC	F'0'
000148	37+SETABLE	DS	0D
	38 *		
	39	IMBS	SET=(01, ITEM-STRUCTURE)
000148 C9E3C5D460E2E3D9	40+SET1	DC	CL16' ITEM-STRUCTURE '
	41	IMBS	SET=(02, ITEM-WHERE-USED)
000158 C9E3C5D460E6C8C5	42+SET2	DC	CL16' ITEM-WHERE-USED '
	43	IMBS	SET=(03, WORK-ROUTING)
000168 E6D6D9D260D9D6E4	44+SET3	DC	CL16' WORK-ROUTING '
	45	IMBS	SET=(04, ITEM-ROUTING)
000178 C9E3C5D460D9D6E4	46+SET4	DC	CL16' ITEM-ROUTING '
	47 *		
	48	IMBS	RECNAME=(ITEMFLE, ITEM-MAS

000188	FFFFFFFF	49+	DC	F' -1'
		50+*		
000190		51+PTRTAB	DS	0D
000190	0000000000000000	52+	DC	16XL16'FF'
		53+*		
000290		54+FTABLE	DS	0D
000290	FFFFFFFF	55+	DC	F' -1'
000294	0000	56+	DC	H'0'
000296	FFFF	57+	DC	H' -1'
000298		58+R1	DS	0F
000298	000002F0	59+	DC	A(R2)
00029C	C9E3C5D4C6D3C5	60+	DC	CL7'ITEMFLE'
0002A3	D4	61+	DC	C'M'
0002A4	C9E3C5D460D4C1E2	62+	DC	CL16'ITEM-MASTER'
0002B4	0005	63+	DC	H'5'
0002B6	0044	64+	DC	H'68'
0002B8	0000000000000000	65+	DC	2F'0'
		66	IMBS	POINTER=(01,X,1)
0002C0	00000190	67+	DC	A(PRTAB+16*(01-1))
0002C4	0000	68+	DC	AL2(1-1)
0002C6	E7	69+	DC	CL1'X'
0002C7	40	70+	DC	CL1' '
		71	IMBS	POINTER=(01,N,10)
0002C8	00000190	72+	DC	A(PRTAB+16*(01-1))
0002CC	0009	73+	DC	AL2(10-1)
0002CE	D5	74+	DC	CL1'N'
0002CF	40	75+	DC	CL1' '
		76	IMBS	POINTER=(02,N,14)
0002D0	000001A0	77+	DC	A(PRTAB+16*(02-1))
0002D4	000D	78+	DC	AL2(14-1)
0002D6	D5	79+	DC	CL1'N'
0002D7	40	80+	DC	CL1' '
		81	IMBS	POINTER=(04,N,18)
0002D8	000001C0	82+	DC	A(PRTAB+16*(04-1))
0002DC	0011	83+	DC	AL2(18-1)
0002DE	D5	84+	DC	CL1'N'
0002DF	40	85+	DC	CL1' '
		86	IMBS	POINTER=(04,P,22)
0002E0	000001C0	87+	DC	A(PRTAB+16*(04-1))
0002E4	0015	88+	DC	AL2(22-1)
0002E6	D7	89+	DC	CL1'P'
0002E7	40	90+	DC	CL1' '
		91 *		
		92	IMBS	RECNAME=(PRODSTR,PROD-STR)
0002E8	FFFFFFFF	93+	DC	F' -1'
0002EC	0044	94+	DC	H'68'

0002EE FFFF	95+	DC	H' -1'
0002F0	96+R2	DS	0F
0002F0 00000348	97+	DC	A(R3)
0002F4 D7D9D6C4E2E3D9	98+	DC	CL7'PRODSTR'
0002FB C3	99+	DC	C'C'
0002FC D7D9D6C460E2E3D9	100+	DC	CL16' PROD-STRUCTURE'
00030C 0005	101+	DC	H'5'
00030E 0024	102+	DC	H'36'
000310 0000000000000000	103+	DC	2F'0'
	104	IMBS	POINTER=(01,0,1)
000318 00000190	105+	DC	A(PTRTAB+16*(01-1))
00031C 0000	106+	DC	AL2(1-1)
00031E D6	107+	DC	CL1'0'
00031F 40	108+	DC	CL1' '
	109	IMBS	POINTER=(01,N,5)
000320 00000190	110+	DC	A(PTRTAB+16*(01-1))
000324 0004	111+	DC	AL2(5-1)
000326 D5	112+	DC	CL1'N'
000327 40	113+	DC	CL1' '
	114	IMBS	POINTER=(02,0,9)
000328 000001A0	115+	DC	A(PTRTAB+16*(02-1))
00032C 0008	116+	DC	AL2(9-1)
00032E D6	117+	DC	CL1'0'
00032F 40	118+	DC	CL1' '
	119	IMBS	POINTER=(02,N,13)
000330 000001A0	120+	DC	A(PTRTAB+16*(02-1))
000334 000C	121+	DC	AL2(13-1)
000336 D5	122+	DC	CL1'N'
000337 40	123+	DC	CL1' '
	124	IMBS	POINTER=(02,P,17)
000338 000001A0	125+	DC	A(PTRTAB+16*(02-1))
00033C 0010	126+	DC	AL2(17-1)
00033E D7	127+	DC	CL1'P'
00033F 40	128+	DC	CL1' '
	129 *		
	130	IMBS	RECNAME=(WORKCTR,WORK-CEN
000340 FFFFFFFF	131+	DC	F' -1'
000344 0024	132+	DC	H'36'

000346	FFFF	133+	DC	H' -1'
000348		134+R3	DS	0F
000348	00000390	135+	DC	A(R4)
00034C	E6D6D9D2C3E3D9	136+	DC	CL7'WORKCTR'
000353	D4	137+	DC	C'M'
000354	E6D6D9D260C3C5D5	138+	DC	CL16'WORK-CENTER'
000364	0005	139+	DC	H'5'
000366	0020	140+	DC	H'32'
000368	0000000000000000	141+	DC	2F'0'
		142	IMBS	POINTER=(01,X,1)
000370	00000190	143+	DC	A(PTRTAB+16*(01-1))
000374	0000	144+	DC	AL2(1-1)
000376	E7	145+	DC	CL1'X'
000377	40	146+	DC	CL1' '
		147	IMBS	POINTER=(03,N,10)
000378	000001B0	148+	DC	A(PTRTAB+16*(03-1))
00037C	0009	149+	DC	AL2(10-1)
00037E	D5	150+	DC	CL1'N'
00037F	40	151+	DC	CL1' '
		152	IMBS	POINTER=(03,P,14)
000380	000001B0	153+	DC	A(PTRTAB+16*(03-1))
000384	000D	154+	DC	AL2(14-1)
000386	D7	155+	DC	CL1'P'
000387	40	156+	DC	CL1' '
		157 *		
		158	IMBS	RECNAME=(ROUTING,ROUTINGS)
000388	FFFFFFFF	159+	DC	F' -1'
00038C	0020	160+	DC	H'32'
00038E	FFFF	161+	DC	H' -1'
000390		162+R4	DS	0F
000390	000003E8	163+	DC	A(R5)
000394	D9D6E4E3C9D5C7	164+	DC	CL7'ROUTING'
00039B	C3	165+	DC	C'C'
00039C	D9D6E4E3C9D5C7E2	166+	DC	CL16'ROUTINGS'
0003AC	0000	167+	DC	H'0'
0003AE	0054	168+	DC	H'84'
0003B0	0000000000000000	169+	DC	2F'0'
		170	IMBS	POINTER=(04,0,1)
0003B8	000001C0	171+	DC	A(PTRTAB+16*(04-1))
0003BC	0000	172+	DC	AL2(1-1)
0003BE	D6	173+	DC	CL1'0'
0003BF	40	174+	DC	CL1' '
		175	IMBS	POINTER=(04,N,5)
0003C0	000001C0	176+	DC	A(PTRTAB+16*(04-1))

```

0003C4 0004          177+      DC    AL2(5-1)
0003C6 D5           178+      DC    CL1'N'
0003C7 40           179+      DC    CL1' '
                                180      IMBS  POINTER=(03,0,9)
0003C8 000001B0    181+      DC    A(PTRTAB+16*(03-1))
0003CC 0008        182+      DC    AL2(9-1)
0003CE D6           183+      DC    CL1'0'
0003CF 40           184+      DC    CL1' '
                                185      IMBS  POINTER=(03,N,13)
0003D0 000001B0    186+      DC    A(PTRTAB+16*(03-1))
0003D4 000C        187+      DC    AL2(13-1)

0003D6 D5           188+      DC    CL1'N'
0003D7 40           189+      DC    CL1' '
                                190      IMBS  POINTER=(03,P,17)
0003D8 000001B0    191+      DC    A(PTRTAB+16*(03-1))
0003DC 0010        192+      DC    AL2(17-1)
0003DE D7           193+      DC    CL1'P'
0003DF 40           194+      DC    CL1' '
                                195 *
                                196      IMBS  END
0003E0 FFFFFFFF    197+      DC    F' -1'

0003E4 0054        198+      DC    H'84'
0003E6 FFFF        199+      DC    H' -1'
0003E8             200+R5    DS    0F
0003E8 C5D5C44B    201+      DC    CL4'END.'
0003F0             202+BUFFER DS    0D
0003F0 0000000000000000 203+      DC    XL148'0'
000484 C5D5C44B    204+      DC    CL4'END.'
                                205      END

```

IMBSPROC Database Procedure

IMBSPROC, supplied in source and object form on the CA IDMS DML installation media, is a database procedure. This procedure moves pointers of current records (that participate in the sets described in IMBSTAB) from the subschema table to a CA IDMS/DB dummy record. The bridge program BINDs the dummy record to the IMBSTAB pointer table.

Integration of IMBSPROC into the Bridge Program

Integration of IMBSPROC into the bridge program is as follows:

- When a DBOMP program issues a retrieval or update request, the bridge program issues a GET of the dummy record before:
 - Moving the CA IDMS/DB record to the CA IDMS/DB logical record buffer
 - or**
 - Returning the DBOMP record to the database
- When the bridge program issues a GET of the dummy record, CA IDMS/DB calls IMBSPROC. IMBSPROC places currency information (pointers) in the dummy record.
- IMBSPROC moves pointers for the sets identified in the IMBSTAB set table from the subschema table to the dummy record and cancels the GET command issued to CA IDMS/DB.
- IMBSPROC returns the updated dummy record to the bridge program.
- The bridge program proceeds to move the pointers for the requested record from the dummy record into the DBOMP file work area, placing them as specified in IMBSTAB.

Note: To protect the integrity of the CA IDMS/DB database, pointers are **not** returned with record data to the database when a write function has been requested.

What You Need To Do

The bridge program and IMBSPROC logic is transparent to the calling program. You must, however:

- Define the dummy record in the schema
- Include the dummy record in any subschema as that bridged programs use, thereby making it available to IMBSPROC and IMBSBRDG

In the schema RECORD description that describes the dummy record, include a CALL statement that directs CA IDMS/DB to call IMBSPROC before GETting the dummy record.

For example, see this sample COBOL RECORD description:

```
record name is imbs-record.
record id is 799.
location mode is direct.
within bill-of-matrl area.
call imbsproc before get.
```

```
05 imbs-pointers occurs n times.
   10 imbs-pointer pic x(4) occurs 4 times.
```

Code the RECORD description paragraph as shown in the sample, changing the values for RECORD NAME, RECORD ID, and AREA name as necessary. Supply a value for *n* (in the 05-level OCCURS statement) that is less than or equal to the value specified in the SETLMT clause of the IMBS macro control statement.

IMBSBRDG program module

IMBSBRDG is the CA IDMS DML Assembler program module that replaces the DBOMP runtime executable code. Specifically, it replaces:

- The BM\$PIO root module
- The AP\$SEQ module
- All FILEORG modules
- The routines generated by the MF\$SQ, FI\$LE, and CF\$RT macros

IMBSBRDG Interface Between Applications and CA IDMS/DB

IMBSBRDG is an interface between application programs and CA IDMS/DB, and simulates IBM bill-of-materials systems (BOMP, DBOMP, CFMS). IMBSBRDG is linked at runtime with IMBSTAB, IDMS, and the DBOMP application program, and appears to CA IDMS/DB as an application program.

Note: CA IDMS DML does not include operating system and input/output interfaces, and does not issue any messages to the console.

IMBSBRDG simulates the DBOMP environment by:

- Converting DBOMP retrieval or update macros and process indicators to CA IDMS/DB commands
- Converting CA IDMS/DB records to DBOMP records, using information supplied by IMBSTAB.

After converting the DBOMP command and the object record, IMBSBRDG returns the requested data and processing information to the calling program.

Converting DBOMP Calls to CA IDMS/DB Statements

The IMBSBRDG program module simulates DBOMP processing by converting DBOMP calls to CA IDMS/DB statements. IMBSBRDG uses its process indicator table to make the conversion. The executing program:

- Examines the process indicator (found in the work area prefix of the object record)
- Searches the process indicator table for the name of the IMBSBRDG routine that issues the equivalent CA IDMS statement
- Passes control to the appropriate IMBSBRDG routine, which performs the requested retrieval or update function

IMBSBRDG Routines

The following table describes the IMBSBRDG routines.

The IMBSBRDG module supplied on the installation media includes comments for each of these routines as well as for the routines that move pointers and data to and from the DBOMP file work area.

Name of routine	What it does
HOUSEKEEPING (performed on each entry to BM\$PIO and AP\$SEQ)	<ul style="list-style-type: none"> ■ Saves registers ■ Establishes addressability ■ Sets sequential flag for entry to AP\$SEQ
MAINLINE	Routes all calls to IMBSBRDG: <ul style="list-style-type: none"> ■ On first call, passes control to INITIALIZATION routine ■ For all subsequent calls, passes control to PROCESS INDICATOR routine and to FILENAME VERIFICATION routine
INITIALIZATION (performed on initial entry to IMBSBRDG)	<ul style="list-style-type: none"> ■ Establishes location of IMBSTAB tables and loads their addresses ■ Signs on to CA IDMS/DB ■ BINDs CA IDMS/DB dummy record to pointer table in IMBSTAB ■ BINDs all record types to CA IDMS/DB logical record buffer in IMBSTAB ■ READYs the CA IDMS/DB database areas in the specified usage mode ■ Initializes the general CA IDMS/DB call ■ Initializes registers
FILENAME VERIFICATION	Equates the DBOMP file name to a CA IDMS/DB record type name
PROCESS INDICATOR	Equates the DBOMP process indicator to a CA IDMS/DB function
MOVE RECORD	<ul style="list-style-type: none"> ■ For retrieval functions, builds the expected DBOMP record from the CA IDMS/DB logical record buffer and passes the record to the named DBOMP file work area ■ For update functions, extracts the data from the DBOMP file work area and passes the data to the CA IDMS/DB logical record buffer (pointers are not moved from the work area to the CA IDMS/DB logical record buffer)
MRAN MRKY	Performs random record retrieval
DIRECT READ	Performs direct record retrieval

Name of routine	What it does
MODIFY RECORD	Updates in place master and chain file records
SEQUENTIAL READ	Performs processing requested by GE\$T
START KEY	Performs processing requested by ST\$KY
START DA	Performs processing requested by ST\$DA
OPEN	<ul style="list-style-type: none"> ■ Determines location mode of CA IDMS/DB record type that corresponds to named DBOMP file ■ Determines, for future MGET processing, whether CA IDMS/DB record type belongs to an indexed set ■ Determines if RACN function is permitted for named DBOMP file and if so returns file control record to named DBOMP file work area
CLOSE	Returns file control record to CA IDMS/DB database and closes database
EXPAND	Moves disk address to named work area prefix from indicated sending field
COMPRESS	Moves disk address from named work area prefix to indicated receiving field

Converting Records Retrieved from CA IDMS/DB

The IMBSBRDG program converts retrieved CA IDMS/DB records to DBOMP records, reconstructs CA IDMS/DB records from updated DBOMP records, and returns the updated records to the database.

Converting Records

To convert records retrieved from the CA IDMS/DB database, IMBSBRDG performs the following tasks:

- Reads the CA IDMS/DB record into the CA IDMS/DB logical record buffer
- Retrieves the CA IDMS/DB dummy record updated by IMBSPROC
- Moves the pointers for the requested record from the CA IDMS/DB dummy record to the DBOMP file work area (using displacement information in IMBSTAB to determine where to place each pointer)
- Moves segments of data from the CA IDMS/DB logical record buffer to the DBOMP file work area, accounting for the pointers already in place

Pointer displacement information is used in determining the size of each data segment moved:

- The size of the first data segment moved equals the number of bytes between the beginning of the DBOMP logical record and the first pointer
- The size of the second segment moved equals the number of bytes between the first and second pointers
- This process continues until all of the data in the CA IDMS/DB logical record buffer has been moved into the file work area, where the simulated DBOMP record is available for processing by the calling program

Reconstructing and Returning Records

To reconstruct updated DBOMP records and return them to the CA IDMS/DB database, CA IDMS DML performs the following tasks:

- Moves segments of data from the updated DBOMP logical record in the file work area to the CA IDMS/DB logical record buffer.

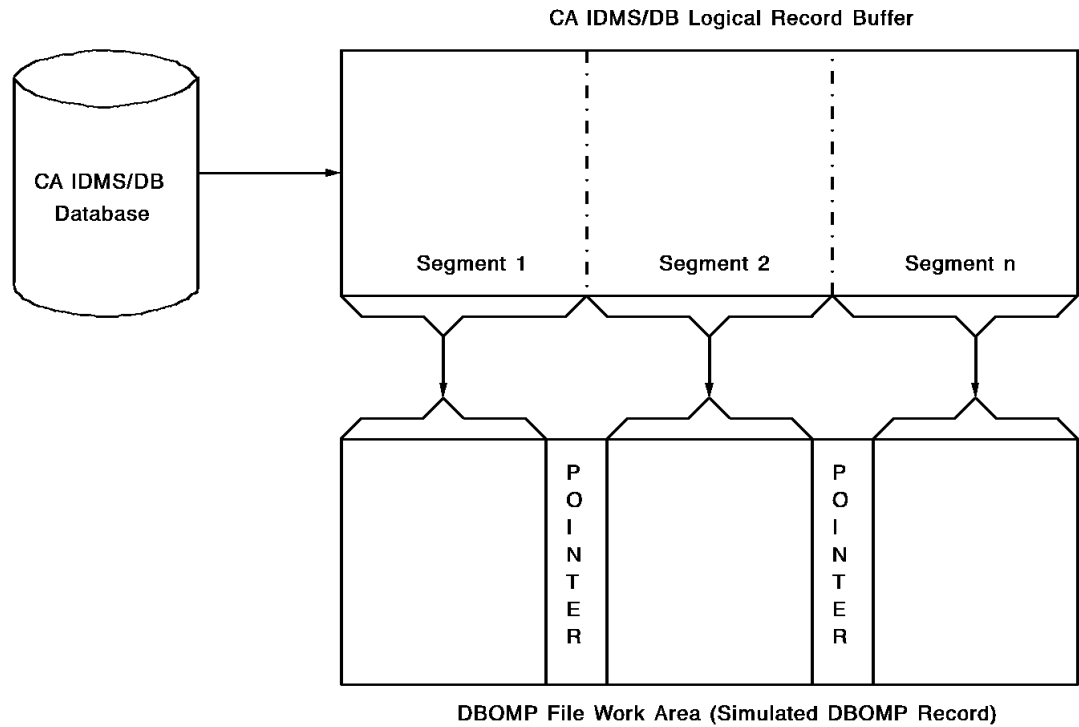
Pointer displacement information is used in determining the size of each data segment:

- The size of the first segment moved equals the number of bytes between the beginning of the DBOMP record and the first pointer
 - The size of the second segment moved equals the number of bytes between first and second pointers
 - This process continues until all data in the DBOMP logical record (except pointers) has been moved to the CA IDMS/DB logical record buffer.
- Issues a MODIFY command to CA IDMS/DB, returning the updated record in the buffer to the database.

The following two figures illustrate how IMBSBRDG moves data between the CA IDMS logical record buffer and the work area of the DBOMP file.

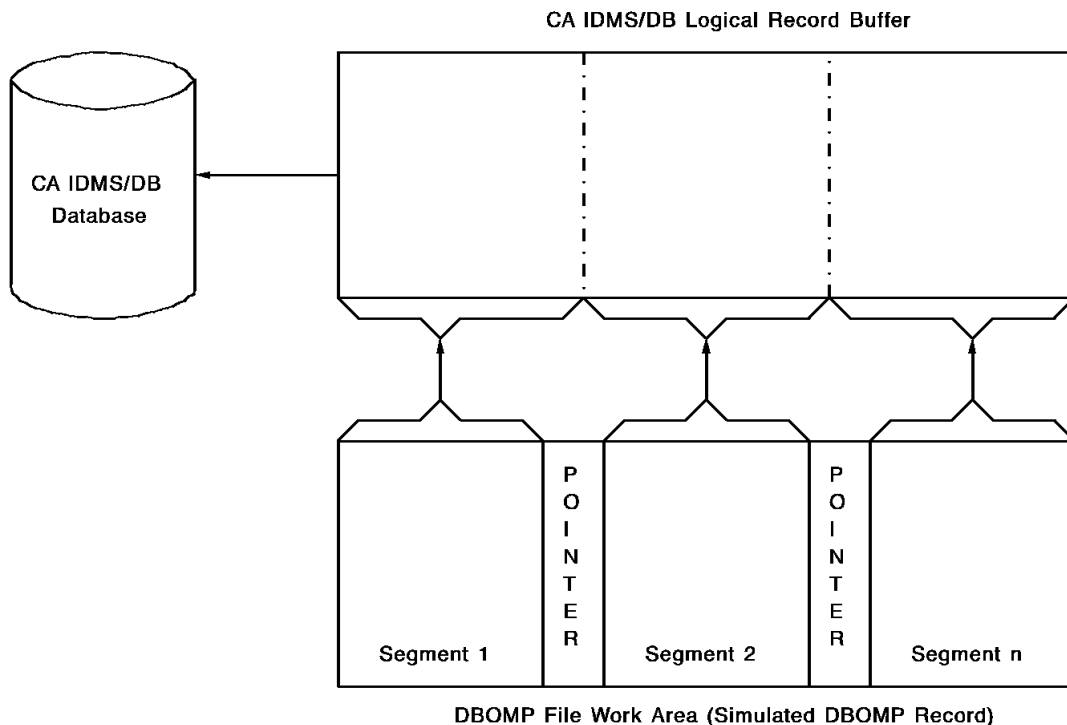
Transfer from IDMS to DBOMP

The following figure shows the transfer of data from the CA IDMS/DB logical record buffer to the work area of the DBOMP file. Note that when the transfer of data takes place, the pointers already have been moved from the CA IDMS/DB dummy record to the DBOMP file work area.



Transfer from DBOMP to IDMS

This figure shows the transfer of data from the work area of the DBOMP file to the CA IDMS/DB logical record buffer. Note that pointers are not returned with record data to the CA IDMS/DB logical record buffer.



Values Returned to the Calling Program

IMBSBRDG returns values to the calling program, as shown in the following table.

Values returned to:	Description of values returned
Work area prefix	<ul style="list-style-type: none"> ■ A hexadecimal value in the error-byte field, returned after a DBOMP request: ■ 0000— Requested function performed successfully ■ 0400— File name not found in IMBSTAB ■ 0004— Process indicator not found in process indicator table ■ 0008— Invalid record at disk address (MDIR and CDIR process indicators) ■ FFFF— Failure in IMBSBRDG program ■ Current disk address, returned when a successful random read (MRAN or MRKY) has been performed ■ Current record key, returned when a successful direct read (MDIR, MRDR, CDIR, or CRDR) has been performed
Work area of the DBOMP file	A DBOMP logical record; after successful execution of a retrieval request
Currency field in IMBSEQ tables	Current address of a record retrieved by a successful execution of the ST\$DA or ST\$KY macro

IMBSEQ macro

IMBSEQ is the Assembler macro that replaces:

- The MF\$SQ macro
- All FI\$LE macros
- The CR\$RT macro in DBOMP Assembler application programs

IMBSEQ generates tables containing information to support the sequential processing requested by GE\$T, PU\$T ST\$DA, and ST\$KY macros in bridged programs. You can place this macro anywhere in the application program, however, it must appear only once.

▶▶ IMBSEQ (*file-name, set-name, end-of-data-address*) ▶▶

IMBSEQ

A required constant that identifies the macro; you can code it anywhere after column 1.

file-name

Specifies the seven-character name of the DBOMP file. One *file-name* entry must exist for every master file referenced in the bridged program.

set-name

Specifies the name of the set as it appears in the subschema.

end-of-data-address

Specifies the end-of-data address for the accompanying *file-name*. One *end-of-data-address* entry must exist for every *file-name*.

IMBSEQ builds one sequential table for each file named in the macro. Each table contains the following values:

- The DBOMP file name
- A last-file flag
- The name of the area for which an area sweep is performed or the name of the index used for sequential access
- The address of the end-of-file routine to which program control is to branch when the end of the file is reached
- The currency field updated after each sequential retrieval

Sequential File Table Layout

The following figure illustrates the layout of the sequential file table.

Displacement	Field Contents	Field Length
0	DBOMP file name	7
7	+ Flag	1
8	Area name or index name	16
24	Address of EOF routine	4
28	Current db-key	4

The IMBSEQ macro requires entries for only those files that are processed sequentially by the DBOMP program. In IMBSTAB, you must describe all files entered in this macro and referenced in the program.

The macros that generate the PL/I and COBOL interfaces include the logic necessary to generate the tables required for sequential processing. The layout for these tables is the same as for those generated by the IMBSEQ macro.

For more information on the PL/I interface, see [PL/I Considerations](#) (see page 85). For more information on the COBOL interface, see [COBOL Considerations](#) (see page 91).

Chapter 5: Converting DBOMP to CA IDMS/DB

This section contains the following topics:

[Overview](#) (see page 59)

[Converting Data](#) (see page 60)

[Converting DBOMP Load and Maintenance Programs](#) (see page 62)

[Converting DBOMP Retrieval and Update Programs](#) (see page 72)

[DBOMP Error Codes With CA IDMS/DB Equivalents](#) (see page 72)

Overview

This chapter provides detailed instructions for converting DBOMP data and programs to CA IDMS/DB.

Conversion Steps

To convert a DBOMP system to CA IDMS/DB, you must:

1. Design the CA IDMS/DB database. Use DBOMP file organization modules, I/O modules, and file description modules as design aids and then discard them; these modules are not integrated into a CA IDMS/DB runtime system.

Note: The Mixed Page Group Binds Allowed feature may not be used with CD IDMS/DBOMP Transparency. For more information on this step, see the *CA IDMS Database Design Guide*.

2. Convert and transfer existing data from the DBOMP database to the CA IDMS/DB database.
3. Convert DBOMP load, maintenance, and retrieval/update programs to CA IDMS/DB.

Cautions on the Duplication of Logic

Because of the basic differences between CA IDMS/DB processing and DBOMP processing, don't expect CA IDMS/DB to duplicate DBOMP logic in all applications. This applies particularly to RACN and chain count routines. Since CA IDMS/DB handles these functions internally, it is usually not necessary to maintain the routines in converted programs.

However, should these routines be required, you must integrate the necessary logic into converted programs. For example, if RACN is implemented in the converted program, you must establish a file control record for each applicable master file and insert the program logic to update it.

Converting Data

To convert and transfer data from a DBOMP database to a CA IDMS/DB database, you write a conversion program that issues calls to DBOMP and to IDMSDBLU.

Note: For more information on IDMSDBLU, see the FASTLOAD section in the *CA IDMS Utilities Guide*.

What the Conversion Program Does

A conversion program does the following:

- Describes each DBOMP master file and equivalent CA IDMS/DB record type (see the information on occurrence descriptors in the FASTLOAD section in the *CA IDMS Utilities Guide*)
- Describes sets, set owners, and record keys to be established on the CA IDMS/DB database (see the information on owner descriptors in the FASTLOAD section in the *CA IDMS Utilities Guide*)
- Issues a DBOMP call to retrieve a record from the parent master file
- Reformats the retrieved DBOMP parent master record into a CA IDMS/DB record
- Issues a call to IDMSDBLU to store the reformatted record on the CA IDMS/DB database
- Establishes set names and record keys
- Issues a DBOMP command for a primary chain chase of the product-structure (internal) chain file anchored in the retrieved parent master record
- Reformats each subordinate master record, as it is retrieved, into a CA IDMS/DB record
- Issues a call to IDMSDBLU for each reformatted subordinate master record to store the record on the CA IDMS/DB database and to connect the record to the appropriate set(s)
- Uses the record key for the parent master record to return it to the user work area; this occurs when the end of the internal chain file is reached
- Issues a DBOMP command for a primary chain chase to retrieve the subordinate master records associated with the parent master record in external relationships
- Reformats each subordinate master record as it is retrieved

- Issues a call to IDMSDBLU to store each reformatted subordinate master record on the CA IDMS/DB database and to connect the record to the appropriate set(s)
- Repeats all of the preceding tasks until the entire parent master file has been read; this occurs when the end of the external chain file is reached

Note: It is recommended that you retain low-level codes when you transfer DBOMP data to a CA IDMS/DB database. If you want to retain sequential dependencies, convert and transfer the DBOMP data as outlined above and describe the record as being stored via its owner, as described under the clause *via set-name set* of the record statement of Schema statements in the *Database Administration* manual. To keep all occurrences of a given record type in physical sequence, they must be stored via a system owned index.

COBOL Example of Conversion Program

The following is an example of a COBOL program that converts DBOMP data to CA IDMS/DB records and loads them into the CA IDMS/DB database.

data division.

working-storage section.

01 dbomp-item.

03 item-pi.

03 item-key.

01 CA IDMS/db-item

03 part-no.

*Refer to CA IDMS Utilities Guide
for information on occurrence descriptors.*

01 dbomp-prodstr.

01 idms-prodstr.

01 dbomp-workctr.

01 idms-workctr.

03 work-no.

01 dbomp-routing.

01 idms-routing.

01 owner-1.

03 set-1.

03 key-1.

*Refer to CA IDMS Utilities Guide
for information on owner descriptors.*

```
01 owner-2.
   03 set-2.
   03 key-2.

procedure division.
   call 'bmpeof' using dbomp-item end-job.
next-item.
   call 'bmpget' using dbomp-item.
   reformat dbomp-item, giving idms-item
   call 'idmsdblu' using idms-item.
   move part-no to key-1.
   move 'item-struct' to set-1.
   move 'where-used' to set-2.
next-structure.
   end-of-chain go to first-route.
   call 'chase' using anlnk nxlkn addnf dbomp-prodstr dbomp-item.
   reformat dbomp-prodstr, giving idms-prodstr
   move part-no to key-2.
   call 'idmsdblu' using idms-prodstr owner-1 owner-2.
   go to next-structure.
first-route.
   move key-1 to item-key.
   move 'mran' to item-pi.
   call 'bmpcall' using dbomp-item.
   move 'item-routing' to set-1.
   move 'work-routing' to set-2.

next-route.
   end-of-chain go to next-item.
   call 'chase' using anlnk nxlkn addnf dbomp-routing dbomp-workctr.
   reformat dbomp-routing, giving idms-routing
   call 'idmsdblu' using idms-routing owner-1 owner-2.
   reformat dbomp-workctr, giving idms-workctr
   call 'idmsdblu' using idms-workctr.
   go to next-route.
```

Converting DBOMP Load and Maintenance Programs

You must convert all DBOMP load and maintenance programs to CA IDMS/DB before you can run them against the CA IDMS/DB database. Converting these programs involves:

- Inserting the necessary CA IDMS/DB DML control statements to prepare the database for processing
- Replacing all DBOMP calls, process indicators, and associated logic with CA IDMS/DB DML statements and associated logic


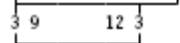
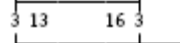

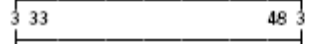


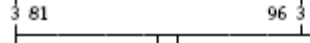

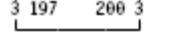
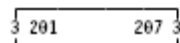
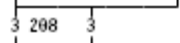
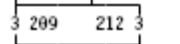
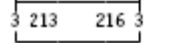
Steps for Converting Load and Maintenance Programs

Follow the eight steps presented below to convert DBOMP Assembler, PL/I, and COBOL load and maintenance programs. To obtain the proper record names and descriptions, set names, area names, and subschema names, consult the dictionary reports produced by the IDMSRPTS utility (see the *CA IDMS Utilities Guide*).

1. Remove all program references to work areas and work area prefixes.
2. Provide a CA IDMS/DB Communications Block for the program, as shown in the figure following this procedure.
3. Allocate space in program variable storage for each CA IDMS/DB record type to be referenced in the converted program. The structure of each record type is described in the data dictionary Subschema Record Description Listing, the SUBREC report generated by the IDMSRPTS utility (see the *CA IDMS Utilities Guide*).
4. Issue an @MODE macro (Assembler only).
5. BIND the subschema and all record types to be referenced in the program.
6. READY those database areas that will be accessed by the program; one READY statement can be issued for all areas, or each area can be READYed explicitly.
7. Replace each DBOMP CA\$LL or BMPCALL with an CA IDMS DML statement equivalent to the function requested by the process indicator in the DBOMP work area prefix. Alter the associated logic as necessary to conform with CA IDMS/DB programming requirements. The section following this list of guidelines shows the DBOMP process indicators (and commands) and their equivalent CA IDMS DML statements and associated logic.
8. Check the CA IDMS/DB error status after every call to CA IDMS/DB (see [DBOMP Error Codes With CA IDMS/DB Equivalents](#) (see page 72)).

Note: Maintain low-level codes in converted structural maintenance programs. You can incorporate this logic into user programs as a subroutine that is invoked following routines that add records to the CA IDMS/DB database. For an example of this low-level code logic, [Sample Application and Procedures](#) (see page 97); you can apply this example to user maintenance programs.

Communications Block from Step 2 of Conversion

	Field	Type	Length (in bytes)	Suggested Initial Value
	Program name	Alphanumeric	8	Program name
	Error-status indicator	Alphanumeric	4	'1400'
	Db-key	Binary	4 (fullword)	0000
	Record name	Alphanumeric	16	Spaces
	Area name	Alphanumeric	16	Spaces
	Error set name	Alphanumeric	16	Spaces
	Error record name	Alphanumeric	16	Spaces
	Error area name	Alphanumeric	16	Spaces
	IDBMSCOM array	Alphanumeric	100	
	Direct db-key	Binary	4 (fullword)	0000
	Reserved for system	Alphanumeric	7	Spaces
	Filler	. . .	1	. . .
	Record occurrence	Binary	4 (fullword)	0000
	DML sequence	Binary	4 (fullword)	0000

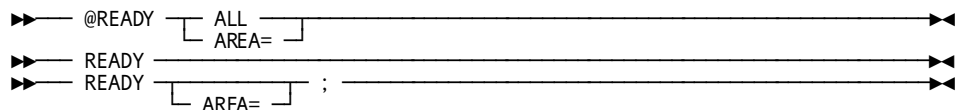
* word aligned

DBOMP Process Indicators and Corresponding DML

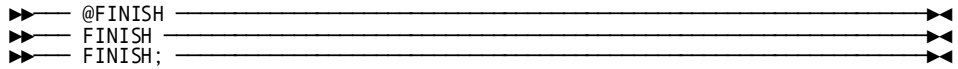
Replacing DBOMP process indicators with equivalent CA IDMS DML statements is part of program conversion (see the steps for converting programs). On the following pages, DBOMP process indicators are shown with their equivalent DML statements (and associated logic, where appropriate). DML statements are shown in this order:

- Assembler
- COBOL
- PL/I

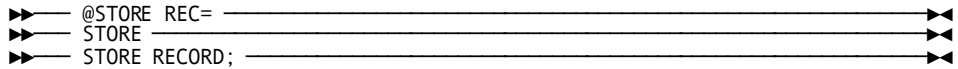
OPEN



CLOS



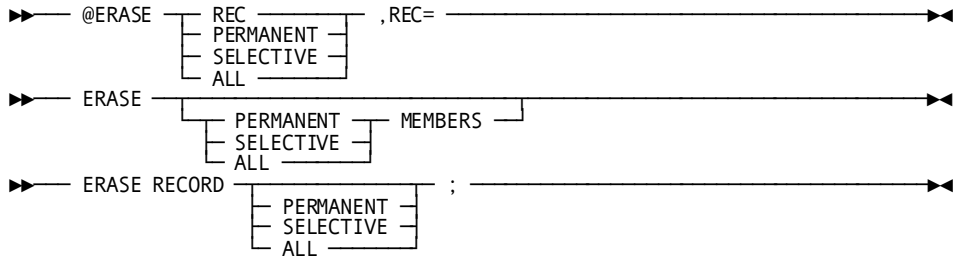
MADD and MCRT



Associated Logic

Build record in user work area and move key to required field before STORE.

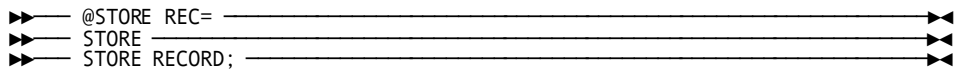
MDEL and MTAG



Associated Logic

For MTAG, insert user logic to accomplish tagging.

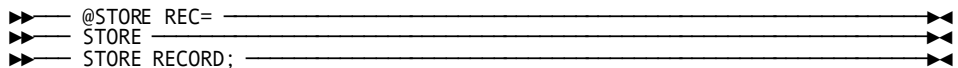
CADD



Associated Logic

Move parent master record key to program variable storage; FIND CALC parent master record; build 'chain' record; move subordinate master key to program variable storage; FIND CALC subordinate master record; CONNECT subordinate master record to appropriate set; perform low-level code routine; set membership for product-structure relationship is MM.

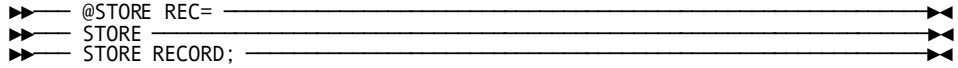
CADD (Subordinate Master)



Associated Logic

Move parent master record key to program variable storage; move subordinate master record key to program variable storage; build 'chain' record in program variable storage; FIND CALC parent master record; FIND CALC subordinate master record; STORE 'chain' record; NOTE: set membership for subordinate master record is assumed MA.

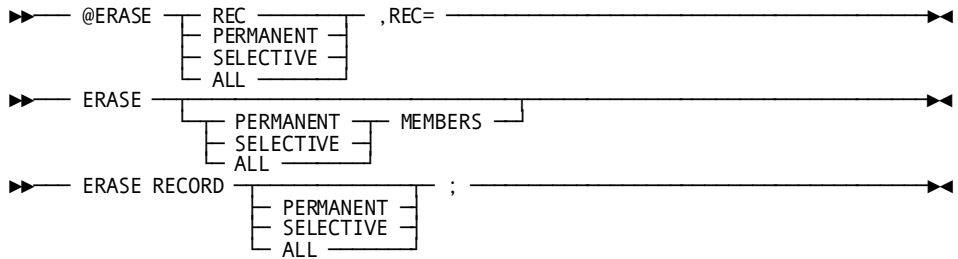
CADD (No Subordinate Master)



Associated Logic

Move master record key to program variable storage; build 'chain' record; FIND CALC master record; STORE 'chain' record.

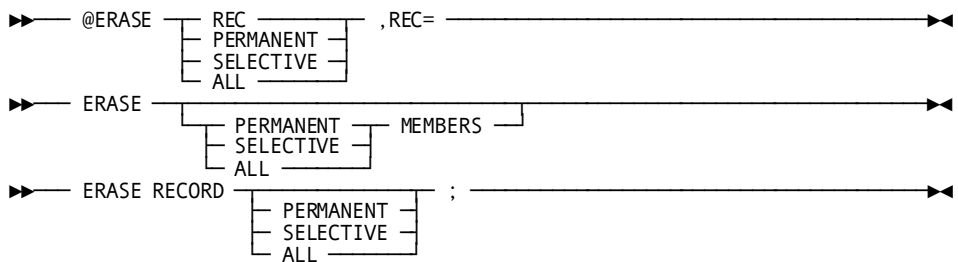
CDLS



Associated Logic

Move master record key to program variable storage; FIND CALC master record; OBTAIN NEXT record within set; check error status; loop until record is found or end of set reached; delete found record.

CDLM



Associated Logic

Move master record to program variable storage; FIND CALC master record; OBTAIN NEXT record within set; delete 'chain' record; check error status; loop until end of set.

CCSR

```

▶▶ @MODIFY REC= _____▶▶
▶▶ MODIFY _____▶▶
▶▶ MODIFY RECORD; _____▶▶
    
```

Associated Logic

Move subordinate master record key to program variable storage; OBTAIN CALC subordinate master record; change subordinate master record key to desired value; MODIFY subordinate master record.

CEQL

```

▶▶ @STORE REC= _____▶▶
▶▶ STORE _____▶▶
▶▶ STORE RECORD; _____▶▶
    
```

Associated Logic

Move parent master record key to program variable storage; FIND CALC parent master record; OBTAIN NEXT record within set; move key of obtained record to program variable storage for parent master record; FIND CALC record; STORE retrieved ('chain') record.

CCHG

```

▶▶ @MODIFY REC= _____▶▶
▶▶ MODIFY _____▶▶
▶▶ MODIFY RECORD; _____▶▶
    
```

Associated Logic

Move master record key to program variable storage; FIND CALC master record; MODIFY record as required.

CFIN and CEND

Have no IDMS equivalents

Associated Logic

If end of set is desire, FIND OWNER within set.

SADD

```

▶▶ @CONNECT REC=, SET= _____▶▶
▶▶ CONNECT TO _____▶▶
▶▶ CONNECT RECORD SET; _____▶▶
    
```

Associated Logic

Move master record key to program variable storage; FIND CALC master record; OBTAIN NEXT record within set; move subordinate record key to master record key in program variable storage; FIND CALC master record; CONNECT found master record to appropriate set.

SDEL

```
▶▶ @DISCON REC=,SET= _____▶▶
▶▶ DISCONNECT FROM _____▶▶
▶▶ DISCONNECT RECORD SET; _____▶▶
```

Associated Logic

FIND CALC record; OBTAIN NEXT record within set; DISCONNECT retrieved record.

CCRT

See information for CADD

MRKY

```
▶▶ @FIND CALC,REC= _____▶▶
▶▶ FIND CALC _____▶▶
▶▶ FIND CALC RECORD; _____▶▶
```

MRAN

```
▶▶ @OBTAIN CALC,REC= _____▶▶
▶▶ OBTAIN CALC _____▶▶
▶▶ OBTAIN CALC RECORD; _____▶▶
```

MDIR

```
▶▶ @OBTAIN DBKEY= _____▶▶
▶▶ OBTAIN DB-KEY IS _____▶▶
▶▶ OBTAIN DBKEY; _____▶▶
```

MRDR

```
▶▶ @FIND DBKEY= _____▶▶
▶▶ FIND DB-KEY IS _____▶▶
▶▶ FIND DBKEY; _____▶▶
```

MUPD

```
▶▶ @MODIFY REC= _____▶▶
▶▶ MODIFY _____▶▶
▶▶ MODIFY RECORD; _____▶▶
```

MWRT

Has no CA IDMS/DB equivalent

CDIR

```

▶▶ @OBTAIN DBKEY= _____
▶▶ OBTAIN DB-KEY IS _____
▶▶ OBTAIN DBKEY _____
    
```

CUPD

```

▶▶ @MODIFY REC= _____
▶▶ MODIFY _____
▶▶ MODIFY RECORD; _____
    
```

Associated Logic

OBTAIN record before issuing MODIFY.

CWRT

Has no CA IDMS/DB equivalent

CMPR and EXPN

Have no CA IDMS/DB equivalents; addresses are not compressed in CA IDMS/DB

DBOMP Commands and Corresponding DML

Replacing DBOMP commands with equivalent CA IDMS DML statements is part of program conversion (see the previous list of guidelines for conversion). On the following pages DBOMP commands are shown with their equivalent DML statements (and associated logic, where appropriate). DML statements are shown in this order:

- Assembler
- COBOL
- PL/I

CHASE BMPCHASE

See associated logic

Associated Logic

FIND CALC set owner record; OBTAIN NEXT record (member) within set; check for the end of the set; repeat OBTAIN NEXT and check error status until the end of the set.

GET BMPGET

```

▶▶ @OBTAIN NEXT, [ SET= _____ ]
▶▶ OBTAIN NEXT WITHIN [ AREA= _____ ]
▶▶ OBTAIN NEXT [ SET _____ ] ; _____
▶▶ [ AREA _____ ]
    
```

PU\$T BMPPUT

```
▶▶— @MODIFY REC= _____▶▶
▶▶— MODIFY _____▶▶
▶▶— MODIFY RECORD; _____▶▶
```

ST\$KY BMPSTKY

```
▶▶— @OBTAIN ,REC= ,SET= ,USING= _____▶▶
▶▶— OBTAIN WITHIN USING _____▶▶
▶▶— OBTAIN RECORD SET USING; _____▶▶
```

Associated Logic

Obtains a record in an indexed set using a symbolic key.

ST\$DA BMPSTDA

```
▶▶— @OBTAIN DBKEY=DIRCTKY ,REC= _____▶▶
▶▶— OBTAIN DB-KEY IS DIRECTKY _____▶▶
▶▶— OBTAIN DBKEY DIRCTKY; _____▶▶
```

Associated Logic

Record retrieved in physical sequential order by symbolic key. (DIRCTKY)

CA\$LL BMPCALL

See process indicator equivalents

Commands having no equivalents

These DBOMP commands have no CA IDMS/DB equivalents:

- BM\$WA
- MSG
- TY\$PE
- MO\$VE
- EQ\$RG
- BM\$DS
- MF\$SQ
- CF\$RT
- FI\$LE
- CGE\$T
- CPU\$T
- BM\$FO

- BMPFO
- EO\$F
- BMPEOF
- BM\$RACN
- BMPRACN
- BM\$OFAD
- BMPOFFAD

Sequence of Logic in Converted Programs

The general sequence of logic in the converted load and maintenance programs should be as follows:

1. Read input data or transaction record.
2. Format the input data into the CA IDMS/DB record work area. (The COBOL code to accomplish this is generated automatically.)
3. Establish necessary currencies.
4. Issue the appropriate DML Assembler macro:
 - **@STORE**— Add a record occurrence to the database.
 - **@ERASE**— Delete a record occurrence from the database.
 - **@MODIFY**— Alter a record key or sequence field.
 - **@CONNECT**— Add a record occurrence to a set occurrence.
 - **@DISCONNECT**— Remove a record occurrence from a set occurrence.
5. Check the status code returned by CA IDMS/DB (see [DBOMP Error Codes With CA IDMS/DB Equivalents](#) (see page 72)).

Note: Check the CA IDMS/DB status after *every* call to CA IDMS/DB to determine whether the requested function was performed. The status codes returned to the program may indicate program errors, or they may be tested by program logic to determine subsequent program action. For more information on status codes and their meanings, see the *CA IDMS DML Reference Guide for COBOL* and the *CA IDMS DML Reference Guide for PL/I*.

Converting DBOMP Retrieval and Update Programs

The final task in conversion to CA IDMS/DB is converting DBOMP retrieval and update programs.

Steps for Converting Retrieval and Update Programs

Follow the eight steps presented below to convert DBOMP Assembler, PL/I, and COBOL load and maintenance programs to CA IDMS/DB. To obtain the proper record names and descriptions, set names, area names, and subschema names, consult the data dictionary reports produced by the IDMSRPTS utility (see the *CA IDMS Utilities Guide*).

1. Remove all program references to DBOMP filework areas and work area prefixes.
2. Provide a CA IDMS/DB Communications Block for the program (see the same step under [Converting DBOMP Load and Maintenance Programs](#) (see page 62), in this chapter).
3. Allocate space in the CA IDMS/DB program variable storage for each CA IDMS/DB record type to be referenced in the converted program. The structure of each record type is described in the dictionary Subschema Record Description Listing, or SUBREC report.

Note: For more information on SUBREC, see IDMSRPTS in the *CA IDMS Utilities Guide*.

4. Issue an @MODE macro (Assembler only).
5. BIND the subschema and all record types to be referenced in the program.
6. READY those database areas that will be accessed by the program; one READY statement can be issued for all areas, or each area can be READYed explicitly.
7. Convert each DBOMP command and accompanying process indicator to an equivalent DML command. Alter the program logic associated with the DBOMP command as necessary to conform with CA IDMS/DB programming requirements. Refer to the syntax shown under [Converting DBOMP Load and Maintenance Programs](#) (see page 62) for the CA IDMS/DB statements that are equivalent to DBOMP commands save process indicators.
8. Check the status code returned by CA IDMS/DB after every call to CA IDMS/DB (see the table under [DBOMP Error Codes With CA IDMS/DB Equivalentents](#) (see page 72)).

DBOMP Error Codes With CA IDMS/DB Equivalentents

DBOMP Code	DBOMP P.I.	IDMS Status	IDMS Macro	Meaning
0400	Any	0308	Any	Invalid record type

DBOMP Code	DBOMP P.I.	IDMS Status	IDMS Macro	Meaning
0200	Addition	1211	@STORE	No space in area
0008	File read	0326	@FIND/@OBTAIN	Record not found
0008	File read	0302	@FIND/@OBTAIN	Db-key not within page range for specified record
0004	Any	xx63	—	Invalid function
0001	Addition	1205	@STORE	Duplicate record
0001	Deletion	0230	@ERASE	Record occurrence is owner of nonempty set
END	CHASE	0307	@OBTAIN NEXT, SET= AREA=	End of set or area

Chapter 6: Using the Transparency as a Bridge to CA IDMS/DB

This section contains the following topics:

[Overview](#) (see page 75)

[Preparing DBOMP Assembler Programs](#) (see page 75)

[Executing DBOMP Assembler Programs](#) (see page 76)

[Diagnosing Errors](#) (see page 81)

Overview

You can use the CA IDMS DBOMP Transparency as a bridge between your existing unconverted DBOMP application program and a database that has been converted from DBOMP to CA IDMS/DB. Using the transparency involves these activities:

- Preparing DBOMP programs for processing
- Executing the programs
- Locating and diagnosing program errors that occur during processing

This chapter explains the procedures you use to prepare and execute Assembler programs and for diagnosing errors in bridged Assembler, PL/I, and COBOL programs.

For more information on preparing and executing PL/I programs, see [PL/I Considerations](#) (see page 85). For more information on preparing and executing COBOL programs, see [COBOL Considerations](#) (see page 91).

Preparing DBOMP Assembler Programs

The amount of preparation necessary to make a DBOMP Assembler program acceptable to the transparency varies based on the functions performed by the program. Before submitting a DBOMP Assembler application program via the transparency, make the following changes:

- Remove any MF\$SQ, FI\$LE, or CF\$RT macros from the program. Replace the macros with the IMBSEQ macro.

Note: The IMBSEQ macro must appear only once in the program.

- Remove any program logic that depends on RACN support for more than one file (record type). IMBSBRDG ignores program reference to file control records for files other than the one designated in IMBSTAB as using RACN.

- If the program issues any allowable CAIDMS/DB verbs, insert the proper calls to IMBSBRDG (see [The Transparency Environment](#) (see page 21)). Use IDMS-REQUEST as the work area file name.
- If any retrieval or update process indicators other than those supported by the transparency are used in the program, replace them with process indicators that are supported (see [The Transparency Environment](#) (see page 21)).

Executing DBOMP Assembler Programs

Perform these steps to execute a DBOMP Assembler program using the transparency:

1. Assemble IMBSTAB by submitting the user customizing parameters to the IMBS macro. (Omit this step and the next step if an existing version of IMBSTAB is compatible with the application program.) The third and fourth steps are required only for sequential processing of DBOMP files.
2. Link edit IMBSTAB.
3. Assemble the IMBSEQ macro with the IMBSASMB interface macro, specifying the user-defined parameters for the IMBSEQ macro.
4. Link edit the IMBSEQ macro.
5. Reassemble and link edit the DBOMP application program, including IMBSBRDG, IMBSTAB, IMBSEQ, and IDMS.

Note 1: IDMS 16.0 supports Z/OS V2R10 as well as z/OS 1.1 and above. However, we will always refer to z/OS in this document.

Note 2: Programs running under z/OS need only be reassembled if any of the changes detailed above have been made; programs running under Z/VSE must be reassembled whether or not any of these changes have been made, unless the programs exist in the relocatable library.

6. Execute the DBOMP application program. The program is now bridged to CA IDMS/DB.

The JCL you use to execute each of these tasks is provided on the following pages.

Assembling and Executing Under z/OS

[z/OS/Central Version](#)

The following is the JCL for assembling and executing DBOMP Assembler programs using the transparency, in a z/OS operating system, under the central version.

Assemble/Execute DBOMP Assembler Program Using the Transparency (IMBSBRDG) (z/OS)

```

//ASMTABLE    EXEC ASMA90
//ASM.SYSLIB  DD DISP=SHR,DSN=yourHLQ.CAGJSRC
//           DD DISP=SHR,DSN=imbs.srcLib
//ASM.SYSIN   DD DISP=SHR,DSN=yourHLQ.CAGJSRC(imbstab)
//LKED.SYSLMOD DD DISP=SHR,DSN=imbs.loadLib(imbstab)
/*
//ASMPROG     EXEC ASMA90
//ASM.SYSLIB  DD DISP=SHR,DSN=cfms.srcLib
//ASM.SYSIN   DD *
    DBOMP program statements
    END
/*
//LKED.SYSLMOD DD DISP=SHR,DSN=user.loadLib(pgmname)
//LKED.IDMSLIB DD DISP=SHR,DSN=idms.loadLib
//LKED.IMBSLIB DD DISP=SHR,DSN=imbs.loadLib
    INCLUDE IDMSLIB(IDMS)
    INCLUDE IMBSLIB(IMBSBRDG)
    INCLUDE IMBSLIB(imbstab)
    INCLUDE IMBSLIB(IMBSEQ)
/*
//RUNPROG     EXEC PGM=pgmname
//STEPLIB     DD DSN=user.loadLib,DISP=SHR
              DD DSN=idms.dba.loadLib,DISP=SHR
              DD DSN=idms.loadLib,DISP=SHR
additional JCL for application program, as required
//SYSOUT      DD SYSOUT=A
//SYSUDUMP    DD SYSOUT=A
//SYSCTL      DD DSN=idms.sysctl,DISP=SHR
//SYSIDMS     DD *
DMCL=dmcl-name
Other SYSIDMS parameters, as appropriate
/*
program input, as required

```

Include as many STEPLIB DD statements as there are libraries containing program, CA IDMS DBOMP Transparency, and CA IDMS/DB load modules.

Note: If you are going to use the transparency frequently under the central version, consider making IMBSPROC and any applicable subschemas resident. Assemble and link IMBSEQ as described previously and include it on the link edit of the application. For more information on optional SYSIDMS runtime parameters, see the *CA IDMS Common Facilities Guide*.

z/OS/Local Mode

To run the same job in local mode, substitute the following statements after the //STEPLIB statement:

```
//STEPLIB      DD DSN=user.loadLib,DISP=SHR
//             DD DSN=imbs.loadLib,DISP=SHR
//             DD DSN=idms.dba.loadLib,DISP=SHR
//             DD DSN=idms.loadLib,DISP=SHR
//sysjrn1 DD DSN=idms.tapejrn1,DISP=(NEW,PASS),
//             UNIT=tape
//userdd      DD DSN=database,DISP=(OLD,PASS)
//SYSIDMS     DD *
DMCL=dmcl-name
additional SYSIDMS parameters, as appropriate
/*
additional database file assignments, as required
additional JCL for application program, as required
//SYSOUT     DD SYSOUT=A
//SYSUDUMP   DD SYSOUT=A
program input, as required
```

Explanation of Variables

yourHLQ.CAGJMAC	Dataset name for CA IDMS/DB macro library
imbs.srclib	Dataset name for the transparency or CA IDMS/DB source library containing IMBS customizing macro
disk	Symbolic device name for disk unit
&.&object.	Temporary dataset name for IMBSTAB object module
imbs.srclib(imbstab)	Dataset name for user parameters input to IMBS customizing macro
idms.dba.loadlib	Dataset name for the load library containing the DMCL and database name table load modules
idms.loadlib	Dataset name for the load library containing CA IDMS executable modules
imbs.loadlib	Dataset name for the transparency or CA IDMS/DB load library containing transparency modules
imbstab	Dataset name for link edited output from IMBS macro
cfms.maclib	Dataset name for user macro library
user.loadlib	Dataset name for load library containing DBOMP application program
pgmname	Name of DBOMP application program

dmcl-name	Name of the CA IDMS DMCL describing the CA IDMS files used by the transparency
sysjrn1	DD name for CA IDMS/DB journal file
idms.tapejrn1	Dataset name for CA IDMS/DB journal file
tape	Symbolic device name for CA IDMS/DB journal file
userdb	DD name for CA IDMS/DB database file
user.userdb	Dataset name for CA IDMS/DB database file
sysctl	Dataset name for the SYSCTL file

CA IDMS DBOMP Transparency database procedure

Assembling and Executing Under Z/VSE

Z/VSE/Central Version

The following is the JCL for assembling and executing DBOMP Assembler programs using the transparency, in a Z/VSE operating system, under the central version. Note that you can use either an UPSI statement or a SYSCTL statement to indicate central version.

Assemble/Execute DBOMP Assembler Program Using the Transparency (IMBSBRDG) (Z/VSE)

```
// ASSGN SYSPCH,X'281'
// OPTION DECK
   CATALR imbstab
// EXEC ASMA90
user input parameters for IMBS customizing macro
   END
/*
// MTC REW,X'281'
// ASSGN SYSIPT,X'281'
// EXEC MAINT
/*
// OPTION CATAL
   PHASE pgmname
// EXEC ASMA90
program statements
   END
```

```
/*
    INCLUDE IMBSBRDG
    INCLUDE imbstab
    INCLUDE IDMS
// EXEC LNKEDT
/&
// JOB EXECPGM
// UPSI      1
// DLBL SYSIDMS,'#SYSIPT',0,SD
DMCL=dmc1-name
Other SYSIDMS runtime parameters, as appropriate
/*
additional JCL for application program, as required
// EXEC pgmname
program input, as required
/*
```

Note: If you are going to use the transparency frequently under the central version, consider making IMBSPROC and any applicable subschemas resident.

Z/VSE/Local Mode

To run the same job in local mode, substitute the following JCL after the // JOB EXECPGM statement:

```
// ASSGN sys009,X'281'
// ASSGN sys010,X'137'
// DLBL sys010,'database',,DA
// EXTENT sys010,444444,1,76,1776
additional database assignments, as required
additional JCL for application program, as required
// EXEC pgmname
// DLBL SYSIDMS,'#SYSIPT',0,SD
DMCL=dmc1-name
Other SYSIDMS runtime parameters, as appropriate
program input, as required
/*
```

Explanation of Variables

pgmname	Name of DBOMP application program
imbstab	Dataset name for link edited output from IMBS customizing macro
sys009	Logical unit assignment for CA IDMS/DB journal file
281	Physical device assignment for CA IDMS/DB journal file
sys010	Logical unit assignment for CA IDMS/DB database file

137	Physical device assignment for CA IDMS/DB database file
database	Dataset name for CA IDMS/DB database file
444444	Serial number of disk containing CA IDMS/DB database file
76	Relative track where CA IDMS/DB database file begins
1776	Number of tracks used by CA IDMS/DB database file
dmcl-name	Name of the CA IDMS DMCL describing the CA IDMS files used by the transparency

Diagnosing Errors

Since the CA IDMS DBOMP Transparency does not issue diagnostic messages, you must locate and diagnose errors that occur during the execution of a bridged DBOMP program.

Note: If the bridge system aborts:

- z/OS issues an SOC2 program check message
- Z/VSE issues a PRIVILEGED OPERATION EXCEPTION message

What to Look For When Errors Occur During Program Processing

Error-byte Field

Check the error-byte field in the work area prefix of each file processed by the program. The contents of the error-byte field indicate:

- Whether the error occurred during IMBSBRDG processing
- Which file was being handled at the time the error occurred

If the error-byte field of a work area prefix contains a value other than '0000', the error occurred while that file was being handled by IMBSBRDG.

For more information on error-byte values, see [IMBSBRDG program module](#) (see page 48).

CA IDMS/DB Communications Block

Check the CA IDMS/DB communications block (SSCTRL) in IMBSTAB. If an error occurred during CA IDMS/DB processing, the IDMS Communications Block will contain an error status code other than '0000' and the name of the record last involved in the operation that resulted in the error.

Note: For more information on the complete listing of CA IDMS/DB error codes, see the *CA IDMS Messages and Codes Guide*.

Process Indicators

Check which process indicator in the work area prefix was being handled at the time that the error occurred.

IMBSBRDG generates this process indicator:	In response to:
MGET	GE\$T
MPUT	PU\$T
STKY	ST\$KY
STDA	ST\$DA

Table Generation and Accuracy

Verify that the IMBSEQ, IMBSCOB, or IMBSPL1 table has been generated and is accurate if any sequential processing functions are requested by the program.

Subschema and DMCL Module

Verify that the subschema name known to CA IDMS/DB is available, and that the DMCL module is available.

IMBS Parameters

Verify the accuracy of the parameters input to the IMBS customizing macro.

What to Look For When Inaccurate Data is Returned

If your program runs successfully but returns inaccurate data to the work area, make sure:

- The CA IDMS/DB subschema record descriptions agree with the DBOMP file descriptions
- The file table in IMBSTAB contains correct file types and pointer displacements
- The CA IDMS/DB files are loaded properly

Where to Find Values During Debugging

The following table lists the registers that point to the location of transparency components containing values pertinent to the debugging process.

Register	Points to:
R5	IMBSTAB
R6	Active work area prefix
R7	Active file and file table in IMBSTAB
R8	Active record name
R11	CA IDMS/DB logical record buffer in IMBSTAB
R12	Beginning of active IMBSBRDG routine
R14	Instruction following a branch to FORCEDMP; important only when the message program check S0C2 (z/OS) or PRIVILEGED OPERATION EXCEPTION (Z/VSE) has been issued

Appendix A: PL/I Considerations

This section contains the following topics:

[Overview](#) (see page 85)

[Transparency Support For DBOMP PL/I Commands](#) (see page 85)

[IMBSPL1 Interface Macro](#) (see page 87)

[DBOMP PL/I Program Preparation and Execution](#) (see page 88)

Overview

This appendix provides you with additional information necessary to use DBOMP PL/I programs with CA IDMS DBOMP Transparency.

Except as noted here, CA IDMS DBOMP Transparency bridges DBOMP PL/I programs in the same manner it bridges DBOMP Assembler programs.

The topics covered in this appendix are:

- CA IDMS DBOMP Transparency support of DBOMP PL/I commands
- IMBSPL1 interface macro
- DBOMP PL/I program preparation and execution

Transparency Support For DBOMP PL/I Commands

The transparency's support of DBOMP PL/I commands parallels that of DBOMP Assembler macros. The following table shows DBOMP PL/I commands and their interpretation by the CA IDMS DBOMP Transparency.

Note: See IBM DBOMP documentation for the syntax for these commands.

DBOMP PL/I command	CA IDMS DBOMP Transparency interpretation of command
OP\$EN	The first call to OP\$EN causes IMBSBRDG to open the entire CA IDMS/DB database and prepare it for processing: BINDs are issued for the run unit and all record types described in the subschema, and database areas are READYed. The transparency returns the file control record for the file for which RACN has been specified in IMBSTAB. Subsequent calls to OP\$EN are ignored once the database has been opened.

DBOMP PL/I command	CA IDMS DBOMP Transparency interpretation of command
CLO\$E	The first call to CLO\$E causes IMBSBRDG to close all areas in the CA IDMS/DB database by issuing a FINISH command. Subsequent calls to CLO\$E are ignored once the database has been closed. If any command other than CLO\$E is issued after the first CLO\$E, the transparency automatically reopens the CA IDMS/DB database and processes the command; a subsequent CLO\$E causes the transparency to close the database again.
CA\$LL	The work area prefix for the named file is passed to IMBSBRDG, which interprets the process indicator contained in the work area prefix and performs the requested function. See Chapter 3, "The Transparency Environment" for those process indicators supported by the transparency.
GE\$T	IMBSBRDG retrieves the first record in the named file and returns it to the work area. Subsequent calls to GE\$T using the same file cause IMBSBRDG to retrieve records in logical sequential order from that point if the record type is not indexed. When an end-of-file condition is detected, control is passed to the routine specified for the file in the EO\$F command (discussed below).
EO\$F	IMBSBRDG handles EO\$F in the same manner as does DBOMP, but obtains the necessary file information from the module generated by the IMBSPL1 interface macro (see below) rather than from the module generated by the DBOMP PL\$BM macro. A call to EO\$F must specify the end-of-file routines in the same sequence as the corresponding files are entered in the IMBSPL1 macro.
ST\$KY	IMBSBRDG retrieves a record by the key specified in the work area prefix for the named file and returns the record to the work area. The currency for the file is set at the retrieved record. Subsequent GE\$T commands for the file retrieve records in logical sequential order from that point if the record type is not indexed. Note that the transparency support of logical sequential processing assumes the use of an index.
ST\$DA	IMBSBRDG retrieves a record by the disk address specified in the work area prefix for the named file and returns the record to the work area. The currency for the file is set at the retrieved record. Subsequent GE\$T commands for the file retrieve records in logical sequential order from that point if the record type is indexed, or in physical sequential order from that point if the record type is not indexed. Note that the transparency's support of logical sequential processing assumes the use of indexing.

DBOMP PL/I command	CA IDMS DBOMP Transparency interpretation of command
PU\$T	IMBSBRDG writes back to the CA IDMS/DB database the last record retrieved by a GE\$T command. Chain address fields (pointers) are not updated or written back to the database.
CHASE	The transparency supports this command unconditionally. Programs that request only the CHASE function need not be modified before interfacing with the bridge, and should be linked with the PL\$CH macro as indicated in IBM DBOMP documentation.
BM\$OFAD	The transparency does not support this command. If a call to BM\$OFAD is encountered by the bridge, no action takes place and control returns to the calling program.
BM\$FO	The transparency does not support this command. If a BM\$FO command is encountered, an unresolved external reference results in the link edit map.
BM\$RACN	The transparency does not support this command. If a BM\$RACN command is encountered, no action takes place and control returns to the calling program. The transparency's maintenance of RACN in PL/I programs is the same as for Assembler programs.

IMBSPL1 Interface Macro

The IMBSPL1 interface macro replaces the DBOMP PL\$BM macro. This Assembler macro generates tables containing the information necessary to establish communication between the DBOMP PL/I program and IMBSBRDG. Also incorporated in these tables is the information required to support the sequential processing requested by calls to GE\$T, PU\$T, ST\$KY, and ST\$DA.

Syntax

IMBSPL1 macro

```

▶▶ IMBSPL1 ( file-name,  index-set-name  NOTSEQ ),  YES  NO ; ▶▶

```

Parameters

IMBSPL1

A required constant that identifies the macro; you can code it anywhere after column 1.

file-name

The seven-character name of the DBOMP master file as specified in the program work area. You must enter the routines named in the EO\$F command in the same order as you enter the corresponding file names in the IMBSPL1 macro. This ensures that the address of the proper routine is passed to IMBSBRDG when the end of a file named in a GE\$T command is reached. One *file-name* entry must be present for each DBOMP file that is processed.

index-set-name/NOTSEQ

The name of the index set to be used for logical sequential processing; specify NOTSEQ if the file is not to be processed in logical sequential order. One *index-set-name*/NOTSEQ entry must be present for each *file-name* entry.

YES/NO

The compiler option indicator; specified as follows:

- YES if the optimizing compiler is used and IMBSPL1 is not identified as an assembler entry
- NO if the D- or F-level compiler is used

Note: It is recommended that you name every file on the DBOMP database in one execution of the IMBSPL1 macro so that this macro does not need to be assembled and link edited more than once.

Assembling and Linking IMBSPL1

To assemble and link-edit IMBSPL1, you must use SMP/E (Z/OS) or MSHP (Z/VSE).

Note: For more information on using SMP/E and MSHP, see the *CA IDMS Installation—Z/OS* or the *CA IDMS Installation—Z/VSE*.

DBOMP PL/I Program Preparation and Execution

The guidelines for preparing a DBOMP PL/I program and executing it using the transparency parallel those detailed for DBOMP Assembler programs in [Using the Transparency as a Bridge to CA IDMS/DB](#) (see page 75).

Preparing the PL/I Program

- Remove the PL\$BM macro.
- Remove those DBOMP PL/I commands that are not supported by CA IDMS DBOMP Transparency and modify associated program logic as necessary.
- Modify the PL/I logic as necessary to conform with CA IDMS DBOMP Transparency specifications for sequential processing and RACN processing.
- If the program issues any of the allowable CA IDMS DML statements, insert the following call to IMBSBRDG, making sure that the CA IDMS DML statement argument is available in program variable storage (see [The Transparency Environment](#) (see page 21)):

```
call ca$ll (argument_name, 'end.')
```
- If any retrieval or update process indicators except for those supported by CA IDMS DBOMP Transparency (see [The Transparency Environment](#) (see page 21)) are used in the program, replace them with those that are supported.

Executing the Program

- Assemble and link edits IMBSTAB if a version compatible with the application does not exist in the load library.
- Recompile and link edit the DBOMP PL/I program, including IMBSBRDG, IMBSTAB, IMBSPL1, and CA IDMS/DB. This step assumes that IMBSPL1 has been assembled and link edited as discussed above.

Note: You do not need to recompile programs that run under Z/OS unless any of the changes listed above have been made; you must, however, recompile programs that run under Z/VSE whether or not any of these changes have been made, unless the programs exist in the relocatable library.

- Submit the DBOMP PL/I program for execution.

Appendix B: COBOL Considerations

This section contains the following topics:

[Overview](#) (see page 91)

[Transparency Support For DBOMP COBOL Commands](#) (see page 91)

[IMBSCOBOL Interface Macro](#) (see page 93)

[DBOMP COBOL Program Preparation and Execution](#) (see page 94)

Overview

This appendix provides you with additional information necessary to interface DBOMP COBOL programs with CA IDMS DBOMP Transparency.

Except as noted in this appendix, CA IDMS DBOMP Transparency bridges DBOMP COBOL programs in the same manner as it bridges DBOMP Assembler programs.

The topics covered in this appendix are:

- CA IDMS DBOMP Transparency support of DBOMP COBOL commands
- IMBSCOBOL interface macro
- DBOMP COBOL program preparation and execution

Transparency Support For DBOMP COBOL Commands

The transparency's support for DBOMP COBOL commands parallels its support for DBOMP Assembler macros. The following table shows DBOMP COBOL commands and their interpretation by the transparency.

Note: See IBM DBOMP documentation for the syntax for these commands.

DBOMP PL/I command	CA IDMS DBOMP Transparency interpretation of command
BMPOPEN	The first call to BMPOPEN causes IMBSBRDG to open the entire CA IDMS/DB database and prepare it for processing: BINDs are issued for the run unit and all record types described in the subschema, and database areas are READYed. The transparency returns the file control record for the file for which RACN has been specified in IMBSTAB. Subsequent calls to BMPOPEN are ignored once the database has been opened.

DBOMP PL/I command	CA IDMS DBOMP Transparency interpretation of command
BMPCLOSE	The first call to BMPCLOSE causes IMBSBRDG to close all areas in the CA IDMS/DB database by issuing a FINISH command. Subsequent calls to BMPCLOSE are ignored once the database has been closed. If any command other than BMPCLOSE is issued after the first BMPCLOSE, the transparency automatically reopens the CA IDMS/DB database and processes the command; a subsequent BMPCLOSE causes the transparency to close the database again.
BMPCALL	The work area prefix for the named file is passed to IMBSBRDG, which interprets the process indicator contained in the work area prefix and performs the requested function. For information on process indicators that are supported by the transparency, see Chapter 3, "The Transparency Environment".
BMPGET	IMBSBRDG retrieves the first record in the named file and returns it to the work area. Subsequent calls to BMPGET using the same file cause IMBSBRDG to retrieve records in logical sequential order from that point if the record type is indexed, or in physical sequential order from that point if the record type is not indexed. When an end-of-file condition is detected, control passes to the routine specified for the file in the BMPEOF command (discussed below).
BMPEOF	IMBSBRDG handles BMPEOF in the same manner as does DBOMP, but obtains the necessary file information from the module generated by the DBOMP CB\$BM macro. A call to BMPEOF must specify the end-of-file routines in the same sequence as the corresponding files are entered in the IMBSCOBOL macro.
BMPSTKY	IMBSBRDG retrieves a record by the key specified in the work area prefix for the named file and returns the record to the work area. The currency for the file is set at the retrieved record. Subsequent BMPGET commands for the file retrieve records in logical sequential order from that point if the record type is indexed, or in physical sequential order from that point if the record type is not indexed. Note that the transparency's support of logical sequential processing assumes the use of indexing.

DBOMP PL/I command	CA IDMS DBOMP Transparency interpretation of command
BMPSTDA	IMBSBRDG retrieves a record by the disk address specified in the work area prefix for the named file and returns the record to the work area. The currency for the file is set at the retrieved record. Subsequent BMPGET commands for the file retrieve records in logical sequential order from that point if the record type is indexed, or in physical sequential order from that point if the record type is not indexed. Note that the transparency's support of logical sequential processing assumes the use of indexing.
BMPPUT	IMBSBRDG writes back to the CA IDMS/DB database the last record retrieved by a BMPGET command. Chain address fields (pointers) are not updated or written back to the database.
CHASE	The transparency supports this command unconditionally. Programs that request only the CHASE function need not be modified before interfacing with the bridge, and should be linked with the CB\$CH macro as indicated in IBM DBOMP documentation.
BMPOFFAD	The transparency does not support this command. If a call to BMPOFFAD is encountered by the bridge, no action takes place and control returns to the calling program.
BMPFO	The transparency does not support this command. If a BMPFO statement is encountered, an unresolved external reference results in the link edit map.
BMPRACN	The transparency does not support this command. If a BMPRACN command is encountered, no action takes place and control returns to the calling program. The transparency's maintenance of RACN in COBOL programs is the same as for Assembler programs.

IMBSCOBOL Interface Macro

The IMBSCOBOL interface macro replaces the DBOMP CB\$BM macro. This Assembler macro generates tables containing the information necessary to establish communication between the DBOMP COBOL program and IMBSBRDG. Also incorporated in these tables is the information required to support sequential processing requested by calls to BMPGET, BMPPUT, BMPSTKY, and BMPSTDA.

Syntax

IMBSCOBOL macro

▶— IMBSCOBL (*file-name*, index-set-name
NOTSEQ) —▶

Parameters

IMBSCOBOL

A required constant that identifies the macro; it can be coded anywhere after column 1.

file-name

The seven-character name of the DBOMP master file as specified in the program work area. You must enter the routines named in the BMPEOF command in the same order as you enter the corresponding file names in the IMBSCOBOL macro. This ensures that the address of the proper routine is passed to IMBSBRDG when the end of a file named in a BMPGET command is reached. One *file-name* entry must be present for every DBOMP file that is processed.

index-set-name/NOTSEQ

The name of the index set to be used for logical sequential processing; specify NOTSEQ if the file is not to be processed in logical sequential order. One *index-set-name/NOTSEQ* entry must be present for each *file-name* entry.

Note: It is recommended that you name every file on the DBOMP database in one execution of the IMBSCOBOL macro so that this macro does not need to be assembled and link edited more than once.

Assembling and Linking IMBSCOBOL

To assemble and link-edit IMBSCOBOL, you must use SMP/E (Z/OS) or MSHP (Z/VSE).

Note: For more information on using SMP/E and MSHP, see the *CA IDMS Installation—Z/OS* or the *CA IDMS Installation—Z/VSE*.

DBOMP COBOL Program Preparation and Execution

The guidelines for preparing and executing a DBOMP COBOL program using the transparency parallel those detailed for DBOMP Assembler programs in [Using the Transparency as a Bridge to CA IDMS/DB](#) (see page 75).

Preparing the COBOL Program

- Remove the CB\$BM macro.
- Remove DBOMP COBOL commands that are not supported by CA IDMS DBOMP Transparency, and modify associated program logic as necessary.
- Modify the COBOL logic as necessary to conform with CA IDMS DBOMP Transparency specifications for sequential processing and RACN processing.
- If the program issues any of the allowable CA IDMS DML statements, insert the following call to IMBSBRDG, making sure that the CA IDMS DML statement argument is available in working storage (see [The Transparency Environment](#) (see page 21)):

`call 'bmpcall' using argument-name.`
- If any retrieval or update process indicators except for those supported by CA IDMS DBOMP Transparency (see [The Transparency Environment](#) (see page 21)) are used in the program, replace them with those that are supported.

Executing the Program

- Assemble and link edit IMBSTAB if a version compatible with the application does not exist in the load library.
- Recompile and link edit the DBOMP COBOL program, including IMBSBRDG, IMBSTAB, IMBSCOB, and CA IDMS/DB. This step assumes that IMBSCOB has been assembled and link edited as discussed above.

Note: You do not need to recompile programs that run under Z/OS unless any of the changes listed above have been made; you must, however, recompile programs that run under Z/VSE whether or not any of these changes have been made, unless the programs exist in the relocatable library.

- Submit the DBOMP COBOL program for execution.

Appendix C: Sample Application and Procedures

This section contains the following topics:

[Overview](#) (see page 97)

[IMBSBILL Sample Application](#) (see page 97)

[IMBSMJ01 Sample JCL for z/OS](#) (see page 99)

[IMBSMJ02 Sample JCL for z/OS](#) (see page 100)

Overview

This appendix contains the following sample application and JCL for z/OS:

- **IMBSBILL sample application**—Illustrates the sequence and structure of database access procedures necessary to perform standard bill-of-materials functions against a CA IDMS/DB manufacturing database. IMBSBILL is written in ANS COBOL and issues CA IDMS/DB COBOL Data Manipulation Language statements requesting database services.
- **IMBSMJ01 sample JCL for z/OS**—IMBSMJ01 is a collection of EXEC statements which you can use as a reference when you convert a DBOMP database to a CA IDMS/DB database.
- **IMBSMJ02 sample JCL for z/OS**—IMBSMJ02 is a collection of EXEC statements which you can use as a reference when you execute DBOMP applications using the transparency.

IMBSBILL Sample Application

IMBSBILL Functions

IMBSBILL serves two purposes:

- To aid in the conversion of DBOMP load, maintenance, and retrieval/update programs to CA IDMS/DB
- To serve as a prototype for the development of systems oriented to the manufacturing environment

Record Types Referenced by IMBSBILL

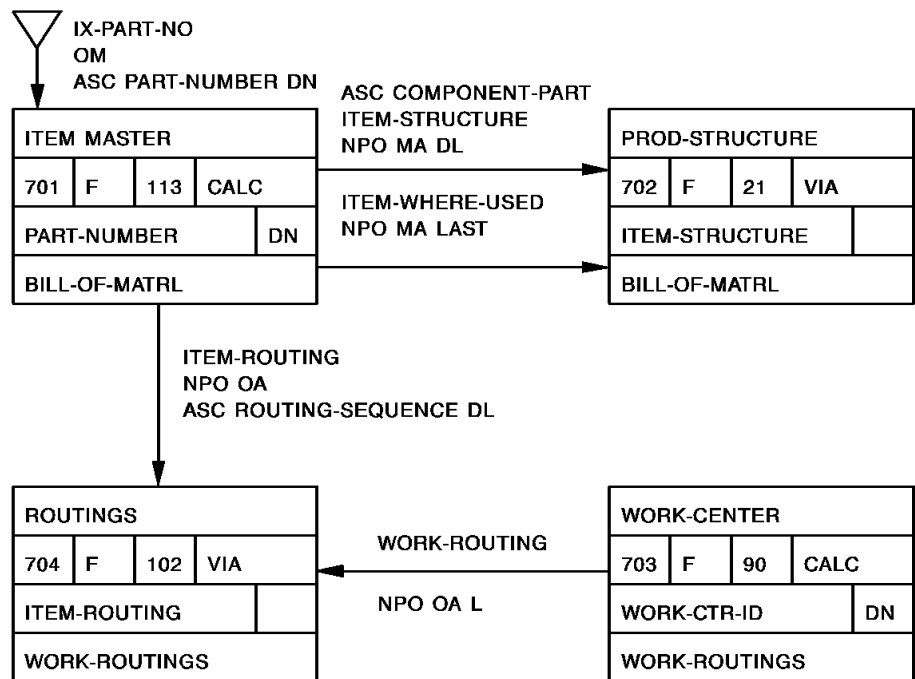
IMBSBILL references these CA IDMS/DB record types:

- ITEM-MASTER
- PROD-STRUCTURE
- WORK-CENTER
- ROUTINGS

IMBSBILL retrieves, modifies, adds, and deletes occurrences of each of these record types. It demonstrates single-level, indented, and summarized explosion and implosion, and performs a serial retrieval of occurrences of the ITEM-MASTER record type. IMBSBILL also contains the CA IDMS/DB logic necessary to implement RACN, low-level coding, and chain counts.

Database Accessed by IMBSBILL

The design for the sample database accessed by IMBSBILL is shown in the following figure.



IMBSBILL Flow of Logic

The general flow of logic in IMBSBILL is as follows:

1. BIND the run unit and all record types
2. Read a transaction
3. Branch to the routine indicated by the transaction code
4. Access the CA IDMS/DB database using the appropriate DML commands
5. Display the results of the transaction on the printer
6. Repeat the above steps until all transactions have been processed

IMBSBILL Code

The following is the code for IMBSBILL.

- IMBSBILL - The program described here
- IMBSCHM - The schema IMBSBILL uses
- IMBDMCL - The DMC IMBSBILL uses
- IMBSUBS - The subschema IMBSBILL uses

IMBSMJ01 Sample JCL for z/OS

Explanation of Statements in IMBSMJ01

Each EXEC statement in IMBSMJ01 is a job step. The steps are described in the following table.

EXEC statement	What happens
IDMSCHEM	Compiles the sample schema, IMBSSCHM
IDMSDMCL	Processes the sample DMCL module, IMBDMCL
LINKDMCL	Link edits the assembled output from the DMCL processor
SUBSCHEM	Compiles the sample subschema, IMBSSUBS, and punches load module
LINKSUB	Link edits IMBSSUBS
DMLC	Submits the sample COBOL source program, IMBSBILL, to the CA IDMS DML compiler
COB	Compiles the output from DMLC

EXEC statement	What happens
LINKCOB	Link edits the compiled COBOL program
IDMSRPTS	Prints reports from the data dictionary
INITSAMP	Initializes the sample database
EXECPGM	Executes the sample CA IDMS/DB application program, IMBSBILL

Note: Be sure to modify the parameters in the EXECPGM step to suit your installation requirements.

IMBSMJ02 Sample JCL for z/OS

Explanation of Statements in IMBSMJ02

Each EXEC statement in IMBSMJ02 is a job step. The steps are described in the following table.

EXEC statement	What happens
ASMCBDG	Assembles IMBSTAB
LINKCBDG	Link edits IMBSTAB
ASMCobl	Assembles IMBSCobl interface
LINKCOBL	Link edits IMBSCobl module
DMLC	Submits sample COBOL DBOMP source program, IMBSDBMP, to the CA IDMS DML compiler
COB	Compiles output from DMLC
LINKCOB	Link edits IMBSDBMP
EXECPGM	Executes the sample DBOMP application program, IMBSDBMP, using the CA IDMS/DBOMP Transparency bridge program IMBSBRDG

Note: Be sure to modify the parameters in the EXECPGM step to suit your installation requirements.

Appendix D: Setting Up CA IDMS/DBOMP Transparency Under z/OS

This section contains the following topics:

[Overview](#) (see page 101)

[Customizing and Executing IMBSMJ01 and IMBSMJ02](#) (see page 102)

Overview

Object Modules

The following table lists the object modules placed into the CA IDMS/DB object library during the install.

Module	Description
IMBSPROC	Database procedure
IMBSBRDG	Bridge program

Source Modules

The following table lists the source modules placed into the CA IDMS/DB source library during the install.

Module	Description
IMBSMJ02	JCL for IMBSMJ02 procedure
IMBS	Customizing macro
IMBSASMB	IMBS Assembler interface macro
IMBSBILL	Sample CA IDMS/DB COBOL manufacturing application program
IMBSBRDG	Assembler source code for IMBSBRDG object module
IMBSCOBL	CA IDMS/DBOMP Transparency COBOL interface macro
IMBSDBMP	Sample COBOL DBOMP program (to be bridged)
IMBSDMCL	Sample DMCL description module

Module	Description
IMBSEQ	CA IDMS/DBOMP Transparency Assembler interface macro
IMBSINP1	Sample input to IMBSBILL
IMBSINP2	Sample input to IMBSDBMP
IMBSPL1	CA IDMS/DBOMP Transparency PL/I interface macro
IMBSPROC	Source code for database procedure object module
IMBSMJ01	JCL for IMBSMJ01 procedure
IMBSSCHM	Sample CA IDMS/DB schema description
IMBSSUBS	Sample CA IDMS/DB subschema description
IMBSTAB	Sample input to IMBS customizing macro

Load Modules

The following table lists the load modules placed in the CA IDMS/DB load library during the install.

Module	Description
IMBSBRDG	Bridge program
IMBSPROC	Database procedure

Customizing and Executing IMBSMJ01 and IMBSMJ02

The JCL is shown in the IMBSMJ01 and IMBSMJ02 procedures as they exist in the source library.

Source library member IMBSMJ01 contains a procedure that compiles the schema, DMCL, and subschema for the sample database. It then initializes the database and runs the sample DML program, IMBSBILL.

Member IMBSMJ02 compiles a sample DBOMP program, IMBSDBMP, and CA IDMS/DBOMP Transparency, which uses the same database as was set up by IMBSMJ01.

Explanation of EXEC Statements in IMBSMJ01 Procedure

The IMBSMJ01 procedure uses the 15 EXEC statements described in the following table.

EXEC statement	What happens
IDMSCHEM	Compiles the sample schema, IMBSSCHM
IDMSDMCL	Processes the sample DMCL module, IMBSDMCL and punches the load module
LINKDMCL	Link edits the assembled output from the DMCL processor
SUBSCHEM	Compiles the sample subschema, IMBSSUBS
LINKSUB	Link edits IMBSSUBS
DMLC	Submits the sample COBOL source program, IMBSBILL, to the CA IDMS DML compiler
COB	Compiles the output from DMLC
LINKCOB	Link edits the compiled COBOL program
IDMSRPTS	Prints reports from the data dictionary
INITSAMP	Initializes the sample database
EXECPGM	Executes the sample CA IDMS/DB application program, IMBSBILL

Note: You must modify the parameters in the EXEC IMBSMJ01 statement (the last EXEC statement in the procedure) to suit your installation requirements. For more information, see [Customizing IMBSMJ01](#) (see page 103).

Customizing IMBSMJ01

You must modify the defaults shown in the EXEC IMBSMJ01 statement (the last JCL statement) in the IMBSMJ01 procedure. The following JCL shows the exec IMBSMJ01. Change the items shown in italics to suit your installation requirements.

IMBSMJ01 (z/OS)

```
//SAMPLE EXEC IMBSMJ01
//          PRT='SYSOUT=A',
//          UNIT=disk,
//          LIB='imbs.loadlib',
//          IDMSLIB='idms.loadlib',
//          COBLIB='coblib',
//          COBSTEP='cob.steplib',
//          PGSIZE=2496,
//          DISP=CATLG,
//          BASE='data.direct',
//          IMBSBILL='imbs013',
//          IMBSWORK='IMBSWORK',
//          SRCLIB='imbs.srclib',
//          IDMSSRC='yourHLQ.CAGJMAC',
//          VOL='VOL=SER=nnnnnn',
//          SYSCTLDS='idms.sysctl',
//          IDMSMCL='cvdmc1',
//          MSGDD='dcmsg',
//          MSGDSN='idms.ddldcmsg',
//          DDLDD='sysddl',
//          DDLDSN='idms.sysddl',
//          DICTNAME='apldict',
```

Parameter	Description
disk	Symbolic device name for data dictionary and database files
imbs.loadlib	Dataset name of CA IDMS/DBOMP Transparency load library if a load library was allocated in optional ALLOC step of INSTALL procedure; or dataset name of CA IDMS load library if CA IDMS DBOMP Transparency load library was not allocated
idms.loadlib	Dataset name of CA IDMS load library
coblib	Dataset name of COBOL library
cob.steplib	Dataset name of COBOL step library
data.direct	Dataset name of data dictionary; may be a sample or user directory
IMBSBILL	Dataset name of sample CA IDMS database file
IMBSWORK	Dataset name of sample CA IDMS database file

Parameter	Description
imbs.srclib	Dataset name of CA IDMS DBOMP Transparency source library if a source library was allocated in optional ALLOC step of INSTALL procedure; or dataset name of CA IDMS source library if CA IDMS DBOMP Transparency source library was not allocated
yourHLQ.CAGJMAC	Dataset name of CA IDMS macro library
nnnnnn	Volume serial number of disk where data dictionary and sample CA IDMS database files are stored
idms.sysctl	Dataset name of IDMS SYSCTL file for running CV
cvdml	Name of the DMCL that IDMS uses, for CV or local
dcmsg	The ddname or IDMS message area
idms.ddldcmsg	Dataset name of the IDMS message area, for CV and local jobs
ddldd	The ddname of the IDMS dictionary
idms.sysddl	Dataset name of the IDMS dictionary
apldict	Dictionary to be used

Explanation of EXEC Statements in IMBSMJ02 Procedure

The IMBSMJ02 procedure uses the eight EXEC statements described in the following table.

EXEC statement	What happens
ASMCBDG	Assembles IMBSTAB
LINKCBDG	Link edits IMBSTAB
ASMCobl	Assembles IMBSCobl interface
LINKCOBL	Link edits IMBSCobl module
DMLC	Submits sample COBOL DBOMP source program, IMBSDBMP, to the CA IDMS DML compiler
COB	Compiles output from DMLC
LINKCOB	Link edits IMBSDBMP
EXECPGM	Executes the sample DBOMP application program, IMBSDBMP, using the CA IDMS/DBOMP Transparency bridge program IMBSBRDG

Note: You must modify the parameters in the EXEC IMBSMJ02 statement (the last EXEC statement in the procedure) to suit your installation requirements. For more information, see [Customizing IMBSMJ02](#) (see page 106).

Customizing IMBSMJ02

You must modify the defaults shown in the EXEC IMBSMJ02 statement (the last JCL statement) in the IMBSMJ02 procedure. The following JCL shows the exec IMBSMJ02. Change the items shown in italics to suit your installation requirements.

IMBSMJ02 (z/OS)

```
//SAMPLE EXEC IMBSMJ02
//          PRT='SYSOUT=A',
//          UNIT=disk,
//          LIB='imbs.loadlib',
//          IDMSLIB='idms.loadlib',
//          IDMSsrc='yourHLQ.CAGJMAC',
//          COBLIB='coblib',
//          COBSTEP='cob.steplib',
//          BASE='data.direct',
//          IMBSBILL='imbs013',
//          IMBSWORK='IMBSWORK',
//          SRCLIB='imbs.srclib',
```

Parameter	Description
disk	Symbolic device name for data dictionary and database files
imbs.loadlib	Dataset name of CA IDMS/DBOMP Transparency load library if a load library was allocated in optional ALLOC step of INSTALL procedure; or dataset name of CA IDMS load library if CA IDMS DBOMP Transparency load library was not allocated
idms.loadlib	Dataset name of CA IDMS load library
yourHLQ.CAGJMAC	Dataset name of CA IDMS macro library
coblib	Dataset name of COBOL library
cob.steplib	Dataset name of COBOL step library
data.direct	Dataset name of data dictionary; may be a sample or user directory
IMBSBILL	Dataset name of sample CA IDMS database file
IMBSWORK	Dataset name of sample CA IDMS database file

Parameter	Description
imbs.srclib	Dataset name of CA IDMS DBOMP Transparency source library if a source library was allocated in optional ALLOC step of INSTALL procedure; or dataset name of CA IDMS source library if CA IDMS/DB source library was not allocated

Executing IMBSMJ01 and IMBSMJ02

After you tailor the IMBSMJ01 and IMBSMJ02 procedures to your installation requirements, you can submit them together as a job.

Appendix E: Setting Up CA IDMS DBOMP Transparency under Z/VSE

This section contains the following topics:

[Customizing and Executing IMBSVJ01 and IMBSVJ02](#) (see page 109)

[Running IMBSVJ01](#) (see page 111)

[Running IMBSVJ02](#) (see page 111)

Customizing and Executing IMBSVJ01 and IMBSVJ02

The JCL is shown in the IMBSVJ01 and IMBSVJ02 procedures as they exist in the source library.

Source library member IMBSVJ01 contains a procedure that compiles the schema, DMCL, and subschema for the sample database. It then initializes the database and runs the sample DML program, IMBSBILL.

Member IMBSVJ02 compiles a sample DBOMP program, IMBSDBMP, and the components needed to run it through CA IDMS DBOMP Transparency, which uses the same database as was set up by IMBSVJ01.

Explanation of EXEC Statements in IMBSVJ01 Procedure

The IMBSVJ01 procedure uses the EXEC statements described in the following table.

EXEC statement	What happens
IDMSCHEM	Compiles the sample schema, IMBSSCHM
IDMSDMCL	Compiles the sample and punches DMCL module, IMBSDMCL
LNKEDT	Link edits sample DMCL module, IMBSDMCL
IDMSUBSC	Compiles the sample and punches subschema, IMBSSUBS
ASSEMBLY	Assembles IMBSSUBS
LNKEDT	Link edits IMBSSUBS
IDMSDMLC	Submits the sample COBOL program, IMBSBILL, to the CA IDMS Data Manipulation Language compiler
FCOBOL	Submits IMBSBILL to the COBOL compiler

EXEC statement	What happens
LNKEDT	Link edits IMBSBILL
IDMSRPTS	Prints all dictionary/directory reports
IDMSBCF	Initializes the sample database
IMBSBILL	Executes the sample program, IMBSBILL

Explanation of EXEC Statements in IMBSVJ02 Procedure

The IMBSVJ02 procedure uses the eight EXEC statements described in the following table.

EXEC statement	What happens
ASSEMBLY	Assembles the IMBS customizing macro
MAINT	Catalogs IMBSTAB to relocatable library
ASSEMBLY	Assembles the IMBSCOBOL macro
MAINT	Catalogs assembled IMBSCOBOL to relocatable library
IDMSDMLC	Submits the sample COBOL DBOMP program, IMBSDBMP, to the Data Manipulation Language compiler
FCOBOL	Submits IMBSDBMP to the COBOL compiler
LNKEDT	Link edits IMBSDBMP
DEMOPROG	Executes the sample DBOMP program, IMBSDBMP, against CA IDMS DBOMP Transparency

Modules placed in the relocatable library

The following table lists the modules placed in the relocatable library during installation.

Module	Description
IMBSBRDG	Bridge program
IMBSPROC	Database procedure

Modules placed in the source statement library

The following table lists the modules placed in the source statement library during installation.

Module	Description
IMBS	Customizing macro
IMBSASMB	Interface module (Assembler)
IMBSBILL	Sample CA IDMS/DB COBOL manufacturing application program
IMBSBRDG	Assembler source code for IMBSBRDG object module
IMBSCOBOL	CA IDMS DBOMP Transparency COBOL interface macro (Assembler)
IMBSDBMP	Sample COBOL DBOMP program to be bridged
IMBSDMCL	Sample DMCL description module
IMBSEQ	Interface module (Assembler)
IMBSINP1	Sample input to IMBSBILL
IMBSINP2	Sample input to IMBSDBMP
IMBSPL1	CA IDMS DBOMP Transparency interface macro (PL/I)
IMBSPROC	Source code for database procedure object module
IMBSSCHM	Sample CA IDMS/DB schema description
IMBSSUBS	Sample CA IDMS/DB subschema description
IMBSTAB	Sample input to IMBS customizing macro

Running IMBSVJ01

Run IMBSVJ01, which executes a CA IDMS/DB manufacturing application, using test data provided on the installation media and cataloged in the source statement library.

Running IMBSVJ02

Run IMBSVJ02, which executes a DBOMP program with the CA IDMS DBOMP Transparency bridge, using test data provided on the installation media and cataloged in the source statement library.

The JCL in IMBSVJ01 and IMBSVJ02 must first be edited so that the dataset names are correct for your site.

Index

C

- CA IDMS DBOMP Transparency Transparency, using,
 - 83, 87
 - IMBSPL1 • 87
- CA IDMS DBOMP Transparency, using, • 75, 76, 81, 83
 - DBOMP Assembler program, executing, • 76
 - DBOMP Assembler program, preparing, • 75
 - debugging process, values • 83
 - debugging techniques, • 81
 - errors, diagnosing in bridged programs, • 81
 - JCL, • 76
- CA IDMS/DB, system conversion to, • 60, 62, 64, 69, 72
 - CA IDMS/DB Communications Block, • 62
 - data conversion • 60
 - DBOMP commands, with equivalent CA IDMS/DB DML statements, • 69
 - DBOMP load and maintenance program conversion, • 72
 - DBOMP process indicators, with equivalent CA IDMS/DB DML statements • 64
 - retrieval and update program conversion, • 72
- CA IDMS/DB., system conversion to, • 59

D

- DBOMP macros, • 23, 24
 - BM\$DS • 24
 - BM\$WA • 24
 - EQ\$RG • 24
 - MGS • 24
 - MO\$VE • 24
 - not supported by the transparency • 23
 - processed independently of the transparency • 24
 - TY\$PE • 24
- DBOMP process indicators • 24, 25, 26, 27, 29
 - @ACCEPT • 26
 - @BIND PROC • 26
 - @COMMIT • 26
 - @ROLLBAK • 26
 - arguments, building • 27
 - arguments, table of • 27
 - CDIR • 24

- CLOS • 25
- CMPR • 24
- CRDR • 24
- CUPD • 24
- EXPN • 25
- MDIR • 24
- MRAN • 24
- MRDR • 24
- MRKY • 24
- MUPD • 24
- not supported by the transparency • 25
- OPEN • 25
- supported by the transparency • 24, 25

I

- IDMS, system conversion to • 59
- IMBS, user-coded customizing macro, • 30, 31, 33, 35, 39, 40
 - control statement • 31
 - delimiter statement, • 35
 - file/record type description statement, • 33
 - IMBS input statements • 31
 - IMBSTAB, error messages • 39
 - MNOTES, IMBSTAB • 39
 - pointer/set relationship statement, • 35
 - set identification statement, • 33
- IMBSBRDG program module, • 15, 17, 48, 49, 51, 55
 - CALC, • 17
 - command conversion, • 49
 - DIRECT, • 17
 - moving data, • 51
 - record conversion, • 51
 - retrieval processing, • 15
 - update processing, • 15
 - values, returned to the calling program, • 51
 - VIA, • 17
- IMBSEQ macro, • 55
 - sequential file table, • 55
- IMBSTAB, customized bridge module • 36
 - assembly and linkage, • 36
- IMBSTAB, customized bridge module, • 40, 46, 76, 81
 - dummy record, CA IDMS/DB, • 46
 - integration into bridge program, • 46
 - JCL, • 76

INSTALL procedure • 102
introducing COBOL considerations, • 91, 93
IMBSCOBOL • 93

P

programming restrictions • 17, 18
Assembler macros • 17
prototype bill-of-materials application, • 97, 99
IMBSBILL, logic flow • 97
sample program, logic flow • 97

R

Run Activity Control Number • 21, 22
CA\$LL • 22
CHA\$E • 22
GE\$T • 22
PU\$T • 22
requiring program modification and reassembly,
• 22
ST\$KY • 22
supported unconditionally by the transparency •
22

S

syntax, • 31, 33, 35, 55
IMBS macro control statement, • 31
IMBS macro delimiter statement, • 35
IMBS macro file/record type description
statement, • 33
IMBS macro pointer/set relationship statement,
• 35
IMBS macro set identification statement, • 33
IMBSEQ macro, • 55

V

VSE/ESA Setting Up, • 109, 111
IMBSVJ01, • 111
IMBSVJ02, • 111

Z

z/OS and OS/390 Setting Up • 101, 102, 103, 106,
107
IMBSMJ01 procedure • 102
IMBSMJ02 procedure • 102
load modules, in CA IDMS/DB load library • 101
modifications, in IMBSMJ01 procedure, • 103
modifications, in IMBSMJ02 procedure, • 106

object modules, in CA IDMS/DB or object library
• 101

Setting Up procedure, • 101

source modules, in CA IDMS/DB source library •
101