CA IDMS - 19.0
Using Transparency for DBOMP

Date: 15-Jan-2018
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 © 2017 CA. All rights reserved. All trademarks, trade names, service marks, and logos referenced herein belong to their respective companies.
Table of Contents

Functions and Modules .............................................................................................................. 8
Functions ................................................................................................................................. 8
Modules ................................................................................................................................. 9

Data Description Guidelines ................................................................................................. 10

Programming Requirements ................................................................................................. 10

Installation .............................................................................................................................. 11

The Transparency Environment ............................................................................................. 12
DBOMP Macros Supported ....................................................................................................... 13
  Macros Supported Unconditionally ......................................................................................... 13
  Macros That Require Program Modification and Reassembly ............................................ 13
  Macros Not Supported ......................................................................................................... 14
  Macros Processed Independently of the Transparency ......................................................... 14
DBOMP Process Indicators Supported .................................................................................... 15
  Process Indicators Fully Supported .................................................................................... 15
  Process Indicators Supported with Exceptions .................................................................... 16
  Process Indicators Not Supported ...................................................................................... 16
DBOMP Routines Supported .................................................................................................. 17
CA IDMS DML Statements Supported in Bridged Programs ................................................ 17
How to Include CA IDMS DML Statements .......................................................................... 18

Using the Transparency as a Bridge to CA IDMS/DB .......................................................... 19
Preparing DBOMP Assembler Programs ............................................................................... 19
Executing DBOMP Assembler Programs ............................................................................... 20
  Assembling and Executing Under z/OS .............................................................................. 21
  Assembling and Executing Under Z/VSE ........................................................................... 23
Diagnosing Errors .................................................................................................................. 24
<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>What to Look For When Errors Occur During Program Processing</td>
<td>24</td>
</tr>
<tr>
<td>What to Look For When Inaccurate Data is Returned</td>
<td>26</td>
</tr>
<tr>
<td>Where to Find Values During Debugging</td>
<td>26</td>
</tr>
<tr>
<td>Transparency Programs and Macros</td>
<td>26</td>
</tr>
<tr>
<td>IMBS Customizing Macro</td>
<td>27</td>
</tr>
<tr>
<td>Control Statement</td>
<td>27</td>
</tr>
<tr>
<td>Set Identification Statement</td>
<td>29</td>
</tr>
<tr>
<td>File/Record Type Description Statement</td>
<td>30</td>
</tr>
<tr>
<td>Pointer/Set Relationship Statement</td>
<td>31</td>
</tr>
<tr>
<td>Delimiter Statement</td>
<td>31</td>
</tr>
<tr>
<td>Output From IMBS Macro -- IMBSTAB</td>
<td>31</td>
</tr>
<tr>
<td>IMBSTAB Error Messages</td>
<td>35</td>
</tr>
<tr>
<td>Sample IMBS and IMBSTAB</td>
<td>36</td>
</tr>
<tr>
<td>IMBSPROC Database Procedure</td>
<td>40</td>
</tr>
<tr>
<td>IMBSBRDG program module</td>
<td>41</td>
</tr>
<tr>
<td>Converting DBOMP Calls to CA IDMS/DB Statements</td>
<td>42</td>
</tr>
<tr>
<td>Converting Records Retrieved from CA IDMS/DB</td>
<td>43</td>
</tr>
<tr>
<td>IMBSEQ macro</td>
<td>46</td>
</tr>
<tr>
<td>Converting DBOMP to CA IDMS/DB</td>
<td>48</td>
</tr>
<tr>
<td>Converting Data</td>
<td>49</td>
</tr>
<tr>
<td>Converting DBOMP Load and Maintenance Programs</td>
<td>51</td>
</tr>
<tr>
<td>DBOMP Process Indicators and Corresponding DML</td>
<td>53</td>
</tr>
<tr>
<td>DBOMP Commands and Corresponding DML</td>
<td>57</td>
</tr>
<tr>
<td>Sequence of Logic in Converted Programs</td>
<td>59</td>
</tr>
<tr>
<td>Converting DBOMP Retrieval and Update Programs</td>
<td>60</td>
</tr>
<tr>
<td>DBOMP Error Codes With CA IDMS/DB Equivalents</td>
<td>60</td>
</tr>
<tr>
<td>PL/I Considerations</td>
<td>61</td>
</tr>
<tr>
<td>Transparency Support For DBOMP PL/I Commands</td>
<td>61</td>
</tr>
<tr>
<td>IMBSPL1 Interface Macro</td>
<td>63</td>
</tr>
<tr>
<td>DBOMP PL/I Program Preparation and Execution</td>
<td>64</td>
</tr>
<tr>
<td>COBOL Considerations</td>
<td>65</td>
</tr>
<tr>
<td>Transparency Support For DBOMP COBOL Commands</td>
<td>65</td>
</tr>
<tr>
<td>Section</td>
<td>Page</td>
</tr>
<tr>
<td>-------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>IMBCOBL Interface Macro</td>
<td>67</td>
</tr>
<tr>
<td>DBOMP COBOL Program Preparation and Execution</td>
<td>68</td>
</tr>
<tr>
<td>Sample Application and Procedures</td>
<td>69</td>
</tr>
<tr>
<td>IMBSBILL Sample Application</td>
<td>69</td>
</tr>
<tr>
<td>IMBSMJ01 Sample JCL for z/OS</td>
<td>71</td>
</tr>
<tr>
<td>IMBSMJ02 Sample JCL for z/OS</td>
<td>71</td>
</tr>
<tr>
<td>Setting Up CA IDMS/DBOMP Transparency Under z/OS</td>
<td>72</td>
</tr>
<tr>
<td>Customizing and Executing IMBSMJ01 and IMBSMJ02</td>
<td>73</td>
</tr>
<tr>
<td>Explanation of EXEC Statements in IMBSMJ01 Procedure</td>
<td>74</td>
</tr>
<tr>
<td>Customizing IMBSMJ01</td>
<td>74</td>
</tr>
<tr>
<td>IMBSMJ01 (z/OS)</td>
<td>74</td>
</tr>
<tr>
<td>Explanation of EXEC Statements in IMBSMJ02 Procedure</td>
<td>76</td>
</tr>
<tr>
<td>Customizing IMBSMJ02</td>
<td>76</td>
</tr>
<tr>
<td>Executing IMBSMJ01 and IMBSMJ02</td>
<td>77</td>
</tr>
<tr>
<td>Setting Up CA IDMS DBOMP Transparency under Z/VSE</td>
<td>77</td>
</tr>
<tr>
<td>Customizing and Executing IMBSVJ01 and IMBSVJ02</td>
<td>77</td>
</tr>
<tr>
<td>Explanation of EXEC Statements in IMBSVJ01 Procedure</td>
<td>78</td>
</tr>
<tr>
<td>Explanation of EXEC Statements in IMBSVJ02 Procedure</td>
<td>78</td>
</tr>
<tr>
<td>Running IMBSVJ01</td>
<td>79</td>
</tr>
<tr>
<td>Running IMBSVJ02</td>
<td>79</td>
</tr>
</tbody>
</table>
Using Transparency for DBOMP

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 69).

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.

This section 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

The remainder of this section discusses the following topics:
Functions and Modules

Contents
- Functions (see page 8)
- Modules (see page 9)

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.

![Diagram of 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 26).

**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.

<table>
<thead>
<tr>
<th>Simulation of</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Retrieval processing</strong></td>
<td>Accepts a DBOMP call to entry point BM$PIO or AP$SEQ&lt;br&gt;Converts the process indicator to a CA IDMS/DB call&lt;br&gt;Retrieves the requested record from the CA IDMS/DB database&lt;br&gt;Converts the retrieved CA IDMS/DB record to a DBOMP record&lt;br&gt;Returns the converted record to the calling program&lt;br&gt;Converts the CA IDMS/DB error status to the appropriate DBOMP error code&lt;br&gt;Updates the work area prefix&lt;br&gt;Returns control to the calling program</td>
</tr>
<tr>
<td><strong>Update processing</strong></td>
<td>Accepts a DBOMP call to entry point BM$PIO or AP$SEQ&lt;br&gt;Validates the DBOMP file name and process indicator&lt;br&gt;Converts the process indicator to a CA IDMS/DB call</td>
</tr>
</tbody>
</table>
### Simulation of:

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reconstructs a CA IDMS/DB record from the updated DBOMP record</td>
</tr>
<tr>
<td>Returns the reconstructed record to the CA IDMS/DB database</td>
</tr>
<tr>
<td>Converts the CA IDMS/DB error status to the appropriate DBOMP error code</td>
</tr>
<tr>
<td>Updates the work area prefix</td>
</tr>
<tr>
<td>Returns control to the calling program</td>
</tr>
</tbody>
</table>

### 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 sectionlines 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
For PL/I equivalents of these macros, see PL/I Considerations (see page 61). 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 26).

- **COBOL Considerations (see page 65)** Use an index for the logical sequential ordering of master records.

  ![Note:](image)

  For more information on indexing, see the CA IDMS Database Administration Section.

- 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 18).

- 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 51).

## Installation

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

![Note:](image)

For more information on installation, see the CA IDMS Installation Section 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.

<table>
<thead>
<tr>
<th>Module</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IMBSBRDG</td>
<td>Bridge program</td>
</tr>
<tr>
<td>IMBSPROC</td>
<td>Database procedure</td>
</tr>
</tbody>
</table>
Source Modules Placed During Installation

Modules listed in the following table exist as source only.

<table>
<thead>
<tr>
<th>Module</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BRDGSAMP</td>
<td>Z/OS JCL for BRDGSAMP procedure</td>
</tr>
<tr>
<td></td>
<td>(for more information, see Sample Application and Procedures (see page 69))</td>
</tr>
<tr>
<td>IMBS</td>
<td>Customizing macro</td>
</tr>
<tr>
<td>IMBSBILL</td>
<td>Sample CA IDMS/DB COBOL manufacturing application program</td>
</tr>
<tr>
<td>IMBSBRDG</td>
<td>Assembler source code for IMBSBRDG object module</td>
</tr>
<tr>
<td>IMBSCOBL</td>
<td>IMBS COBOL interface macro</td>
</tr>
<tr>
<td>IMBSDBMP</td>
<td>Sample COBOL DBOMP program (to be bridged)</td>
</tr>
<tr>
<td>IMBSDMCL</td>
<td>Sample DMCL description module</td>
</tr>
<tr>
<td>IMBSINP1</td>
<td>Sample input to IMBSBILL</td>
</tr>
<tr>
<td>IMBSINP2</td>
<td>Sample input to IMBSDBMP</td>
</tr>
<tr>
<td>IMBSPL1</td>
<td>CA IDMS DBOMP Transparency PL/I interface macro</td>
</tr>
<tr>
<td>IMBSPROC</td>
<td>Source code for database procedure object module</td>
</tr>
<tr>
<td>IMBSSAMP</td>
<td>Z/OS JCL for IMBSSAMP procedure</td>
</tr>
<tr>
<td></td>
<td>(for more information, see Sample Application and Procedures (see page 69))</td>
</tr>
<tr>
<td>IMBSSCHM</td>
<td>Sample CA IDMS/DB schema description</td>
</tr>
<tr>
<td>IMBSSUBS</td>
<td>Sample CA IDMS/DB subschema description</td>
</tr>
<tr>
<td>IMBSTAB</td>
<td>Sample input to IMBS customizing macro</td>
</tr>
</tbody>
</table>

The Transparency Environment

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 section discusses support for the following entities in the transparency environment:

- DBOMP Macros Supported (see page 13)
- DBOMP Process Indicators Supported (see page 15)
- DBOMP Routines Supported (see page 17)
- CA IDMS DML Statements Supported in Bridged Programs (see page 17)
- How to Include CA IDMS DML Statements (see page 18)
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 (see page 13)
- Macros That Require Program Modification and Reassembly (see page 13)
- Macros Not Supported (see page 14)
- Macros Processed Independently of the Transparency (see page 14)

For more information on PL/I equivalent macros, see PL/I Considerations (see page 61). For more information on COBOL equivalent macros, see COBOL Considerations (see page 65).

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 26)):

- 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.

<table>
<thead>
<tr>
<th>Remove this macro</th>
<th>Replace it with</th>
</tr>
</thead>
<tbody>
<tr>
<td>MF$SQ</td>
<td>IMBSEQ</td>
</tr>
<tr>
<td>FI$LE</td>
<td>IMBSEQ</td>
</tr>
<tr>
<td>CF$RT</td>
<td>IMBSEQ</td>
</tr>
<tr>
<td>CGE$T</td>
<td>GE$T</td>
</tr>
<tr>
<td>CPU$T</td>
<td>PU$T</td>
</tr>
</tbody>
</table>

### 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.

The transparency does not support any DBOMP process indicators that request structural maintenance functions. For more information, see the following topics:

- **Process Indicators Fully Supported** (see page 15)
- **Process Indicators Supported with Exceptions** (see page 16)
- **Process Indicators Not Supported** (see page 16)

**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 on the transparency’s support of RACN, see DBOMP Routines Supported (see page 17)).

- **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.

<table>
<thead>
<tr>
<th>Field</th>
<th>Usage</th>
<th>Length</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Chara</td>
<td>8</td>
<td>The literal value of the CA IDMS verb issued by the bridged program. Acceptable values are: @BIND, @COMMIT, @ROLLBAK, @ACCEPT. PL/I or COBOL equivalents are also acceptable.</td>
</tr>
<tr>
<td>2</td>
<td>Chara</td>
<td>8</td>
<td>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.</td>
</tr>
<tr>
<td>3</td>
<td>Chara</td>
<td>1-256</td>
<td>The variable data passed by: @BIND PROC, @ACCEPT PROC, or @ACCEPT STATS. This field is necessary only if one of these DML statements is issued.</td>
</tr>
</tbody>
</table>

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    OD
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$LL BMP$IO, 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.

Using the Transparency as a Bridge to CA IDMS/DB

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 Assembler Programs (see page 19)
- Executing DBOMP Assembler Programs (see page 20)
- Diagnosing Errors (see page 24)

This section explains the procedures you use to prepare and execute Assembler programs and for diagnosing errors in bridged Assembler, see PL/I Considerations [http://wiki-dev.ca.com/pages/viewpage.action?pageId=85197958] and COBOL Considerations [http://wiki-dev.ca.com/display/IDMS/COBOL+Considerations].

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 CA IDMS/DB verbs, insert the proper calls to IMBSBRDG (see The Transparency Environment (see page 12)). 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 12)).

Executing DBOMP Assembler Programs

Contents

- Assembling and Executing Under z/OS (see page 21)
- Assembling and Executing Under Z/VSE (see page 23)

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
//ASM.SYSLIB 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,DNS=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 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 Section*.

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
//sysjrnl      DD DSN=idms.tapejrnl,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. CAGIMAC           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
sysjrnl                   DD name for CA IDMS/DB journal file
idms.tapejrnl             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=dmcl-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=dmcl-name
Other SYSIDMS runtime parameters, as appropriate
program input, as required
/*

Explanation of Variables

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pgmname</td>
<td>Name of DBOMP application program</td>
</tr>
<tr>
<td>imbstab</td>
<td>Dataset name for link edited output from IMBS customizing macro</td>
</tr>
<tr>
<td>sys009</td>
<td>Logical unit assignment for CA IDMS/DB journal file</td>
</tr>
<tr>
<td>281</td>
<td>Physical device assignment for CA IDMS/DB journal file</td>
</tr>
<tr>
<td>sys010</td>
<td>Logical unit assignment for CA IDMS/DB database file</td>
</tr>
<tr>
<td>137</td>
<td>Physical device assignment for CA IDMS/DB database file</td>
</tr>
<tr>
<td>database</td>
<td>Dataset name for CA IDMS/DB database file</td>
</tr>
<tr>
<td>444444</td>
<td>Serial number of disk containing CA IDMS/DB database file</td>
</tr>
<tr>
<td>76</td>
<td>Relative track where CA IDMS/DB database file begins</td>
</tr>
<tr>
<td>1776</td>
<td>Number of tracks used by CA IDMS/DB database file</td>
</tr>
<tr>
<td>dmcl-name</td>
<td>Name of the CA IDMS DMCL describing the CA IDMS files used by the transparency</td>
</tr>
</tbody>
</table>

Diagnosing Errors

Contents

- What to Look For When Errors Occur During Program Processing (see page 24)
- What to Look For When Inaccurate Data is Returned (see page 26)
- Where to Find Values During Debugging (see page 26)

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 S0C2 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 41).

**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 Section.

**Process Indicators**

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

<table>
<thead>
<tr>
<th>IMBSBRDG generates this process indicator:</th>
<th>In response to:</th>
</tr>
</thead>
<tbody>
<tr>
<td>MGET</td>
<td>GE$T</td>
</tr>
<tr>
<td>MPUT</td>
<td>PU$T</td>
</tr>
<tr>
<td>STKY</td>
<td>ST$KY</td>
</tr>
<tr>
<td>STDA</td>
<td>ST$DA</td>
</tr>
</tbody>
</table>

**Table Generation and Accuracy**

Verify that the IMBSEQ, IMBSCOBL, 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.

<table>
<thead>
<tr>
<th>Register</th>
<th>Points to:</th>
</tr>
</thead>
<tbody>
<tr>
<td>R5</td>
<td>IMBSTAB</td>
</tr>
<tr>
<td>R6</td>
<td>Active work area prefix</td>
</tr>
<tr>
<td>R7</td>
<td>Active file and file table in IMBSTAB</td>
</tr>
<tr>
<td>R8</td>
<td>Active record name</td>
</tr>
<tr>
<td>R11</td>
<td>CA IDMS/DB logical record buffer in IMBSTAB</td>
</tr>
<tr>
<td>R12</td>
<td>Beginning of active IMBSBRDG routine</td>
</tr>
<tr>
<td>R14</td>
<td>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</td>
</tr>
</tbody>
</table>

Transparency Programs and Macros

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

<table>
<thead>
<tr>
<th>Component</th>
<th>Brief description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IMBS customizing macro</td>
<td>Describes the DBOMP files and the equivalent CA IDMS/DB database. The IMBS macro generates IMBSTAB.</td>
</tr>
<tr>
<td>IMBSTAB</td>
<td>Contains (in tabular format) the data that the bridge program uses to convert CA IDMS/DB records to DBOMP records.</td>
</tr>
<tr>
<td>IMBSPROC database procedure</td>
<td>Moves pointers from the subschema table into a CA IDMS/DB dummy record.</td>
</tr>
<tr>
<td>IMBSBRDG program module</td>
<td>Simulates DBOMP records and processing using IMBSTAB, IMBSPROC, IMBSEQ (or equivalent COBOL or PL/I macros), and CA IDMS/DB.</td>
</tr>
<tr>
<td>IMBSEQ macro</td>
<td></td>
</tr>
<tr>
<td>Component</td>
<td>Brief description</td>
</tr>
<tr>
<td>-----------</td>
<td>-------------------</td>
</tr>
<tr>
<td></td>
<td>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 61) and see COBOL Considerations (see page 65).</td>
</tr>
</tbody>
</table>

**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:

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 Section), running these reports:

<table>
<thead>
<tr>
<th>Report name</th>
<th>Gives information on:</th>
</tr>
</thead>
<tbody>
<tr>
<td>RECDES</td>
<td>Record types defined in a schema</td>
</tr>
<tr>
<td>SETDES</td>
<td>Sets defined in a schema</td>
</tr>
<tr>
<td>DATDIR</td>
<td>Record types copied into a subschema (general)</td>
</tr>
<tr>
<td>SUBREC</td>
<td>Record types copied into a subschema (comprehensive)</td>
</tr>
<tr>
<td>SUBAREA</td>
<td>Areas copied into a subschema</td>
</tr>
<tr>
<td>SUBSET</td>
<td>Sets copied into a subschema</td>
</tr>
</tbody>
</table>

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 SUBSCH=subschema-name
       SYSTEM= DBMP
       CFMS

,IMBSREC= IMBS-RECORD
             idms-dummy-record-name

,RACN= ITEM-MASTER
       racn-record-name

,USAGE= PU
       PR
       SU
       SR
       ER
       EU
```
,PGMNAME=program-name

,SETLMT= 16
   limit-number

,DBPROC= IMBSPROC
   db-procedure-name

,CATALR= NO
   YES
   relocatable-module-name

- **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  ,KEYL=key-length
           C
           S
      ,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 name 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 on 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>
<thead>
<tr>
<th>Table</th>
<th>Contains:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Control table</td>
<td>Control information</td>
</tr>
<tr>
<td>Set table</td>
<td>An entry for each set described to the IMBS macro</td>
</tr>
<tr>
<td>Pointer table</td>
<td>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.</td>
</tr>
<tr>
<td>File table</td>
<td>A group of entries for each file described to the IMBS macro</td>
</tr>
</tbody>
</table>

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

**Control Table**

<table>
<thead>
<tr>
<th>Displacement</th>
<th>Field Contents</th>
<th>Field Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>System name</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>Addresses of other tables and logical record buffer</td>
<td>16</td>
</tr>
<tr>
<td>10</td>
<td>Advantage CA-IDMS/DB dummy record name</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>Database procedure name</td>
<td>8</td>
</tr>
<tr>
<td>14</td>
<td>Subschema name</td>
<td>8</td>
</tr>
<tr>
<td>52</td>
<td>Program name</td>
<td>8</td>
</tr>
<tr>
<td>60</td>
<td>RACON record name</td>
<td>16</td>
</tr>
<tr>
<td>76</td>
<td>Usage mode</td>
<td>4</td>
</tr>
<tr>
<td>80</td>
<td>Advantage CA-IDMS/DB communications</td>
<td></td>
</tr>
</tbody>
</table>

**Set Table**

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

<table>
<thead>
<tr>
<th>Displacement</th>
<th>Field Contents</th>
<th>Field Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>set-name-1</td>
<td>16</td>
</tr>
<tr>
<td>16</td>
<td>set-name-2</td>
<td>16</td>
</tr>
<tr>
<td>*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>set-name-n</td>
<td></td>
<td>16</td>
</tr>
</tbody>
</table>
**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.

<table>
<thead>
<tr>
<th>Displacement</th>
<th>Field Contents</th>
<th>Field Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>set-name-1 pointers</td>
<td>16</td>
</tr>
<tr>
<td>4</td>
<td>prior pointer</td>
<td>4</td>
</tr>
<tr>
<td>8</td>
<td>current pointer</td>
<td>4</td>
</tr>
<tr>
<td>12</td>
<td>next pointer</td>
<td>4</td>
</tr>
<tr>
<td>16</td>
<td>set-name-2 pointers</td>
<td>16</td>
</tr>
<tr>
<td>20</td>
<td>prior pointer</td>
<td>4</td>
</tr>
<tr>
<td>24</td>
<td>current pointer</td>
<td>4</td>
</tr>
<tr>
<td>28</td>
<td>next pointer</td>
<td>4</td>
</tr>
</tbody>
</table>

**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.
IDMSDB--File Table

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 19).

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)
```
### Sample Output from IMBS

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

<table>
<thead>
<tr>
<th>LOC</th>
<th>OBJECT CODE</th>
<th>ADDR1</th>
<th>ADDR2</th>
<th>IMBS END</th>
<th>STATEMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>000000</td>
<td>47F0 E000</td>
<td>000020</td>
<td>000000</td>
<td>IMBSCNTRL</td>
<td>15,0(,14)</td>
</tr>
<tr>
<td>000004</td>
<td>00000298</td>
<td>000000</td>
<td>000000</td>
<td>DBMP'</td>
<td>A(R1)</td>
</tr>
<tr>
<td>00002C</td>
<td>00000148</td>
<td>000000</td>
<td>000000</td>
<td>A(SETABLE)</td>
<td>A(IMBSCNTL)</td>
</tr>
<tr>
<td>000030</td>
<td>C9D4C2D4D7</td>
<td>000000</td>
<td>000000</td>
<td>CL16'IMBS-RECORD'</td>
<td>CL4'DBMP'</td>
</tr>
<tr>
<td>000044</td>
<td>C9D4C2E2D70D6C3</td>
<td>000000</td>
<td>000000</td>
<td>CL8'IMBSSPROC'</td>
<td>CL8'IMBSSUBS'</td>
</tr>
<tr>
<td>000054</td>
<td>C9C4D4E2C4C2D4D7</td>
<td>000000</td>
<td>000000</td>
<td>CL8'IDMSDBMP'</td>
<td>CL16'ITEM-MASTER'</td>
</tr>
<tr>
<td>000060</td>
<td>C9E3C5D60D4C1E2</td>
<td>000000</td>
<td>000000</td>
<td>A(SSCIDBCM+38-1)</td>
<td>A(SSCIDBCM+38-1)</td>
</tr>
<tr>
<td>000134</td>
<td>C9E3C5D60D4C1E2</td>
<td>000000</td>
<td>000000</td>
<td>000000</td>
<td>000000</td>
</tr>
<tr>
<td>000138</td>
<td>C9E3C5D60D4C1E2</td>
<td>000000</td>
<td>000000</td>
<td>000000</td>
<td>000000</td>
</tr>
<tr>
<td>000148</td>
<td>C9E3C5D60D4C1E2</td>
<td>000000</td>
<td>000000</td>
<td>000000</td>
<td>000000</td>
</tr>
<tr>
<td>000158</td>
<td>C9E3C5D60D4C1E2</td>
<td>000000</td>
<td>000000</td>
<td>000000</td>
<td>000000</td>
</tr>
<tr>
<td>000168</td>
<td>C9E3C5D60D4C1E2</td>
<td>000000</td>
<td>000000</td>
<td>000000</td>
<td>000000</td>
</tr>
<tr>
<td>000178</td>
<td>C9E3C5D60D4C1E2</td>
<td>000000</td>
<td>000000</td>
<td>000000</td>
<td>000000</td>
</tr>
</tbody>
</table>
CA IDMS - 19.0

000190 0000000000000000 52+ DC 16XL16 'FF'
53++
000290 4F80 0000000000000000 54+ FTABLE DS 0D
00029A FF00 0000000000000000 55+ DC F'1'
00029B FF00 57+ DC H'0'
00029B 58+R1 DS 0F
00029B 000002F0 59+ DC A(R2)
00029C 53++
00029D 0000000000000000 54+ FTABLE DS 0D
00029E 0000000000000000 55+ DC F'1'
00029F 0000000000000000 56+ DC H'68'
0002A0 53++
0002A1 0000000000000000 54+ R1 DS 0F
0002A1 0000000000000000 55+ DC A(R3)
0002A2 C9E3C5D4C6D3C5 56+ DC CL7'ITEMFLE'
0002A3 57+ DC C'M'
0002A4 58+ CL16'ITEM-MASTER'
0002A5 59+ DC CL1'
0002A6 60+ DC CL1'
0002A7 61+ DC CL1'
0002A8 62+ DC CL1'
0002A9 63+ DC CL1'
0002AA 64+ DC CL1'
0002AB 65+ DC CL1'
0002AC 66+ DC CL1'
0002AD 67+ DC CL1'
0002AE 68+ DC CL1'
0002AF 69+ DC CL1'
0002B0 70+ DC CL1'
0002B1 71+ DC CL1'
0002B2 72+ DC CL1'
0002B3 73+ DC CL1'
0002B4 74+ DC CL1'
0002B5 75+ DC CL1'
0002B6 76+ DC CL1'
0002B7 77+ DC CL1'
0002B8 78+ DC CL1'
0002B9 79+ DC CL1'
0002BA 80+ DC CL1'
0002BB 81+ DC CL1'
0002BC 82+ DC CL1'
0002BD 83+ DC CL1'
0002BE 84+ DC CL1'
0002BF 85+ DC CL1'
0002C0 86+ DC CL1'
0002C1 87+ DC CL1'
0002C2 88+ DC CL1'
0002C3 89+ DC CL1'
0002C4 90+ DC CL1'
0002C5 91+ DC CL1'
0002C6 92+ DC CL1'
0002C7 93+ DC CL1'
0002C8 94+ DC CL1'
0002C9 95+ DC CL1'
0002CA 96+ DC CL1'
0002CB 97+ DC CL1'
0002CC 98+ DC CL1'
0002CD 99+ DC CL1'
0002CE 100+ DC CL1'
0002CF 101+ DC CL1'
0002D0 102+ DC CL1'
0002D1 103+ DC CL1'
0002D2 104+ DC CL1'
0002D3 105+ DC CL1'
0002D4 106+ DC CL1'
0002D5 107+ DC CL1'
0002D6 108+ DC CL1'
0002D7 109+ DC CL1'
0002D8 110+ DC CL1'
0002D9 111+ DC CL1'
0002DA 112+ DC CL1'
0002DB 113+ DC CL1'
0002DC 114+ DC CL1'
0002DD 115+ DC CL1'
0002DE 116+ DC CL1'
0002DF 117+ DC CL1'
0002E0 118+ DC CL1'
0002E1 119+ DC CL1'
0002E2 120+ DC CL1'
0002E3 121+ DC CL1'
0002E4 122+ DC CL1'
0002E5 123+ DC CL1'
0002E6 124+ DC CL1'
0002E7 125+ DC CL1'
0002E8 126+ DC CL1'
0002E9 127+ DC CL1'
0002EA 128+ DC CL1'
0002EB 129+ DC CL1'
0002EC 130+ DC CL1'
0002ED 131+ DC CL1'
0002EE 132+ DC CL1'
0002EF 133+ DC CL1'
0002F0 134+ DC CL1'
0002F1 135+ DC CL1'
0002F2 136+ DC CL1'
0002F3 137+ DC CL1'
0002F4 138+ DC CL1'
0002F5 139+ DC CL1'
0002F6 140+ DC CL1'
0002F7 141+ DC CL1'
0002F8 142+ DC CL1'
0002F9 143+ DC CL1'
0002FA 144+ DC CL1'
0002FB 145+ DC CL1'
0002FC 146+ DC CL1'
0002FD 147+ DC CL1'
0002FE 148+ DC CL1'
0002FF 149+ DC CL1'
000300 150+ DC CL1'
000301 151+ DC CL1'
000302 152+ DC CL1'
000303 153+ DC CL1'
000304 154+ DC CL1'
000305 155+ DC CL1'
000306 156+ DC CL1'
000307 157+ DC CL1'
000308 158+ DC CL1'
000309 159+ DC CL1'
00030A 160+ DC CL1'
00030B 161+ DC CL1'
00030C 162+ DC CL1'
00030D 163+ DC CL1'
00030E 164+ DC CL1'
00030F 165+ DC CL1'
000310 166+ DC CL1'
000311 167+ DC CL1'
000312 168+ DC CL1'
000313 169+ DC CL1'
000314 170+ DC CL1'
000315 171+ DC CL1'
000316 172+ DC CL1'
000317 173+ DC CL1'
000318 174+ DC CL1'
000319 175+ DC CL1'
00031A 176+ DC CL1'
00031B 177+ DC CL1'
00031C 178+ DC CL1'
00031D 179+ DC CL1'
00031E 180+ DC CL1'
00031F 181+ DC CL1'
000320 182+ DC CL1'
000321 183+ DC CL1'
000322 184+ DC CL1'
000323 185+ DC CL1'
000324 186+ DC CL1'
000325 187+ DC CL1'
000326 188+ DC CL1'
000327 189+ DC CL1'
000328 190+ DC CL1'
000329 191+ DC CL1'
00032A 192+ DC CL1'
00032B 193+ DC CL1'
00032C 194+ DC CL1'
00032D 195+ DC CL1'
00032E 196+ DC CL1'
00032F 197+ DC CL1'
000330 198+ DC CL1'
000331 199+ DC CL1'
000332 200+ DC CL1'
000333 201+ DC CL1'
000334 202+ DC CL1'
000335 203+ DC CL1'
000336 204+ DC CL1'

15-Jan-2018 38/80
000337 40 123+ DC CL1 ' 
000338 000001A0 124+ IMBS POINTER=(02,P,17) 
00033C 0010 125+ DC A(PTRTAB+16*(02-1)) 
00033E D7 126+ DC AL2(17-1) 
00033F 40 127+ DC CL1 'P' 
000340 FFFFFFF 128+ DC CL1 ' 
000344 0024 129+ DC IMBS RECNAME=(WORKCTR,WORK-CEN 
000346 0000 130+ DC F'-1' 
000348 0010 131+ DC H'36' 
000349 0000 132+ DC H'-1' 
00034A 00000390 133+ DC A(R4) 
00034C E6D6D9D23E3D9 134+ DC CL7 'WORKCTR' 
000353 D4 135+ DC C'M' 
000354 E6D6D9D260C3C5D5 136+ DC CL16 'WORK-CENTER' 
000364 0005 137+ DC H'S' 
000366 0020 138+ DC H'32' 
000368 0000000000000000 139+ DC 2F'0' 
000370 00000190 140+ DC A(PTRTAB+16*(01-1)) 
000374 0000 141+ DC AL2(1-1) 
000376 E7 142+ DC CL1 'X' 
000377 40 143+ DC CL1 ' 
000378 000001B0 144+ DC IMBS RECNAME=(ROUTING,ROUTINGS 
00037C D5 145+ DC F'-1' 
00037E 40 146+ DC H'32' 
00037F 0000 147+ DC A(PRRTAB+16*(03-1)) 
000380 000003E8 148+ DC AL2(10-1) 
000384 D9D6E4E3C9D5C7 149+ DC CL1 'N' 
00038C C3 150+ DC CL1 ' 
00038E D5 151+ DC IMBS POINTER=(03,N,10) 
000390 0000009 152+ DC A(PRTTAB+16*(03-1)) 
000394 0000 153+ DC CL1 'X' 
000396 D6 154+ DC CL1 ' 
000398 000001B0 155+ DC IMBS RECNAME=(ROUTING,ROUTINGS 
00039C 40 156+ DC F'-1' 
00039E 0000 157+ DC H'32' 
0003A0 0000000000000000 158+ DC H'-1' 
0003A8 000003E8 159+ DC A(R5) 
0003AC 0000 160+ DC CL7 'ROUTING' 
0003AB 000003E8 161+ DC C'C' 
0003AE 0054 162+ DC CL16 'ROUTINGS' 
0003B0 0000000000000000 163+ DC H'0' 
0003B8 000001C0 164+ DC H'84' 
0003BC 0000 165+ DC A(PRRTAB+16*(04-1)) 
0003BD 0000 166+ DC AL2(1-1) 
0003BE D6 167+ DC CL1 'O' 
0003BF 40 168+ DC CL1 ' 
0003C0 000001C0 169+ DC IMBS RECNAME=(ROUTING,ROUTINGS 
0003C4 0000 170+ DC F'-1' 
0003CA 0000 171+ DC H'32' 
0003C6 D5 172+ DC A(PRRTAB+16*(04-1)) 
0003CC 0000 173+ DC AL2(1-1) 
0003CD 0000 174+ DC CL1 'O' 
0003CF 40 175+ DC CL1 ' 
0003D0 000001B0 176+ DC IMBS RECNAME=(ROUTING,ROUTINGS 
0003D4 0000 177+ DC F'-1' 
0003DA 0000 178+ DC H'32' 
0003DB 000001B0 179+ DC A(PRRTAB+16*(04-1)) 
0003DC 0010 180+ DC AL2(1-1) 
0003DE D7 181+ DC CL1 'O' 
15-Jan-2018 39/80
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:

```cobol
record name is imbs-record.
record id is 799.
location mode is direct.
within bill-of-matrIL 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.

<table>
<thead>
<tr>
<th>Name of routine</th>
<th>What it does</th>
</tr>
</thead>
<tbody>
<tr>
<td>HOUSEKEEPING</td>
<td>Saves registers, Establishes addressability, Sets sequential flag for entry to AP$SEQ</td>
</tr>
<tr>
<td>MAINLINE</td>
<td>Routes all calls to IMBSBRDG: On first call, passes control to INITIALIZATION routine, For all subsequent calls, passes control to PROCESS INDICATOR routine and to FILENAME VERIFICATION routine</td>
</tr>
<tr>
<td>INITIALIZATION</td>
<td>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</td>
</tr>
<tr>
<td>FILENAME VERIFICATION</td>
<td>Equates the DBOMP file name to a CA IDMS/DB record type name</td>
</tr>
<tr>
<td>PROCESS INDICATOR</td>
<td>Equates the DBOMP process indicator to a CA IDMS/DB function</td>
</tr>
<tr>
<td>MOVE RECORD</td>
<td></td>
</tr>
</tbody>
</table>
### 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.
IDMSDB--Converting Records Retrieved from CA IDMS/DB

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.
DBOMP to IDMS

Values Returned to the Calling Program

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

<table>
<thead>
<tr>
<th>Values returned to:</th>
<th>Description of values returned</th>
</tr>
</thead>
<tbody>
<tr>
<td>Work area prefix</td>
<td>A hexadecimal value in the error-byte field, returned after a DBOMP request:</td>
</tr>
<tr>
<td></td>
<td>0000 -- Requested function performed successfully</td>
</tr>
<tr>
<td></td>
<td>0400 -- File name not found in IMBSTAB</td>
</tr>
<tr>
<td></td>
<td>0004 -- Process indicator not found in process indicator table</td>
</tr>
<tr>
<td></td>
<td>0008 -- Invalid record at disk address (MDIR and CDIR process indicators)</td>
</tr>
<tr>
<td></td>
<td>FFFF -- Failure in IMBSBRDG program</td>
</tr>
<tr>
<td></td>
<td>Current disk address, returned when a successful random read (MRAN or MRKY) has been performed</td>
</tr>
<tr>
<td></td>
<td>Current record key, returned when a successful direct read (MDIR, MRDR, CDIR, or CRDR) has been performed</td>
</tr>
<tr>
<td>Work area of the DBOMP file</td>
<td>A DBOMP logical record; after successful execution of a retrieval request</td>
</tr>
<tr>
<td>Currency field in IMBSEQ tables</td>
<td>Current address of a record retrieved by a successful execution of the ST$DA or ST$KY macro</td>
</tr>
</tbody>
</table>
• 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 GET, PUT, 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.
IDMSDB--Sequential File Table Layout

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 61). For more information on the COBOL interface, see COBOL Considerations (see page 65).

Converting DBOMP to CA IDMS/DB

This section 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 Section.

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 Section.

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 Section)
- 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 Section)
- 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 section. 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.
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.
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 nxlnk addnf dbomp-prodstr dbomp-item.
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.

- **DBOMP Process Indicators and Corresponding DML (see page 53)**
- **DBOMP Commands and Corresponding DML (see page 57)**
- **Sequence of Logic in Converted Programs (see page 59)**

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 Section*).

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 Section*).
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 sectionlines 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 60)).

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 69); you can apply this example to user maintenance programs.

Communications Block from Step 2 of Conversion

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Length (in bytes)</th>
<th>Suggested Initial Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program name</td>
<td>Alphanumeric</td>
<td>8</td>
<td>Program name</td>
</tr>
<tr>
<td>Error-status indicator</td>
<td>Alphanumeric</td>
<td>4</td>
<td>'1160'</td>
</tr>
<tr>
<td>Db-key</td>
<td>Binary</td>
<td>4 (fullword)</td>
<td>0000</td>
</tr>
<tr>
<td>Record name</td>
<td>Alphanumeric</td>
<td>16</td>
<td>Spaces</td>
</tr>
<tr>
<td>Area name</td>
<td>Alphanumeric</td>
<td>16</td>
<td>Spaces</td>
</tr>
<tr>
<td>Error set name</td>
<td>Alphanumeric</td>
<td>16</td>
<td>Spaces</td>
</tr>
<tr>
<td>Error record name</td>
<td>Alphanumeric</td>
<td>16</td>
<td>Spaces</td>
</tr>
<tr>
<td>Error area name</td>
<td>Alphanumeric</td>
<td>16</td>
<td>Spaces</td>
</tr>
<tr>
<td>IDBMSCMD array</td>
<td>Alphanumeric</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>Direct db-key</td>
<td>Binary</td>
<td>4 (fullword)</td>
<td>0000</td>
</tr>
<tr>
<td>Reserved for system</td>
<td>Alphanumeric</td>
<td>7</td>
<td>Spaces</td>
</tr>
<tr>
<td>Filler</td>
<td>.</td>
<td>1</td>
<td>.</td>
</tr>
<tr>
<td>Record occurrence</td>
<td>Binary</td>
<td>4 (fullword)</td>
<td>0000</td>
</tr>
<tr>
<td>DML sequence</td>
<td>Binary</td>
<td>4 (fullword)</td>
<td>0000</td>
</tr>
</tbody>
</table>

* 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**

```
@READY ALL
READY
READY ;
```

**CLOS**

```
@FINISH
FINISH;
```

**MADD and MCRT**

```
@STORE REC=
STORE
STORE RECORD;
```

**Associated Logic**

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

**MDEL and MTAG**

```
@ERASE REC ,REC=
    PERMANENT
    SELECTIVE
    ALL
ERASE
    PERMANENT
    SELECTIVE
    ALL
ERASE RECORD ;
```

**Associated Logic**

For MTAG, insert user logic to accomplish tagging.

**CADD**
CA IDMS - 19.0

@store rec=
store
store record;

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)

@store rec=
store
store record;

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)

@store rec=
store
store record;

associated logic

move master record key to program variable storage; build 'chain' record; find calc master record;
store 'chain' record.

CDLS

@erase rec, rec=
permanent
selective
all
erase
permanent
members
selective
all
erase record;
permanent
selective
all

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

@erase rec, rec=
permanent
selective
all
ERASE

PERMANENT MEMBERS
SELECTIVE
ALL

ERASE RECORD ;

PERMANENT
SELECTIVE
ALL

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 sectionlines 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

**CHA$E 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.

**GE$T BMPGET**

@OBTAIN NEXT, SET=
OBTAIN NEXT WITHIN
OBTAIN NEXT SET ;

**PU$T BMPPUT**
CA IDMS - 19.0

@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. (DIRECTKY)

**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
- CG$ST
- CPU$T
- BM$FO
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 60)).

⚠️ **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 Section for COBOL and the CA IDMS DML Reference Section 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 Section*).

1. Remove all program references to DBOMP file work 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 51), in this section).

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 Section*.

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 51) 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 Equivalents* (see page 60)).

### DBOMP Error Codes With CA IDMS/DB Equivalents

<table>
<thead>
<tr>
<th>DBOMP Code</th>
<th>DBOMP P.I. IDMS Status</th>
<th>IDMS Macro</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0400</td>
<td>Any</td>
<td>0308</td>
<td>Any</td>
</tr>
</tbody>
</table>
### PL/I Considerations

This section 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 section are:
- Transparency Support For DBOMP PL/I Commands (see page 61)
- IMBSPL1 Interface Macro (see page 63)
- DBOMP PL/I Program Preparation and Execution (see page 64)

### 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.

<table>
<thead>
<tr>
<th>DBOMP PL/I command</th>
<th>CA IDMS DBOMP Transparency interpretation of command</th>
</tr>
</thead>
<tbody>
<tr>
<td>OP$EN</td>
<td><em>Note: See IBM DBOMP documentation for the syntax for these commands.</em></td>
</tr>
<tr>
<td>Command</td>
<td>CA IDMS DBOMP Transparency interpretation of command</td>
</tr>
<tr>
<td>---------</td>
<td>------------------------------------------------------</td>
</tr>
<tr>
<td><strong>DBOMP PL/I command</strong></td>
<td>The first call to OP$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$EN are ignored once the database has been opened.</td>
</tr>
<tr>
<td><strong>CLO$E</strong></td>
<td>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 CLOSE causes the transparency to close the database again.</td>
</tr>
<tr>
<td><strong>CA$LL</strong></td>
<td>The work area prefix for the named file is passed to IMBSBRDG, which interprets the process indicator contained in the work area prefix and performs the requested function. See Section 3, &quot;The Transparency Environment&quot; for those process indicators supported by the transparency.</td>
</tr>
<tr>
<td><strong>GE$T</strong></td>
<td>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).</td>
</tr>
<tr>
<td><strong>EO$F</strong></td>
<td>IMBSBRDG handles EO$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.</td>
</tr>
<tr>
<td><strong>ST$KY</strong></td>
<td>IMBSBRDG retrieves a record by the key specified in the work area prefix for the named file and returns the record to the work area. The currency for the file is set at the retrieved record. Subsequent GE$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.</td>
</tr>
<tr>
<td><strong>ST$DA</strong></td>
<td>IMBSBRDG retrieves a record by the disk address specified in the work area prefix for the named file and returns the record to the work area. The currency for the file is set at the retrieved record. Subsequent GE$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.</td>
</tr>
<tr>
<td><strong>PU$T</strong></td>
<td>IMBSBRDG writes back to the CA IDMS/DB database the last record retrieved by a GE$T command. Chain address fields (pointers) are not updated or written back to the database.</td>
</tr>
<tr>
<td><strong>CHASE</strong></td>
<td>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$CH macro as indicated in IBM DBOMP documentation.</td>
</tr>
<tr>
<td><strong>BM$OFAD</strong></td>
<td>The transparency does not support this command. If a call to BM$OFAD is encountered by the bridge, no action takes place and control returns to the calling program.</td>
</tr>
<tr>
<td><strong>BM$F$O</strong></td>
<td>The transparency does not support this command. If a BM$F$O command is encountered, an unresolved external reference results in the link edit map.</td>
</tr>
<tr>
<td>Command</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
</tr>
<tr>
<td>BM$RACN</td>
<td>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.</td>
</tr>
</tbody>
</table>

### 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 ), YES ;
         NOTSEQ      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

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 12)):

  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 12)) 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.

COBOL Considerations

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:
- Transparency Support For DBOMP COBOL Commands (see page 65)
- IMBSCOBBL Interface Macro (see page 67)
- DBOMP COBOL Program Preparation and Execution (see page 68)

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.

<table>
<thead>
<tr>
<th>DBOMP PL/I command</th>
<th>CA IDMS DBOMP Transparency interpretation of command</th>
</tr>
</thead>
<tbody>
<tr>
<td>BMPOPE</td>
<td>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.</td>
</tr>
<tr>
<td>BMPCLOSE</td>
<td></td>
</tr>
<tr>
<td>Command</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
</tr>
<tr>
<td>BMPCLOSE</td>
<td>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.</td>
</tr>
<tr>
<td>BMPCALL</td>
<td>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 Section3, &quot;The Transparency Environment&quot;.</td>
</tr>
<tr>
<td>BMPGET</td>
<td>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).</td>
</tr>
<tr>
<td>BMPEOF</td>
<td>IMBSBRDG handles BMPEOF in the same manner as does DBOMP, but obtains the necessary file information from the module generated by the DBOMP CB$BM macro. A call to BMPEOF must specify the end-of-file routines in the same sequence as the corresponding files are entered in the IMBSCOBL macro.</td>
</tr>
<tr>
<td>BMPSTKY</td>
<td>IMBSBRDG retrieves a record by the key specified in the work area prefix for the named file and returns the record to the work area. The currency for the file is set at the retrieved record. Subsequent BMPGET commands for the file retrieve records in logical sequential order from that point if the record type is indexed, or in physical sequential order from that point if the record type is not indexed. Note that the transparency's support of logical sequential processing assumes the use of indexing.</td>
</tr>
<tr>
<td>BMPSTDA</td>
<td>IMBSBRDG retrieves a record by the disk address specified in the work area prefix for the named file and returns the record to the work area. The currency for the file is set at the retrieved record. Subsequent BMPGET commands for the file retrieve records in logical sequential order from that point if the record type is indexed, or in physical sequential order from that point if the record type is not indexed. Note that the transparency's support of logical sequential processing assumes the use of indexing.</td>
</tr>
<tr>
<td>BMPPUT</td>
<td>IMBSBRDG writes back to the CA IDMS/DB database the last record retrieved by a BMPGET command. Chain address fields (pointers) are not updated or written back to the database.</td>
</tr>
<tr>
<td>CHASE</td>
<td>The transparency supports this command unconditionally. Programs that request only the CHASE function need not be modified before interfacing with the bridge, and should be linked with the CB$CH macro as indicated in IBM DBOMP documentation.</td>
</tr>
<tr>
<td>BMPOFFA</td>
<td>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.</td>
</tr>
<tr>
<td>BMPFO</td>
<td>The transparency does not support this command. If a BMPFO statement is encountered, an unresolved external reference results in the link edit map.</td>
</tr>
<tr>
<td>BMPRCN</td>
<td></td>
</tr>
</tbody>
</table>
DBOMP command

CA IDMS DBOMP Transparency interpretation of command

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.

IMBSCOBBL Interface Macro

The IMBSCOBBL 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

IMBSCOBBL macro

►►─── IMBSCOBBL (file-name, ─┬─ index-set-name ─┬─ ) ───────────────────────────►◄
└─NOTSEQ ─────────┘

Parameters

- **IMBSCOBBL**
  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 IMBSCOBBL 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:](Note.png)

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

Assembling and Linking IMBSCOBBL

To assemble and link-edit IMBSCOBBL, you must use SMP/E (Z/OS) or MSHP (Z/VSE).
DBOMP COBOL Program Preparation and Execution

The sectionlines 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 19).

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 12)):
  
  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 12)) 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, IMBSCOBL, and CA IDMS/DB. This step assumes that IMBSCOBL has been assembled and link edited as discussed above.

- Submit the DBOMP COBOL program for execution.

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.

Note: For more information on using SMP/E and MSHP, see the CA IDMS Installation -- Z/OS or the CA IDMS Installation -- Z/VSE.
Sample Application and Procedures

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

IMBSBILL Sample Application (http://wiki-dev.ca.com/display/IDMS/IMBSBILL+Sample+Application)

IMBSBILL 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 (http://wiki-dev.ca.com/pages/viewpage.action?pageId=85197971)

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 (http://wiki-dev.ca.com/pages/viewpage.action?pageId=85197972)

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.

![IDMSDB--IMBSBILL Sample Application](Image)

**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.

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

⚠️ **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.
CA IDMS - 19.0

EXEC statement | What happens
--- | ---
ASMCBDG | Assembles IMBSTAB
LINKCBDG | Link edits IMBSTAB
ASMCOBL | Assembles IMBSCOB interface
LINKCOBL | Link edits IMBSCOB 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.

Setting Up CA IDMS/DBOMP Transparency Under z/OS

Object Modules

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

<table>
<thead>
<tr>
<th>Module</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IMBSPROC</td>
<td>Database procedure</td>
</tr>
<tr>
<td>IMBSBRDG</td>
<td>Bridge program</td>
</tr>
</tbody>
</table>

Source Modules

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

<table>
<thead>
<tr>
<th>Module</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IMBSMJ02</td>
<td>JCL for IMBSMJ02 procedure</td>
</tr>
<tr>
<td>IMBS</td>
<td>Customizing macro</td>
</tr>
<tr>
<td>IMBSASMB</td>
<td>IMBS Assembler interface macro</td>
</tr>
<tr>
<td>IMBSBILL</td>
<td>Sample CA IDMS/DB COBOL manufacturing application program</td>
</tr>
</tbody>
</table>
### Load Modules

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

<table>
<thead>
<tr>
<th>Module</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IMBSSBRDG</td>
<td>Assembler source code for IMBSSBRDG object module</td>
</tr>
<tr>
<td>IMBSCOBL</td>
<td>CA IDMS/DBOMP Transparency COBOL interface macro</td>
</tr>
<tr>
<td>IMBSDBMP</td>
<td>Sample COBOL DBOMP program (to be bridged)</td>
</tr>
<tr>
<td>IMBSDMCL</td>
<td>Sample DMCL description module</td>
</tr>
<tr>
<td>IMBSEQ</td>
<td>CA IDMS/DBOMP Transparency Assembler interface macro</td>
</tr>
<tr>
<td>IMBSINP1</td>
<td>Sample input to IMBSBILL</td>
</tr>
<tr>
<td>IMBSINP2</td>
<td>Sample input to IMBSDBMP</td>
</tr>
<tr>
<td>IMBSPL1</td>
<td>CA IDMS/DBOMP Transparency PL/I interface macro</td>
</tr>
<tr>
<td>IMBSPROC</td>
<td>Source code for database procedure object module</td>
</tr>
<tr>
<td>IMBSMJ01</td>
<td>JCL for IMBSMJ01 procedure</td>
</tr>
<tr>
<td>IMBSSCHM</td>
<td>Sample CA IDMS/DB schema description</td>
</tr>
<tr>
<td>IMBSUBS</td>
<td>Sample CA IDMS/DB subschema description</td>
</tr>
<tr>
<td>IMBSTAB</td>
<td>Sample input to IMBS customizing macro</td>
</tr>
</tbody>
</table>

For more information, see the following topics:
- [Customizing and Executing IMBSMJ01 and IMBSMJ02](#) (see page 73)
- [Explanation of EXEC Statements in IMBSMJ01 Procedure](#) (see page 74)
- [Customizing IMBSMJ01](#) (see page 74)
- [IMBSMJ01 (z/OS)](#) (see page 74)
- [Explanation of EXEC Statements in IMBSMJ02 Procedure](#) (see page 76)
Explanation of EXEC Statements in IMBSMJ01 Procedure

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

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

**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 74).

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',
```
```c
// IMBSBILL='imbs013',
// IMBSWORK='IMBSWORK',
// SRCLIB='imbs.srclib',
// IDMSSRC='yourHLQ.CAGJMAC',
// VOL='VOL=SER=nnnnnnn',
// SYSCTLDS='idms.sysctl',
// IDMSDMCL='cvdmcl',
// MSGDD='dcmg',
// MSGDNS='idms.ddldcmsg',
// DDLDD='sysddl',
// DDLDNS='idms.sysddl',
// DICTNAME='appldict',
```

## Parameter Description

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>disk</td>
<td>Symbolic device name for data dictionary and database files</td>
</tr>
<tr>
<td>IMBSBILL</td>
<td>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</td>
</tr>
<tr>
<td>IMBSWOR</td>
<td>Dataset name of sample CA IDMS database file</td>
</tr>
<tr>
<td>IMBSBILL</td>
<td>Dataset name of sample CA IDMS database file</td>
</tr>
<tr>
<td>IMBSWOR</td>
<td>Dataset name of sample CA IDMS database file</td>
</tr>
<tr>
<td>imbs.</td>
<td>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</td>
</tr>
<tr>
<td>coblib</td>
<td>Dataset name of COBOL library</td>
</tr>
<tr>
<td>cob.</td>
<td>Dataset name of COBOL step library</td>
</tr>
<tr>
<td>data.</td>
<td>Dataset name of data dictionary; may be a sample or user directory</td>
</tr>
<tr>
<td>direct</td>
<td>Dataset name of data dictionary; may be a sample or user directory</td>
</tr>
<tr>
<td>VOL</td>
<td>Volume serial number of disk where data dictionary and sample CA IDMS database files are stored</td>
</tr>
<tr>
<td>idms.</td>
<td>Dataset name of IDMS SYSCTL file for running CV</td>
</tr>
<tr>
<td>sysctl</td>
<td>Name of the DMCL that IDMS uses, for CV or local</td>
</tr>
<tr>
<td>cvdmcl</td>
<td>Name of the DMCL that IDMS uses, for CV or local</td>
</tr>
<tr>
<td>药材</td>
<td>The ddname or IDMS message area</td>
</tr>
<tr>
<td>dcmg</td>
<td>The ddname or IDMS message area, for CV and local jobs</td>
</tr>
<tr>
<td>idms.</td>
<td>Dataset name of the IDMS dictionary</td>
</tr>
<tr>
<td>sysddl</td>
<td>Dataset name of the IDMS dictionary</td>
</tr>
<tr>
<td>appldict</td>
<td>Dictionary to be used</td>
</tr>
</tbody>
</table>
**Explanation of EXEC Statements in IMBSMJ02 Procedure**

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

<table>
<thead>
<tr>
<th>EXEC statement</th>
<th>What happens</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASMCBDG</td>
<td>Assembles IMBSTAB</td>
</tr>
<tr>
<td>LINKCBDG</td>
<td>Link edits IMBSTAB</td>
</tr>
<tr>
<td>ASMCOBL</td>
<td>Assembles IMBSCOBL interface</td>
</tr>
<tr>
<td>LINKCOBL</td>
<td>Link edits IMBSCOBL module</td>
</tr>
<tr>
<td>DMLC</td>
<td>Submits sample COBOL DBOMP source program, IMBSDBMP, to the CA IDMS DML compiler</td>
</tr>
<tr>
<td>COB</td>
<td>Compiles output from DMLC</td>
</tr>
<tr>
<td>LINKCOB</td>
<td>Link edits IMBSDBMP</td>
</tr>
<tr>
<td>EXECPGM</td>
<td>Executes the sample DBOMP application program, IMBSDBMP, using the CA IDMS /DBOMP Transparency bridge program IMBSBRDG</td>
</tr>
</tbody>
</table>

**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 76).

**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',
// IDMSSRC='yourHLQ.CAGJMAC',
// COBBLIB='coblib',
// COBSTEP='cob.steplib',
// BASE='data.direct',
// IMBSBILL='imbs013',
// IMBSWORK='IMBSWORK',
// SRCLIB='imbs.srclib',
```

<table>
<thead>
<tr>
<th>Parameter Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>disk</td>
</tr>
</tbody>
</table>
## Parameter Description

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>imbs. loadlib</td>
<td>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</td>
</tr>
<tr>
<td>idms. loadlib</td>
<td>Dataset name of CA IDMS load library</td>
</tr>
<tr>
<td>yourHLQ. CAGIJMAC</td>
<td>Dataset name of CA IDMS macro library</td>
</tr>
<tr>
<td>coblib</td>
<td>Dataset name of COBOL library</td>
</tr>
<tr>
<td>cob. steplib</td>
<td>Dataset name of COBOL step library</td>
</tr>
<tr>
<td>data. direct</td>
<td>Dataset name of data dictionary; may be a sample or user directory</td>
</tr>
<tr>
<td>IMBSBILL</td>
<td>Dataset name of sample CA IDMS database file</td>
</tr>
<tr>
<td>IMBSWOR</td>
<td>Dataset name of sample CA IDMS database file</td>
</tr>
<tr>
<td>imbs. srclib</td>
<td>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</td>
</tr>
</tbody>
</table>

## Executing IMBSMJ01 and IMBSMJ02

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

## Setting Up CA IDMS DBOMP Transparency under Z/VSE

## Customizing and Executing IMBSVJ01 and IMBSVJ02

### Contents

- Explanation of EXEC Statements in IMBSVJ01 Procedure (see page 78)
- Explanation of EXEC Statements in IMBSVJ02 Procedure (see page 78)

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.

<table>
<thead>
<tr>
<th>EXEC statement</th>
<th>What happens</th>
</tr>
</thead>
<tbody>
<tr>
<td>IDMSCHEM</td>
<td>Compiles the sample schema, IMBSSCHM</td>
</tr>
<tr>
<td>IDMSDMCL</td>
<td>Compiles the sample and punches DMCL module, IMBSDMCL</td>
</tr>
<tr>
<td>LNKEDT</td>
<td>Link edits sample DMCL module, IMBSDMCL</td>
</tr>
<tr>
<td>IDMSUBSC</td>
<td>Compiles the sample and punches subschema, IMBSSUBS</td>
</tr>
<tr>
<td>ASSEMBLY</td>
<td>Assembles IMBSSUBS</td>
</tr>
<tr>
<td>LNKEDT</td>
<td>Link edits IMBSSUBS</td>
</tr>
<tr>
<td>IDMSDMLC</td>
<td>Submits the sample COBOL program, IMBSBILL, to the CA IDMS Data Manipulation Language compiler</td>
</tr>
<tr>
<td>FCOBOL</td>
<td>Submits IMBSBILL to the COBOL compiler</td>
</tr>
<tr>
<td>LNKEDT</td>
<td>Link edits IMBSBILL</td>
</tr>
<tr>
<td>IDMSRPTS</td>
<td>Prints all dictionary/directory reports</td>
</tr>
<tr>
<td>IDMSBCF</td>
<td>Initializes the sample database</td>
</tr>
<tr>
<td>IMBSBILL</td>
<td>Executes the sample program, IMBSBILL</td>
</tr>
</tbody>
</table>

**Explanation of EXEC Statements in IMBSVJ02 Procedure**

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

<table>
<thead>
<tr>
<th>EXEC statement</th>
<th>What happens</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASSEMBLY</td>
<td>Assembles the IMBS customizing macro</td>
</tr>
<tr>
<td>MAINT</td>
<td>Catalogs IMBSTAB to relocatable library</td>
</tr>
<tr>
<td>ASSEMBLY</td>
<td>Assembles the IMBSCOBL macro</td>
</tr>
<tr>
<td>MAINT</td>
<td>Catalogs assembled IMBSCOBL to relocatable library</td>
</tr>
<tr>
<td>IDMSDMLC</td>
<td>Submits the sample COBOL DBOMP program, IMBSDBMP, to the Data Manipulation Language compiler</td>
</tr>
<tr>
<td>FCOBOL</td>
<td>Submits IMBSDBMP to the COBOL compiler</td>
</tr>
<tr>
<td>LNKEDT</td>
<td>Link edits IMBSDBMP</td>
</tr>
<tr>
<td>DEMOPROG</td>
<td>Executes the sample DBOMP program, IMBSDBMP, against CA IDMS DBOMP Transparency</td>
</tr>
</tbody>
</table>
Modules placed in the relocatable library

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

<table>
<thead>
<tr>
<th>Module</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IMBSBRDG</td>
<td>Bridge program</td>
</tr>
<tr>
<td>IMBSPROC</td>
<td>Database procedure</td>
</tr>
</tbody>
</table>

Modules placed in the source statement library

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

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

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.