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

Copying and Pasting COBOL Code ................................................................. 20

Introduction to CA IDMS Data Manipulation Language (COBOL) .............. 21
  Programming in the CA IDMS Environment (COBOL) ................................ 22
    Accessing the Database ........................................................................... 23
      Navigational DML Statements ............................................................... 23
      LRF DML Statements ........................................................................... 24
      SQL DML Statements .......................................................................... 24
  Navigational DML Statements .................................................................. 23
  LRF DML Statements .............................................................................. 24
  SQL DML Statements ............................................................................. 24
  Programming in the Online Environment .................................................. 25
  Compiling and Executing CA IDMS Programs (COBOL) ............................ 26
    Compiling Programs .............................................................................. 26
    Executing Programs ............................................................................. 29
  Callable Services and Common Facilities (COBOL) .................................... 30

Precompiler Options .................................................................................... 32
  Dictionary Ready Override ....................................................................... 32
  Comment Generation ............................................................................... 33
  List Generation ....................................................................................... 33
  Log Suppression ...................................................................................... 34

Communications Blocks and Error Detection (COBOL) ............................... 35
  16-Byte Communications Blocks (COBOL) ............................................... 35
    IDMS Communications Block (COBOL) ................................................ 35
    IDMS-DC Communications Block (COBOL) .......................................... 40
    Logical-Record Request Control (LRC) Block (COBOL) ......................... 43
  18-Byte Communications Blocks (COBOL) ............................................... 45
    18-Byte IDMS Block ............................................................................. 45
    18-Byte IDMS-DC Block ...................................................................... 46
  ERROR-STATUS Field and Codes ............................................................... 47
    DB Status Codes (COBOL) .................................................................... 47
      Major DB Status Codes ..................................................................... 47
      Minor DB Status Codes ..................................................................... 48
<table>
<thead>
<tr>
<th>Status Codes</th>
<th>91</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACCEPT DATABASE STATISTICS (COBOL)</td>
<td>91</td>
</tr>
<tr>
<td>Syntax</td>
<td>91</td>
</tr>
<tr>
<td>Parameters</td>
<td>91</td>
</tr>
<tr>
<td>Example</td>
<td>91</td>
</tr>
<tr>
<td>Status Codes</td>
<td>93</td>
</tr>
<tr>
<td>ACCEPT DB-KEY FROM CURRENCY (COBOL)</td>
<td>93</td>
</tr>
<tr>
<td>Syntax</td>
<td>93</td>
</tr>
<tr>
<td>Parameters</td>
<td>93</td>
</tr>
<tr>
<td>Example</td>
<td>94</td>
</tr>
<tr>
<td>Status Codes</td>
<td>94</td>
</tr>
<tr>
<td>ACCEPT DB-KEY RELATIVE TO CURRENCY (COBOL)</td>
<td>95</td>
</tr>
<tr>
<td>Syntax</td>
<td>95</td>
</tr>
<tr>
<td>Parameters</td>
<td>95</td>
</tr>
<tr>
<td>Example</td>
<td>97</td>
</tr>
<tr>
<td>Status Codes</td>
<td>97</td>
</tr>
<tr>
<td>ACCEPT page-info-location (COBOL)</td>
<td>97</td>
</tr>
<tr>
<td>Syntax</td>
<td>98</td>
</tr>
<tr>
<td>Parameters</td>
<td>98</td>
</tr>
<tr>
<td>Example</td>
<td>98</td>
</tr>
<tr>
<td>Status Codes</td>
<td>98</td>
</tr>
<tr>
<td>ACCEPT PROCEDURE CONTROL LOCATION (COBOL)</td>
<td>99</td>
</tr>
<tr>
<td>Syntax</td>
<td>99</td>
</tr>
<tr>
<td>Parameters</td>
<td>99</td>
</tr>
<tr>
<td>Example</td>
<td>99</td>
</tr>
<tr>
<td>Status Codes</td>
<td>99</td>
</tr>
<tr>
<td>ACCEPT TRANSACTION STATISTICS (COBOL)</td>
<td>100</td>
</tr>
<tr>
<td>Syntax</td>
<td>100</td>
</tr>
<tr>
<td>Parameters</td>
<td>100</td>
</tr>
<tr>
<td>Example</td>
<td>103</td>
</tr>
<tr>
<td>Status Codes</td>
<td>103</td>
</tr>
<tr>
<td>ATTACH (COBOL)</td>
<td>104</td>
</tr>
<tr>
<td>Syntax</td>
<td>104</td>
</tr>
<tr>
<td>Parameters</td>
<td>104</td>
</tr>
<tr>
<td>Example</td>
<td>104</td>
</tr>
<tr>
<td>Status Codes</td>
<td>105</td>
</tr>
<tr>
<td>BIND MAP (COBOL)</td>
<td>105</td>
</tr>
<tr>
<td>Syntax</td>
<td>106</td>
</tr>
<tr>
<td>Parameters</td>
<td>106</td>
</tr>
<tr>
<td>Example</td>
<td>106</td>
</tr>
<tr>
<td>Status Codes</td>
<td>106</td>
</tr>
</tbody>
</table>
BIND PROCEDURE (COBOL) ........................................................................................................107
Syntax .................................................................................................................................107
Parameters ..........................................................................................................................107
Example .............................................................................................................................107
Status Codes .......................................................................................................................107
BIND RECORD (COBOL) ......................................................................................................108
Syntax .................................................................................................................................108
Parameters ..........................................................................................................................108
Example .............................................................................................................................109
Status Codes .......................................................................................................................109
BIND RUN-UNIT (COBOL) ..................................................................................................109
Syntax .................................................................................................................................110
Parameters ..........................................................................................................................110
Example .............................................................................................................................111
Status Codes .......................................................................................................................111
BIND TASK (COBOL) ..........................................................................................................112
Syntax .................................................................................................................................113
Parameters ..........................................................................................................................113
Example .............................................................................................................................113
Status Codes .......................................................................................................................113
BIND TRANSACTION STATISTICS (COBOL) ....................................................................113
Syntax .................................................................................................................................114
Example .............................................................................................................................114
Status Codes .......................................................................................................................114
CHANGE PRIORITY (COBOL) ..............................................................................................114
Syntax .................................................................................................................................114
Parameters ..........................................................................................................................114
Example .............................................................................................................................115
Status Codes .......................................................................................................................115
CHECK TERMINAL (COBOL) .............................................................................................115
Syntax .................................................................................................................................115
Parameters ..........................................................................................................................115
Example .............................................................................................................................116
Status Codes .......................................................................................................................116
COMMIT (COBOL) .............................................................................................................117
Syntax .................................................................................................................................117
Parameters ..........................................................................................................................117
Example .............................................................................................................................118
Status Codes .......................................................................................................................118
CONNECT (COBOL) ..........................................................................................................118
Syntax .................................................................................................................................119
<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status Codes</td>
<td>133</td>
</tr>
<tr>
<td>ENDPAGE (COBOL)</td>
<td>134</td>
</tr>
<tr>
<td>Syntax</td>
<td>134</td>
</tr>
<tr>
<td>Example</td>
<td>134</td>
</tr>
<tr>
<td>Status Codes</td>
<td>134</td>
</tr>
<tr>
<td>ENQUEUE (COBOL)</td>
<td>134</td>
</tr>
<tr>
<td>Syntax</td>
<td>135</td>
</tr>
<tr>
<td>Parameters</td>
<td>135</td>
</tr>
<tr>
<td>Examples</td>
<td>135</td>
</tr>
<tr>
<td>Status Codes</td>
<td>136</td>
</tr>
<tr>
<td>ERASE (COBOL)</td>
<td>136</td>
</tr>
<tr>
<td>Syntax</td>
<td>137</td>
</tr>
<tr>
<td>Parameters</td>
<td>137</td>
</tr>
<tr>
<td>Example</td>
<td>138</td>
</tr>
<tr>
<td>Status Codes</td>
<td>140</td>
</tr>
<tr>
<td>ERASE (LRF) (COBOL)</td>
<td>141</td>
</tr>
<tr>
<td>Syntax</td>
<td>142</td>
</tr>
<tr>
<td>Parameters</td>
<td>142</td>
</tr>
<tr>
<td>Example</td>
<td>142</td>
</tr>
<tr>
<td>FIND/OBTAIN (COBOL)</td>
<td>143</td>
</tr>
<tr>
<td>FIND/OBTAIN CALC/DUPLICATE (COBOL)</td>
<td>144</td>
</tr>
<tr>
<td>Syntax</td>
<td>144</td>
</tr>
<tr>
<td>Parameters</td>
<td>144</td>
</tr>
<tr>
<td>Example</td>
<td>145</td>
</tr>
<tr>
<td>Status Codes</td>
<td>145</td>
</tr>
<tr>
<td>FIND/OBTAIN CURRENT (COBOL)</td>
<td>145</td>
</tr>
<tr>
<td>Syntax</td>
<td>146</td>
</tr>
<tr>
<td>Parameters</td>
<td>146</td>
</tr>
<tr>
<td>Example</td>
<td>146</td>
</tr>
<tr>
<td>Status Codes</td>
<td>147</td>
</tr>
<tr>
<td>FIND/OBTAIN DB-KEY (COBOL)</td>
<td>148</td>
</tr>
<tr>
<td>Syntax</td>
<td>148</td>
</tr>
<tr>
<td>Parameters</td>
<td>148</td>
</tr>
<tr>
<td>Example</td>
<td>149</td>
</tr>
<tr>
<td>Status Codes</td>
<td>149</td>
</tr>
<tr>
<td>FIND/OBTAIN OWNER (COBOL)</td>
<td>150</td>
</tr>
<tr>
<td>Syntax</td>
<td>150</td>
</tr>
<tr>
<td>Parameters</td>
<td>151</td>
</tr>
<tr>
<td>Example</td>
<td>151</td>
</tr>
<tr>
<td>Status Codes</td>
<td>152</td>
</tr>
<tr>
<td>FIND/OBTAIN WITHIN SET/AREA (COBOL)</td>
<td>152</td>
</tr>
<tr>
<td>Syntax</td>
<td>153</td>
</tr>
</tbody>
</table>
Parameters ............................................................................................................................ 153
Examples ................................................................................................................................. 154
Status Codes ............................................................................................................................ 156

FIND/OBTAIN WITHIN SET USING SORT KEY (COBOL) ....................................................... 157
Syntax .................................................................................................................................... 158
Parameters ............................................................................................................................... 158
Example .................................................................................................................................. 159
Status Codes ............................................................................................................................ 159

FINISH (COBOL) ..................................................................................................................... 160
Syntax .................................................................................................................................... 160
Parameters ............................................................................................................................... 161
Example .................................................................................................................................. 161
Status Codes ............................................................................................................................ 161

FREE STORAGE (COBOL) ........................................................................................................ 161
Syntax .................................................................................................................................... 161
Parameters ............................................................................................................................... 162
Example .................................................................................................................................. 162
Status Codes ............................................................................................................................ 162

GET (COBOL) .......................................................................................................................... 162
Syntax .................................................................................................................................... 163
Parameters ............................................................................................................................... 163
Example .................................................................................................................................. 163
Status Codes ............................................................................................................................ 163

GET QUEUE (COBOL) ................................................................................................................ 164
Syntax .................................................................................................................................... 164
Parameters ............................................................................................................................... 164
Example .................................................................................................................................. 165
Status Codes ............................................................................................................................ 166

GET SCRATCH (COBOL) .......................................................................................................... 166
Syntax .................................................................................................................................... 166
Parameters ............................................................................................................................... 167
Example .................................................................................................................................. 168
Status Codes ............................................................................................................................ 168

GET STORAGE (COBOL) ......................................................................................................... 168
Syntax .................................................................................................................................... 168
Parameters ............................................................................................................................... 169
Example .................................................................................................................................. 169
Status Codes ............................................................................................................................ 171

GET TIME (COBOL) .................................................................................................................. 172
Syntax .................................................................................................................................... 172
Parameters ............................................................................................................................... 172
Example .................................................................................................................................. 173

DML Reference for COBOL 10
<table>
<thead>
<tr>
<th>Procedure</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>PUT QUEUE (COBOL)</td>
<td>221</td>
</tr>
<tr>
<td>Syntax</td>
<td>221</td>
</tr>
<tr>
<td>Parameters</td>
<td>221</td>
</tr>
<tr>
<td>Example</td>
<td>222</td>
</tr>
<tr>
<td>Status Codes</td>
<td>222</td>
</tr>
<tr>
<td>PUT SCRATCH (COBOL)</td>
<td>223</td>
</tr>
<tr>
<td>Syntax</td>
<td>223</td>
</tr>
<tr>
<td>Parameters</td>
<td>223</td>
</tr>
<tr>
<td>Example</td>
<td>224</td>
</tr>
<tr>
<td>Status Codes</td>
<td>224</td>
</tr>
<tr>
<td>READ LINE FROM TERMINAL (COBOL)</td>
<td>224</td>
</tr>
<tr>
<td>Syntax</td>
<td>224</td>
</tr>
<tr>
<td>Parameters</td>
<td>225</td>
</tr>
<tr>
<td>Examples</td>
<td>225</td>
</tr>
<tr>
<td>Status Codes</td>
<td>226</td>
</tr>
<tr>
<td>READ TERMINAL (COBOL)</td>
<td>226</td>
</tr>
<tr>
<td>Syntax</td>
<td>226</td>
</tr>
<tr>
<td>Parameters</td>
<td>226</td>
</tr>
<tr>
<td>Example</td>
<td>228</td>
</tr>
<tr>
<td>Status Codes</td>
<td>228</td>
</tr>
<tr>
<td>READY (COBOL)</td>
<td>228</td>
</tr>
<tr>
<td>Syntax</td>
<td>229</td>
</tr>
<tr>
<td>Parameters</td>
<td>229</td>
</tr>
<tr>
<td>Example</td>
<td>230</td>
</tr>
<tr>
<td>Status Codes</td>
<td>230</td>
</tr>
<tr>
<td>RETURN (COBOL)</td>
<td>231</td>
</tr>
<tr>
<td>Syntax</td>
<td>231</td>
</tr>
<tr>
<td>Parameters</td>
<td>232</td>
</tr>
<tr>
<td>Example</td>
<td>232</td>
</tr>
<tr>
<td>Status Codes</td>
<td>233</td>
</tr>
<tr>
<td>ROLLBACK (COBOL)</td>
<td>233</td>
</tr>
<tr>
<td>Syntax</td>
<td>234</td>
</tr>
<tr>
<td>Parameters</td>
<td>234</td>
</tr>
<tr>
<td>Example</td>
<td>234</td>
</tr>
<tr>
<td>Status Codes</td>
<td>235</td>
</tr>
<tr>
<td>SEND MESSAGE (COBOL)</td>
<td>235</td>
</tr>
<tr>
<td>Syntax</td>
<td>235</td>
</tr>
<tr>
<td>Parameters</td>
<td>235</td>
</tr>
<tr>
<td>Examples</td>
<td>236</td>
</tr>
<tr>
<td>Status Codes</td>
<td>237</td>
</tr>
<tr>
<td>SET ABEND EXIT (COBOL)</td>
<td>237</td>
</tr>
</tbody>
</table>
Example .................................................................................................................. 255
Status Codes ............................................................................................................ 255
WRITE LINE TO TERMINAL (COBOL) ................................................................. 256
  Syntax .................................................................................................................... 256
  Parameters ............................................................................................................ 257
  Examples ............................................................................................................... 258
  Status Codes ....................................................................................................... 258
WRITE LOG (COBOL) ............................................................................................. 258
  Syntax .................................................................................................................... 259
  Parameters ............................................................................................................ 260
  Example ............................................................................................................... 262
  Status Codes ....................................................................................................... 263
WRITE PRINTER (COBOL) ....................................................................................... 263
  Syntax .................................................................................................................... 264
  Parameters ............................................................................................................ 264
  Examples ............................................................................................................... 266
  Status Codes ....................................................................................................... 266
WRITE TERMINAL (COBOL) ................................................................................... 267
  Syntax .................................................................................................................... 267
  Parameters ............................................................................................................ 267
  Example ............................................................................................................... 268
  Status Codes ....................................................................................................... 268
WRITE THEN READ TERMINAL (COBOL) ........................................................... 269
  Syntax .................................................................................................................... 269
  Parameters ............................................................................................................ 269
  Example ............................................................................................................... 271
  Status Codes ....................................................................................................... 271
Logical-Record Clauses (COBOL) ............................................................................ 272
  WHERE ................................................................................................................. 272
    Syntax .................................................................................................................. 273
    Parameters ........................................................................................................... 273
    Expansion of comparison ..................................................................................... 273
    Parameters ........................................................................................................... 274
    Examples ............................................................................................................. 276
  ON .......................................................................................................................... 276
    Syntax .................................................................................................................. 278
    Parameters ........................................................................................................... 278
    Examples ............................................................................................................. 278
    Status Codes ...................................................................................................... 278
DML Reference for COBOL

This section contains reference material for writing applications programs in the COBOL language to use CA IDMS/DB, CA IDMS/DC, and CA IDMS UCF services.

This section is intended to be used by COBOL programmers whose programs access CA IDMS databases and who want to use the DC/UCF system facilities. Programmers using Assembler language or PL/I should refer to CA IDMS DML Reference section for Assembler or CA IDMS DML Reference section for PL/I.
Copying and Pasting COBOL Code

COBOL compiler requires that the source code is correctly indented.

To preserve the indention of code examples in this section, copy the code examples from the HTML version of the section. Copying from the PDF format does not preserve the indention; it is necessary to restore the indention manually after pasting.

For more information, see the following topics:

- Introduction to CA IDMS Data Manipulation Language (COBOL) (see page 21)
- Precompiler Options (see page 32)
- Communications Blocks and Error Detection (COBOL) (see page 35)
- Precompiler-Directive Statements (see page 63)
- COBOL Data Manipulation Language (DML) Statements (see page 81)
- DML Precompile, COBOL Compile, and Link-Edit JCL (see page 280)
- Sample Batch Program (see page 299)
- Sample Online Program (see page 328)
- Call Formats (COBOL) (see page 358)
- CA IDMS Keywords (see page 377)
- Notes to Teleprocessing Monitor Users (see page 380)
- EMPLOYEE Database Definition (see page 383)
- VS COBOL II Support (see page 391)
- Considerations for IBM Language Environment (see page 395)
- Optional Online COBOL Functionality (see page 404)
- Online Debugger Syntax (COBOL) (see page 409)
Introduction to CA IDMS Data Manipulation Language (COBOL)

The CA IDMS data manipulation language (DML) consists of statements that direct CA IDMS database (DB) and data communications (DC) processing. DML statements are coded in the program source as if they were a part of the host language. The precompiler converts DML statements into standard COBOL statements and performs source-level error checking.

The operating environment determines which DML statements your program uses. A batch program uses only database DML statements, but an online program uses both database and data communications DML statements.

- **Batch processing** typically involves large transaction volumes, sequential processing, and output in the form of files and reports. Batch programs use database DML statements only.

The following figure illustrates the flow of a typical batch application. Input to DEPTRPT consists of department IDs. Output consists of a listing of departments and their employees. The error report lists the department IDs of missing and empty departments.
Online processing typically involves transaction requests that are entered from terminals that are connected directly to the computer, transaction results that are displayed at the terminal, multiple requests from multiple sources, and sharing one copy of a program among multiple users. Also, online processing is immediate. Fast response time is essential when processing large transaction volumes from multiple online users. Online programs use data communications DML statements and can include database DML statements.

The following figure illustrates the flow of a typical online application. EMPDISP retrieves information for an operator-specified employee ID. Output to the terminal consists of DEPARTMENT, EMPLOYEE, JOB, and OFFICE information.

Programming in the CA IDMS Environment (COBOL)

CA IDMS DML statements are divided into two categories:
Database statements perform retrieval and update functions in the batch and the online environments. Database statements fall into one of the following categories:

- **Navigational statements** access database records and sets, one record at a time.
- **LRF statements** access groups of database records by using the Logical Record Facility (LRF).
- **SQL statements** access groups of database records by using the Structured Query Language (SQL).
- **Data communications statements** (also known as online statements) request data communications services, such as for online programs.

You can include database statements in batch programs or can combine them with data communications statements in online programs that require database access. See the following discussions about accessing the database by using DML statements and additional considerations for coding online programs.

**Accessing the Database**

In CA IDMS, the database can be accessed by using navigational, LRF DML, or SQL DML statements:

- Navigational statements are used with a subschema usage mode of DML or MIXED.
- LRF DML statements are used with a subschema usage mode of LR or MIXED.
- SQL DML statements access records without reference to subschemas.

Some statements, such as BIND RUN-UNIT, READY, and FINISH, are used in all three environments. They are noted in the individual descriptions of each DML statement in Data Manipulation Language Statements (see page 81).

**Navigational DML Statements**

Navigational DML statements access the database records and sets one record at a time, checking and maintaining currency to ensure correct results. To use navigational DML statements, you must have a thorough knowledge of the database structure. For an example of a data structure diagram, see the EMPLOYEE Database Definition (see page 383).

Navigational DML statements provide the following advantages:

- **Control over error checking**—You can check the result of each navigational statement, enabling more thorough error detection.
- **Flexibility in choosing a database access strategy**—You can enter the database sequentially (area sweep), by using a symbolic key value (CALC), or by using a database key value (DIRECT).

Navigational DML statements are grouped into the following categories:
Control statements initiate and terminate processing, effect recovery, prevent concurrent updates, and evaluate set conditions.

Retrieval statements locate data in the database and make it available to the application program.

Modification statements update the database.

Accept statements pass database keys, storage address information, and statistics to the program.

LRF DML Statements

LRF DML statements use the Logical Record Facility (LRF) to access database records. LRF lets you access fields from multiple database records as if they were data fields in a single record. LRF DML statements let you specify selection criteria (by using the WHERE clause) that enable you to access only the needed logical records.

LRF DML statements provide the following advantages:

- Easy access to database records—You do not have to be familiar with database structure. Your programs do not have to include database navigation logic.
- Data flexibility—Modification and recompilation of LRF programs are not necessarily required when the physical or logical structure of the database is changed.
- Run-time efficiency—LRF minimizes communication between the program and the DBMS.

The following LRF DML statements are available:

- **ERASE** deletes a logical record as specified in the path definition.
- **MODIFY** modifies a logical record as specified in the path definition.
- **OBTAIN** retrieves a logical record as specified in the path definition.
- **STORE** stores a new logical record as specified in the path definition.

SQL DML Statements

You can use SQL DML to access the same databases that you access using navigational DML. Also, you can use SQL DML to access databases that have been defined using SQL DDL. You do not have to be familiar with database structure, and your programs do not have to include database navigation logic.

You can perform the following functions with SQL DML statements:

- **Select rows** ([https://docops.ca.com/display/IDMSCU19/SELECT](https://docops.ca.com/display/IDMSCU19/SELECT))
- **Update rows** ([https://docops.ca.com/display/IDMSCU19/UPDATE](https://docops.ca.com/display/IDMSCU19/UPDATE))
- **Delete rows** ([https://docops.ca.com/display/IDMSCU19/DELETE](https://docops.ca.com/display/IDMSCU19/DELETE))
Programming in the Online Environment

The CA IDMS/DC system is fully integrated with the CA IDMS DBMS and the data dictionary. You can request data communications and database services through the standard subroutine calls that the precompiler generates from DML statements.

The following figure illustrates a typical stream of online DML statements in a COBOL program. This example maps in a user-specified employee ID, retrieves and displays the specified information, and performs a DC RETURN that names TSK02 as the next task to perform:

```cobol
PROCEDURE DIVISION.
  BIND MAP EMPMAPLR.
  BIND MAP EMPMAPLR RECORD EMPLOYEE.
  ACCEPT TASK CODE INTO TASK-CODE-IN.
  IF TASK-CODE-IN = 'TSK01'
    GO TO INITIAL-MAPOUT.
  MAP IN USING EMPMAPLR.
  ...
  navigational, LRF, or SQL database DML statements
  ...
  MAP OUT USING EMPMAPLR
  OUTPUT DATA IS YES
  MESSAGE IS DISPLAY-MESSAGE LENGTH 80.
  DC RETURN NEXT TASK CODE 'TSK02'.
```

Online DML statements, which request CA IDMS to perform data communications services, are grouped into the following categories:

- **Program management statements** govern the flow of control and abend processing.
- **Storage management statements** allocate and release variable storage.
- **Task management statements** provide runtime services that enhance control over task processing.
- **Time management statements** obtain the time and date, and define time-related events.
- **Scratch management statements** create, delete, or retrieve records from the scratch area.
- **Queue management** statements create, delete, or retrieve records in a queue area.
- **Terminal management statements** transfer data between the application program and a terminal.
- **Utility function statements** retrieve task-related information or statistics, send messages, and monitor access to database records.
- **Recovery statements** perform functions relating to database, scratch, and queue area recovery when a system failure occurs.
Compiling and Executing CA IDMS Programs (COBOL)

A CA IDMS COBOL source program contains DML statements for the precompiler to process. The precompiler converts DML statements into COBOL CALL statements and copies information from the data dictionary into the application program. After successful compilation and link editing, the application program can be executed.

Compiling Programs

Three components prepare a COBOL DML program for execution: the precompiler, the COBOL compiler, and the linkage editor:

1. The precompiler converts the DML statements to COBOL CALL statements and copies information from the data dictionary into the application program. For example, the precompiler can copy database record descriptions, file definitions, map records, map definitions, and other predefined modules such as the IDMS communications block into the program. The precompiler generates an output file that serves as input to the COBOL precompiler and as an optional source listing. The output file differs from the source input to the precompiler in the following ways:
   - Source code (such as the IDMS communications block and the IDMS-STATUS routine) has been added to the program.
   - DML statements have been replaced by COBOL CALL statements and have been changed to comment entries (asterisk in column 7).

   The precompiler also produces a listing of the following errors:
   - Incorrect DML entries
   - Statements that are inconsistent with the declared subschema view of the program
   - Any other error conditions that are detected during DML processing
   - Warning messages indicating source code conditions that could adversely affect the run units that are using the program

2. The COBOL compiler compiles the source program after the precompiler has successfully processed it. The COBOL compiler output consists of an object program and a source listing that includes any generated diagnostics.

3. The linkage editor link edits the object program into a specified load library. The linkage editor output consists of a load module (or phase) and a link map.

The job control language that is required to execute each step is listed in DML Precompile, COBOL Compile, and Link-Edit JCL (see page 280).

The following figure shows the component steps to prepare a COBOL DML program for execution:
Executing Programs

At run time, CA IDMS requests are treated as application program subroutine calls. When the subroutine call is executed, control passes to the DBMS or to the DC system, which processes the requested function.

A CA IDMS program must be defined to the CA IDMS system in which it operates. The program can be defined at system generation or at run time by using a DCMT VARY DYNAMIC PROGRAM command.

The following COBOL features are not used in programs running in an online environment under CA IDMS:

- ENVIRONMENT and DATA DIVISION entries that are typically associated with file management (for example, INPUT-OUTPUT SECTION, FILE SECTION).

- The Report Writer and Segmentation features, and features that are invoked by the SORT, EXHIBIT, TRACE, DISPLAY, ACCEPT, STRING, UNSTRING, and INSPECT commands. The EXAMINE and TRANSFORM verbs are not valid verbs in COBOL-II because their logic has been incorporated into the INSPECT verb. Also, the EXTERNAL clause of the record statement is invalid for all COBOL versions.

**Note:** The INSPECT, STRING, and UNSTRING commands are restricted because they issue supervisor calls in some environments. This restriction applies mainly when running VS COBOL on a VSE or z/OS operating system. This restriction does not apply when using LE-compliant COBOL with the IBM runtime Language Environment. See **VS COBOL II Support (see page 391)** for considerations when using these commands with VS/COBOL II.

- The READ, WRITE, OPEN, and CLOSE I/O statements.

- The COBOL compiler DEBUG option.
The COUNT, FLOW, STATE, ENDJOB, TEST, RESIDENT, DYNAM, and SYMDMP commands (OS only).

**Note:** The TEST compile option can be used for a program that was compiled by a Language Environment compliant compiler. However, the DEBUG runtime option must not be used in the online environment. A load module that is compiled with the TEST option can be run with the DEBUG runtime option in batch. Running the load module like this lets you test a load module in a batch environment and then run the module in an online environment without recompiling it.

The COUNT, FLOW, STATE, STXIT, and SYMDMP commands (z/VSE only).

- Any feature that can lead to the issuance of a supervisor call (SVC). Supervisor calls can inhibit system performance and crash the DC system.

Usage of the DBCS COBOL compile-time option by a CA IDMS program can lead to IGYPS0156-E runtime errors. Programs using LRF are especially susceptible to this problem. This option is the default for LE/COBOL compilers, starting with z/OS 3.2.0. Compile CA IDMS programs using the NODBCS option.

The COBOL compiler provides a TRUNC option. Use this option with care in any CA IDMS program that accesses database key values. Exercise particular care when arithmetic operations are performed on the database key, for example multiplying a page number by the database key radix. If an inappropriate TRUNC value is specified, unexpected results can occur because COBOL can truncate numeric values to fit the PICTURE clause. Avoid this problem by using the compiler option TRUNC (BIN). On some older COBOL compilers, the equivalent option is NOTRUNC.

**Note:** For more information about the TRUNC option, see the documentation of your compiler vendor.

Callable Services and Common Facilities (COBOL)

CA IDMS provides callable services and common facilities to use with your application programs.

The callable services include:

- The **IDMSCALC** ([https://docops.ca.com/display/IDMSCU19/IDMSCALC](https://docops.ca.com/display/IDMSCU19/IDMSCALC)) utility that lets you sort input into target page sequence.

- The **IDMSIN01** ([https://docops.ca.com/display/IDMSCU19/IDMSIN01](https://docops.ca.com/display/IDMSCU19/IDMSIN01)) facility that lets you perform miscellaneous CA IDMS functions.

- The **TCP/IP** ([https://docops.ca.com/pages/viewpage.action?pageId=357362491](https://docops.ca.com/pages/viewpage.action?pageId=357362491)) socket program interface that lets you communicate with another TCP/IP application.
The common facilities include:

- The Command Facility that lets you submit command statements in a batch or online environment.

- The Online Compiler Text Editor that lets you edit compiler output and resubmit it as input by using the CA IDMS development tools.

- The Transfer Control Facility that lets you transfer between CA IDMS development tools.

- The SYSIDMS parameter file that contains parameters that you can add to a batch job running in local mode or under the central version. These parameters let you specify environment requirements, runtime directives, and operating system-dependent information.

⚠️ **Note:** For more information about using these common facilities and the SYSIDMS parameter file, see Using Common Facilities (https://docops.ca.com/display/IDMS19/Using+Common+Facilities).
Precompiler Options

This section contains the syntax for COBOL precompiler options that are included as special format entries in the COBOL source code input to the precompiler.

- Dictionary Ready Override (see page 32)
- Comment Generation (see page 33)
- List Generation (see page 33)
- Log Suppression (see page 34)

These options are beneficial for the following uses:

- Override the DDLDML area default usage mode
- Enable the printing of data dictionary and subschema comments
- Control the generation of precompiler source listings
- Suppress the logging of program activity statistics

Dictionary Ready Override

When the DDLDML area of the data dictionary (that is, the main area of the dictionary accessed by the precompiler) is readied, a number of different options are available. The default mode used is shared update. Shared update mode readies the DDLDML area for both retrieval and update and allows other concurrently executing run units to ready the DDLDML area in shared update or shared retrieval usage mode. An application program can override the default usage mode by specifying either retrieval or protected update usage.

Syntax

```
Begin in column 7.

*RETRIEVAL
*PROTECTED-UPDATE
```

Parameters

- *RETRIEVAL
  Readies the DDLDML area for retrieval only and allows other concurrently executing run units to open the area in shared retrieval, shared update, protected retrieval, or protected update usage modes.

⚠️ Note: If the DDLDML area is readied for retrieval only, no program activity statistics can be logged.
• **PROTECTED-UPDATE**
  Readies the DDLDML area for both retrieval and update and allows other concurrently executing run units to ready the area in retrieval usage mode only. The protected update usage mode prevents concurrent update of the area by run units executing under the same central version.

> **Note:** If used, the dictionary ready override statement must precede all source input statements, except for PROCESS (CBL) statements that are used to override Enterprise COBOL compiler options and must be entered first.

### Comment Generation

The *SCHEMA-COMMENTS option causes schema-defined data-item comments and IDD-defined record-element comments in the data dictionary to be printed on the precompiler source listing and inserted into the program.

**Syntax**

```plaintext
*SCHEMA-COMMENTS
```

Code the *SCHEMA-COMMENTS statement after the dictionary ready override statements (if any) and before any DML or COBOL statements: If the *SCHEMA-COMMENTS statement is not included with the input, comment lines are not generated.

### List Generation

The source statement listing output by the precompiler can be turned on or off by inserting a list generation option into the source program.

**Syntax**

```plaintext
*DMLIST
*NODMLIST
```

**Parameters**

- **DMLIST**
  Specifies that the source listing is to be generated for the statements that follow.

- **NODMLIST**
  Specifies that no source listing is to be generated for the statements that follow.

This is the default.
Generation of the list can be turned on or off any number of times within one source program by inserting appropriate *DMLIST and/or *NODMLIST entries in the code.

**Note:** A listing of error messages is always produced. The *DMLIST option controls output of the processor source listing.

## Log Suppression

The *NO-ACTIVITY-LOG option suppresses the logging of program activity statistics. The precompiler generates and logs the following program activity statistics unless the *NO-ACTIVITY-LOG option is specified:

- Program name
- Language
- Date last compiled
- Number of lines
- Number of compilations
- Date created
- Subschema name (if any)
- File statistics
- Database access statistics (records and modules copied from the data dictionary; subprograms called; and records, sets, and areas accessed by DML verbs)

### Syntax

```
*NO-ACTIVITY-LOG
```

The *NO-ACTIVITY-LOG statement follows the NODENAME/DBNAME and dictionary ready override statements.

**Note:** Program activity statistics cannot be logged if the DDLDML area is readied for retrieval only.
Communications Blocks and Error Detection (COBOL)

This section describes the communications blocks that are available under CA IDMS. These blocks return status information about the requested database and data communications services to the application program. This section also describes the ERROR-STATUS field in the IDMS and IDMS-DC communications blocks, status codes, and error detection routines.

- 16-Byte Communications Blocks (COBOL) (see page 35)
- 18-Byte Communications Blocks (COBOL) (see page 45)
- ERROR-STATUS Field and Codes (see page 47)
- Error Detection (COBOL) (see page 56)

16-Byte Communications Blocks (COBOL)

Communications blocks return status information about the requested database and data communications services to the application program. The usage mode that is defined in the subschema (LR, DML, or MIXED) determines which blocks your program uses:

- The IDMS communications block (see page 35) is used when the operating mode is BATCH or BATCH-AUTOSTATUS.
- The logical-record request control (LRC) block (see page 43) is used when the subschema usage mode is LR or MIXED. The LRC block is copied in with the IDMS communications block or the IDMS-DC communications block, depending on the operating mode.
- The IDMS-DC communications block (see page 40) is used when the operating mode is IDMS-DC or DC-BATCH.

Note: For more information about operating modes and protocols, see Precompiler-Directive Statements (see page 63).

IDMS Communications Block (COBOL)

The IDMS communications block is used when the operating mode is BATCH or BATCH-AUTOSTATUS. This block is an interface between the database management system (DBMS) and the application program. Whenever a run unit issues a call to the DBMS for a database operation, the DBMS returns information about the outcome of the requested service to the IDMS communications block of the application program.
The data description (identified as SUBSCHEMA-CTRL) of the IDMS communications block is copied from the data dictionary into the WORKING-STORAGE SECTION or LINKAGE SECTION of the program. When you submit the program to the precompiler, the IDMS communications block is copied automatically unless an IDMS-RECORDS MANUAL statement is included in the ENVIRONMENT DIVISION. In that case, the program can explicitly call in the data description by using a COPY IDMS SUBSCHEMA-CTRL statement.

Note: For more information about the IDMS-RECORDS MANUAL and the COPY IDMS statements, see Precompiler-Directive Statements (see page 63).

Examine the ERROR-STATUS field of the IDMS communications block after every call to the DBMS. The value of this field determines whether you should perform the IDMS-STATUS routine (see IDMS-STATUS Routine (see page 57)). For example, if the ERROR-STATUS value is 0307 (DB-END-OF-SET) while walking a set, perform end-of-set processing. For any other value, perform IDMS-STATUS.

The following figure shows the layout of the 16-byte IDMS communications block.

<table>
<thead>
<tr>
<th>Field</th>
<th>Data Type</th>
<th>Length (bytes)</th>
<th>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</td>
<td>Alphanumeric</td>
<td>4</td>
<td>'1400'</td>
</tr>
<tr>
<td>DBKEY</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</td>
<td>Alphanumeric</td>
<td>16</td>
<td>Spaces</td>
</tr>
<tr>
<td>ERROR-RECORD</td>
<td>Alphanumeric</td>
<td>16</td>
<td>Spaces</td>
</tr>
<tr>
<td>ERROR-AREA</td>
<td>Alphanumeric</td>
<td>16</td>
<td>Spaces</td>
</tr>
<tr>
<td>IDBMS-AREA</td>
<td>Alphanumeric</td>
<td>100</td>
<td>Low Values</td>
</tr>
<tr>
<td>DIRECT-DBKEY</td>
<td>Binary</td>
<td>4(Fullword)</td>
<td>0000</td>
</tr>
<tr>
<td>DATABASE-STATUS</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-OCCUR</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
** PAGE-INFO-GROUP overlays bytes 97 and 98 and PAGE-INFO-DBK-FORMAT overlays bytes 99 and 100. Both of these fields are binary datatype, each with a length of two bytes. Suggested initial values for both are 00. Together these two fields represent PAGE-INFO.

The IDMS communications block contains the following fields:
<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRO</td>
<td>The name of the program being executed, as defined in the IDENTIFICATION DIVISION of the program.</td>
</tr>
<tr>
<td>GRAM-NAM</td>
<td>If the program contains a COPY IDMS SUBSCHEMA-BINDS statement in its PROCEDURE DIVISION, this field initializes automatically at the beginning of program execution. Otherwise, the programmer must initialize this field.</td>
</tr>
<tr>
<td>ERRO-STAT</td>
<td>An alphanumeric value indicating the outcome of the last DML statement execution. The program must initialize the ERROR-STATUS field to 1400. The DBMS updates the ERROR-STATUS field after performing a requested database service and before returning control to the program. A program with more than one run unit must reinitialize the ERROR-STATUS field to 1400 after finishing one run unit and before binding the next.</td>
</tr>
<tr>
<td>DBKEY</td>
<td>The database key of the last record that the run unit accessed. For example, after successful execution of a FIND command, DBKEY is updated with the database key of the located record. DBKEY is not changed if the call to the DBMS results in a nonzero status condition.</td>
</tr>
<tr>
<td>RECO</td>
<td>The last record that the run unit accessed successfully. This field is left-justified and padded with spaces on the right.</td>
</tr>
<tr>
<td>RD-NAM</td>
<td>This field is left-justified and padded with spaces on the right.</td>
</tr>
<tr>
<td>AREA</td>
<td>The last area that the run unit accessed successfully. This field is left-justified and padded with spaces on the right.</td>
</tr>
<tr>
<td>ERRO-SET</td>
<td>The set that was involved in the last operation to produce a nonzero status code. This field is left-justified and padded with spaces on the right.</td>
</tr>
<tr>
<td>ERRO</td>
<td>The record that was involved in the last operation to produce a nonzero status code. This field is left-justified and padded with spaces on the right.</td>
</tr>
<tr>
<td>RD-RECO</td>
<td>This field is left-justified and padded with spaces on the right.</td>
</tr>
<tr>
<td>AREA</td>
<td>This field is left-justified and padded with spaces on the right.</td>
</tr>
</tbody>
</table>
| PAGE   | Two binary halfwords that represent the page information that is associated with the last record that the run unit accessed. PAGE-INFO is not changed when the call to the DBMS results in a nonzero status. The first halfword (PAGE-INFO-GROUP) represents the page group number. The second halfword (PAGE-INFO-DBK-FORMAT) represents the database key radix. The database key radix can be used in interpreting a database key for display purposes and in formatting a database key from page and line numbers. The database key radix represents the number of bits within a database key value that are reserved for the line number of a record. By default, this value is 8, meaning that up to 255 records can be stored on a single page of the area. Given a database key, you can separate its associated page number by dividing the
database key by 2 raised to the power of the database key radix. For example, if the database key radix is 4, divide the database key value by 2**4. The resulting value is the page number of the database key. To separate the line number, multiply the page number by 2 raised to the power of the database key radix and subtract this value from the database key value. The result would be the line number of the database key. The following two formulas can be used to calculate the page and line numbers from a database key value:

\[
\text{Page-number} = \frac{\text{database key value}}{2^{\text{database key radix}}}
\]

\[
\text{Line-number} = \text{database key value} - (\text{page-number} \times 2^{\text{database key radix}})
\]

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IDBM</td>
<td>Used internally by the DBMS for specification of runtime function information.</td>
</tr>
<tr>
<td>SCO</td>
<td>M-AREA</td>
</tr>
<tr>
<td>DIRE</td>
<td>A user-specified database key value or a null database key value of -1.</td>
</tr>
<tr>
<td>CT-DBKE</td>
<td>This field is used for storing a record with a location mode of DIRECT. The user must initialize this field because it is not updated by the DBMS.</td>
</tr>
<tr>
<td>DATA</td>
<td>Reserved for use by the DBMS.</td>
</tr>
<tr>
<td>BASE</td>
<td>US</td>
</tr>
<tr>
<td>FILE</td>
<td>Used to ensure fullword alignment.</td>
</tr>
<tr>
<td>R</td>
<td></td>
</tr>
<tr>
<td>RECO</td>
<td>A record occurrence sequence identifier that is used internally by the DBMS.</td>
</tr>
<tr>
<td>RD-OCC</td>
<td>UR</td>
</tr>
<tr>
<td>DML-SEQUENCE</td>
<td>The source level sequence number that the precompiler generates. If DEBUG is specified in the SEQU ENVIRONMENT DIVISION of the program, this field is updated before each call to the DBMS.</td>
</tr>
<tr>
<td>ENCE</td>
<td>This field is not used by the runtime system.</td>
</tr>
</tbody>
</table>

**Native VSAM users:** The DIRECT-DBKEY field can be used only when storing a record in a native VSAM relative record data set (RRDS) or when storing records with DIRECT location mode. You must initialize DIRECT-DBKEY to the relative record number of the record being stored.

After a call is made to the DBMS, one or more communications block fields can be updated, depending on which DML statement is issued and whether the statement executed successfully. The following figure illustrates the IDMS communications block fields updated by successful and unsuccessful calls to the DBMS. Only those fields that the runtime system accessed are shown. Fields that the DBMS used internally are not shown. Blank fields are not updated by DML statements.
## Control Statements

<table>
<thead>
<tr>
<th>Statement</th>
<th>O/S</th>
<th>Name</th>
<th>ORDS</th>
<th>NAME</th>
<th>DBKEY</th>
<th>ORDS</th>
<th>NAME</th>
<th>DBKEY</th>
</tr>
</thead>
<tbody>
<tr>
<td>BINDRUN-UNIT</td>
<td>0</td>
<td>14nn</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BINDRECORD</td>
<td>0</td>
<td>14nn</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BINDPROCEDURE</td>
<td>0</td>
<td>14nn</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>READY</td>
<td>0</td>
<td>09nn</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FINISH</td>
<td>0 N C</td>
<td>01nn</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>COMMIT(ALL)</td>
<td>0 N C</td>
<td>18nn</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ROLLBAK(CONTINUE)</td>
<td>0 N C</td>
<td>19nn</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>KEEP(EXCLUSIVE)</td>
<td>0 Y Y</td>
<td>06nn</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IFSET</td>
<td>* Y</td>
<td>16nn</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IFNOTSET</td>
<td>* Y</td>
<td>16nn</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

## Retrieval Statements

<table>
<thead>
<tr>
<th>Statement</th>
<th>O/S</th>
<th>Name</th>
<th>ORDS</th>
<th>NAME</th>
<th>DBKEY</th>
<th>ORDS</th>
<th>NAME</th>
<th>DBKEY</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIND/OBtainRECORD</td>
<td>0 Y Y</td>
<td>03nn</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GETRECORD</td>
<td>0 Y Y</td>
<td>05nn</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RETURNRECORD</td>
<td>0 Y Y</td>
<td>17nn</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

## Modification Statements

<table>
<thead>
<tr>
<th>Statement</th>
<th>O/S</th>
<th>Name</th>
<th>ORDS</th>
<th>NAME</th>
<th>DBKEY</th>
<th>ORDS</th>
<th>NAME</th>
<th>DBKEY</th>
</tr>
</thead>
<tbody>
<tr>
<td>STORERECORD</td>
<td>0 Y Y</td>
<td>12nn</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CONNECTRECORD</td>
<td>0 Y Y</td>
<td>07nn</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MODIFYRECORD</td>
<td>0 Y Y</td>
<td>08nn</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DISCONNECTRECORD</td>
<td>0 Y Y</td>
<td>11nn</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ERASERECORD</td>
<td>0 N Y</td>
<td>02nn</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

## Accept Statements

<table>
<thead>
<tr>
<th>Statement</th>
<th>O/S</th>
<th>Name</th>
<th>ORDS</th>
<th>NAME</th>
<th>DBKEY</th>
<th>ORDS</th>
<th>NAME</th>
<th>DBKEY</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACCEPTDBKEYOFCurrency</td>
<td>0 C C</td>
<td>15nn</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ACCEPTDBKEYOFN/P/OCURRENCY</td>
<td>0 C C</td>
<td>15nn</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ACCEPTIDMSSTATISTICS</td>
<td>0 C C</td>
<td>15nn</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ACCEPTBINDRECORD</td>
<td>0 C C</td>
<td>15nn</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ACCEPTPROCEDURE</td>
<td>0 C C</td>
<td>15nn</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ACCEPTpage-info-location</td>
<td>0 C C</td>
<td>15nn</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
IDMS-DC Communications Block (COBOL)

The IDMS DC communications block replaces the IDMS communications block when the operating mode is IDMS-DC or DC-BATCH. At run time, the IDMS-DC communications block is used to pass information about the outcome of requested data communications and database services to an application program.

The data description (identified as SUBSCHEMA-CTRL) of the IDMS-DC communications block is copied from the data dictionary into the WORKING-STORAGE SECTION or LINKAGE SECTION of the program. When the program is submitted to the precompiler, the IDMS-DC communications block is copied automatically unless the IDMS-RECORDS MANUAL statement is included in the ENVIRONMENT DIVISION. In that case, the program can explicitly call in the data description by using a COPY IDMS SUBSCHEMA-CTRL statement. (For more information about the IDMS-RECORDS MANUAL and the COPY IDMS statements, see Precompiler-Directive Statements (see page 63).)

The following figure shows the layout of the IDMS-DC communications block.

```
<table>
<thead>
<tr>
<th>Field</th>
<th>Data Type</th>
<th>(bytes)</th>
<th>Initial Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>PROGRAM-NAME</td>
<td>Alphanumeric</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>ERROR-STATUS</td>
<td>Alphanumeric</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>DBKEY</td>
<td>Binary</td>
<td>4(Fullword) 0000</td>
</tr>
<tr>
<td></td>
<td>RECORD-NAME</td>
<td>Alphanumeric</td>
<td>16</td>
</tr>
<tr>
<td></td>
<td>AREA-NAME</td>
<td>Alphanumeric</td>
<td>16</td>
</tr>
<tr>
<td></td>
<td>ERROR-SET</td>
<td>Alphanumeric</td>
<td>16</td>
</tr>
<tr>
<td></td>
<td>ERROR-RECORD</td>
<td>Alphanumeric</td>
<td>16</td>
</tr>
<tr>
<td></td>
<td>ERROR-AREA</td>
<td>Alphanumeric</td>
<td>16</td>
</tr>
<tr>
<td>**</td>
<td>PAGE-INFO</td>
<td>Binary</td>
<td>4(Fullword) 0000</td>
</tr>
<tr>
<td></td>
<td>IDBMSCOM-AREA</td>
<td>Alphanumeric</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td>DIRECT-DBKEY</td>
<td>Binary</td>
<td>4(Fullword) 0000</td>
</tr>
</tbody>
</table>
```
Field Descriptions

The IDMS-DC communications block contains the following fields:

<table>
<thead>
<tr>
<th>Field name</th>
<th>Position</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRO GRA M- NAM</td>
<td>1-8</td>
<td>The program being executed, as defined in the IDENTIFICATION DIVISION of the program.</td>
</tr>
<tr>
<td>ERRO R- STAT US</td>
<td>9-12</td>
<td>A value indicating the outcome of the last DML statement. The program must initialize the ERROR-STATUS field to 1400.</td>
</tr>
<tr>
<td>DBKEY</td>
<td>13-16</td>
<td>The database key of the last record that the run unit accessed. For example, after a FIND command executes successfully, DBKEY is updated with the database key of the located record. DBKEY is not changed when the database call results in a nonzero status condition.</td>
</tr>
<tr>
<td>RECO RD- NAM E</td>
<td>17-32</td>
<td>The last record that the run unit accessed. This field is left-justified and padded with spaces on the right.</td>
</tr>
<tr>
<td>AREA R- NAM E</td>
<td>33-48</td>
<td>The last area that the run unit accessed successfully. This field is left-justified and padded with spaces on the right.</td>
</tr>
<tr>
<td>49-64</td>
<td>The set that was involved in the last operation to produce a nonzero status code.</td>
<td></td>
</tr>
<tr>
<td>Field name</td>
<td>Position</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>----------</td>
<td>-------------</td>
</tr>
<tr>
<td>ERRO R-SET</td>
<td>65-80</td>
<td>The record that was involved in the last operation to produce a nonzero status code. This field is left-justified and padded with spaces on the right.</td>
</tr>
<tr>
<td>ERRO 81-96 R-AREA</td>
<td>97-100</td>
<td>The area that was involved in the last operation to produce a nonzero status code. This field is left-justified and padded with spaces on the right.</td>
</tr>
<tr>
<td>PAGE 97-100 INFO</td>
<td>Two binary halfwords that represent the page information that is associated with the last record that the run unit accessed. PAGE-INFO is not changed when the call to the DBMS results in a nonzero status. The first halfword (PAGE-INFO-GROUP) represents the page group number. The second halfword (PAGE-INFO-DBK-FORMAT) represents the database key radix.</td>
<td></td>
</tr>
<tr>
<td>IDBM SCO M-AREA</td>
<td>97-196</td>
<td>Used internally by CA IDMS for specification of DBMS runtime function information.</td>
</tr>
<tr>
<td>DIRECT-DBKEY</td>
<td>197-200</td>
<td>A user-specified database key value or a null database key value of -1. This field is used for storing a record with a location mode of DIRECT. The user must initialize this field because CA IDMS does not update it.</td>
</tr>
<tr>
<td>DCB MSC OM-AREA</td>
<td>201-300</td>
<td>Used internally by CA IDMS for specification of runtime function information.</td>
</tr>
</tbody>
</table>

The database key radix portion can be used in interpreting a database key for display purposes and in formatting a database key from page and line numbers. The database key radix represents the number of bits within a database key value that are reserved for the line number of a record. By default, this value is 8, meaning that up to 255 records can be stored on a single page of the area. Given a database key, you can separate its associated page number by dividing the database key by 2 raised to the power of the database key radix. For example, if the database key radix is 4, divide the database key value by 2**4. The resulting value is the page number of the database key. To separate the line number, multiply the page number by 2 raised to the power of the database key radix and subtract this value from the database key value. The result would be the line number of the database key. The following two formulas can be used to calculate the page and line numbers from a database key value:

\[
\text{Page-number} = \frac{\text{database key value}}{2^{\text{database key radix}}} \\
\text{Line-number} = \text{database key value} - (\text{page-number} \times 2^{\text{database key radix}})
\]

Native VSAM users: The DIRECT-DBKEY field can be used when storing a record in a native VSAM relative record data set (RRDS). Initialize DIRECT-DBKEY to the relative record number of the record being stored.
<table>
<thead>
<tr>
<th>Field name</th>
<th>Position</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSC-ERRS</td>
<td>301-304</td>
<td>Used by the IDMS-STATUS routine to save a nonzero ERROR-STATUS when an abend occurs.</td>
</tr>
<tr>
<td>TAT-SAVE</td>
<td>305-308</td>
<td>Used by the IDMS-STATUS routine to save the value of DML-SEQUENCE when an abend occurs.</td>
</tr>
<tr>
<td>DML-SEQU</td>
<td>309-312</td>
<td>The source level sequence number generated by the precompiler. If DEBUG is specified in the ENVIRONMENT DIVISION of the program, this field is updated before each call to CA IDMS. The runtime system does not use this field.</td>
</tr>
<tr>
<td>ENCE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RECO-OC</td>
<td>313-316</td>
<td>A record occurrence sequence identifier that is used internally by CA IDMS.</td>
</tr>
<tr>
<td>RD-UR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SUBS-CHE</td>
<td>317-320</td>
<td>Marks the end of the IDMS-DC communications block.</td>
</tr>
<tr>
<td>MA-CTRL-</td>
<td></td>
<td></td>
</tr>
<tr>
<td>END</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Logical-Record Request Control (LRC) Block (COBOL)**

The logical-record request control (LRC) block is used when the subschema usage mode is LR or MIXED. The LRC block, which is used with the IDMS or IDMS-DC communications block, provides an interface between LRF and the application program. The LRC block passes information about a logical-record request to LRF and returns path status information about the processing of the request to the program.

The data description (identified as SUBSCHEMA-LR-CTRL) of the LRC block is copied from the data dictionary into the WORKING-STORAGE SECTION or LINKAGE SECTION of the program. When the program is submitted to the precompiler, the LRC block and the IDMS or IDMS-DC communications block are copied automatically, unless the IDMS-RECORDS MANUAL statement is included in the ENVIRONMENT DIVISION. In that case, both descriptions can be called in explicitly by the program by using a COPY IDMS SUBSCHEMA-LR-CTRL statement.

For more information about the IDMS-RECORDS MANUAL and the COPY IDMS statements, see Precompiler-Directive Statements (see page 63).

Examine the LR-STATUS field of the LRC block for all possible statuses after every call to LRF. If the value returned is LR-ERROR, examine the ERROR-STATUS field of the IDMS or IDMS-DC communications block.

The following figure shows the layout of the LRC block.
The LRC block contains the following fields:

<table>
<thead>
<tr>
<th>Field name</th>
<th>Position</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LRC-LRPXELNG</td>
<td>1-2</td>
<td>Specifies the length of the LRC block.</td>
</tr>
<tr>
<td>LRC-MAXVXP</td>
<td>3-4</td>
<td>Specifies the length of the work area that is required to evaluate the WHERE clause.</td>
</tr>
<tr>
<td>LRIDENT</td>
<td>5-8</td>
<td>The constant 'LRC' followed by a space.</td>
</tr>
<tr>
<td>LRVERB</td>
<td>9-16</td>
<td>The verb that is passed to LRF.</td>
</tr>
<tr>
<td>LRNAME</td>
<td>17-32</td>
<td>The name of the logical record being accessed.</td>
</tr>
<tr>
<td>LR-STATUS</td>
<td>33-48</td>
<td>The path status of a logical-record request.</td>
</tr>
<tr>
<td>FILLER</td>
<td>49-64</td>
<td>Used internally by LRF.</td>
</tr>
<tr>
<td>PXE</td>
<td>65-end</td>
<td>The variable-length expansion of the WHERE clause. From 0 to 512 1-byte elements.</td>
</tr>
</tbody>
</table>

Path statuses are 1- to 16-character strings. These statuses can be standard or can be defined in the subschema by the DBA. LRF provides three standard path statuses: LR-FOUND, LR-NOT-FOUND, and LR-ERROR.

**Note:** For more information about path statuses, see the Logical-Record Clauses (COBOL) (see page 272).
### 18-Byte Communications Blocks (COBOL)

Instead of using the 16-byte IDMS and IDMS-DC communications blocks, you can specify 18-byte blocks. These blocks contain an extra 18-byte filler field, and the following fields are 18 bytes instead of 16 bytes:

- RECORD-NAME
- AREA-NAME
- ERROR-SET
- ERROR-RECORD
- ERROR-AREA

For COBOL, specify an 18-byte communications block in the SUBSCHEMA-NAMES LENGTH IS clause. This clause is found in the PROTOCOL statement of ENVIRONMENT DIVISION.

### 18-Byte IDMS Block

The following figure shows the 18-byte IDMS communications block:

<table>
<thead>
<tr>
<th>Field</th>
<th>Data Type</th>
<th>Length (bytes)</th>
<th>Initial Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-8</td>
<td>PROGRAM-NAME</td>
<td>Alphanumeric 8</td>
<td>Program Name</td>
</tr>
<tr>
<td>9-12</td>
<td>ERROR-STATUS</td>
<td>Alphanumeric 4</td>
<td>'1400'</td>
</tr>
<tr>
<td>13-16</td>
<td>DBKEY</td>
<td>Binary</td>
<td>0000</td>
</tr>
<tr>
<td>17-34</td>
<td>RECORD-NAME</td>
<td>Alphanumeric 18</td>
<td>Spaces</td>
</tr>
<tr>
<td>35-52</td>
<td>AREA-NAME</td>
<td>Alphanumeric 18</td>
<td>Spaces</td>
</tr>
<tr>
<td>53-70</td>
<td>FILLER</td>
<td>Alphanumeric 18</td>
<td>Spaces</td>
</tr>
<tr>
<td>71-88</td>
<td>ERROR-SET</td>
<td>Alphanumeric 18</td>
<td>Spaces</td>
</tr>
<tr>
<td>89-106</td>
<td>ERROR-RECORD</td>
<td>Alphanumeric 18</td>
<td>Spaces</td>
</tr>
<tr>
<td>107-124</td>
<td>ERROR-AREA</td>
<td>Alphanumeric 18</td>
<td>Spaces</td>
</tr>
<tr>
<td>125-128</td>
<td>PAGE-INFO</td>
<td>Binary</td>
<td>0000</td>
</tr>
<tr>
<td>125-224</td>
<td>IDBMSCOM-AREA</td>
<td>Alphanumeric 100</td>
<td>Low Values</td>
</tr>
<tr>
<td>225-228</td>
<td>DIRECT-DBKEY</td>
<td>Binary</td>
<td>0000</td>
</tr>
<tr>
<td>229-235</td>
<td>DATABASE-STATUS</td>
<td>Alphanumeric 7</td>
<td>Spaces</td>
</tr>
<tr>
<td>236</td>
<td>FILLER</td>
<td>... 1</td>
<td>...</td>
</tr>
</tbody>
</table>
**18-Byte IDMS-DC Block**

The following figure shows the 18-byte IDMS-DC communications block:

<table>
<thead>
<tr>
<th>Field</th>
<th>Data Type</th>
<th>Length (bytes)</th>
<th>Initial Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 - 8</td>
<td>Alphanumeric</td>
<td>8</td>
<td>Program Name</td>
</tr>
<tr>
<td>9 - 12</td>
<td>Alphanumeric</td>
<td>4</td>
<td>'1400'</td>
</tr>
<tr>
<td>13 - 16</td>
<td>Binary</td>
<td>4 (Fullword)</td>
<td>0000</td>
</tr>
<tr>
<td>17 - 34</td>
<td>Alphanumeric</td>
<td>18</td>
<td>Spaces</td>
</tr>
<tr>
<td>35 - 52</td>
<td>Alphanumeric</td>
<td>18</td>
<td>Spaces</td>
</tr>
<tr>
<td>53 - 70</td>
<td>Alphanumeric</td>
<td>18</td>
<td>Spaces</td>
</tr>
<tr>
<td>71 - 88</td>
<td>Alphanumeric</td>
<td>18</td>
<td>Spaces</td>
</tr>
<tr>
<td>89 - 106</td>
<td>Alphanumeric</td>
<td>18</td>
<td>Spaces</td>
</tr>
<tr>
<td>107 - 124</td>
<td>Alphanumeric</td>
<td>18</td>
<td>Spaces</td>
</tr>
</tbody>
</table>

**PAGE-INFO**

<table>
<thead>
<tr>
<th>Field</th>
<th>Data Type</th>
<th>Length (bytes)</th>
<th>Initial Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>125 - 128</td>
<td>Binary</td>
<td>4 (Fullword)</td>
<td>0000</td>
</tr>
</tbody>
</table>

**PAGE-INFO-GROUP overlays bytes 125 and 126 and PAGE-INFO-DBK-FORMAT overlays bytes 127 and 128. Both of these fields are binary datatype, each with a length of two bytes. Suggested initial values for both are 00. Together these two fields represent PAGE-INFO.
**ERROR-STATUS Field and Codes**

You can use the ERROR-STATUS field of the IDMS or IDMS-DC communications block to determine whether a DML request completed successfully. The DBMS or the DC system returns a value to the ERROR-STATUS field indicating the result of each DML request. For more information about the ERROR-STATUS field, see [Error Detection (COBOL)](see page 56).

**LRF users:** Review the LR-STATUS field of the LRC block before checking the ERROR-STATUS field.

**Major and Minor Codes**

The ERROR-STATUS field is a 4-byte zoned decimal field. The first two bytes represent a major code. The second two bytes represent a minor code. Major codes identify the function performed. Minor codes describe the status of that function.

**Code Values**

A value of 0000 indicates successful completion of the requested function. Any other value indicates completion of the function in a manner that might be in error, depending on your expectations. For example, 0326 (DB-REC-NOT-FOUND) is appropriate after FIND CALC retrieval because this code allows you to trap the condition and continue processing.

**DB status codes (see page 47)** have a major code from 01 to 20. These codes occur during database access in batch or online processing.

**DC status codes (see page 52)** have a major code from 30 to 51. These codes occur in online or DC-BATCH processing. Status codes with a major code of 00 apply to all DML functions.

**DB Status Codes (COBOL)**

The following tables list DB major and minor codes and their meanings.

### Major DB Status Codes

<table>
<thead>
<tr>
<th>Major Code</th>
<th>Database Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>Any DML statement</td>
</tr>
<tr>
<td>01</td>
<td>FINISH</td>
</tr>
<tr>
<td>02</td>
<td>ERASE</td>
</tr>
<tr>
<td>03</td>
<td>FIND/OBTAIN</td>
</tr>
<tr>
<td>05</td>
<td>GET</td>
</tr>
<tr>
<td>06</td>
<td>KEEP</td>
</tr>
<tr>
<td>07</td>
<td>CONNECT</td>
</tr>
<tr>
<td>08</td>
<td>MODIFY</td>
</tr>
<tr>
<td>09</td>
<td>READY</td>
</tr>
<tr>
<td>11</td>
<td>DISCONNECT</td>
</tr>
</tbody>
</table>
### Major Code

<table>
<thead>
<tr>
<th>Major Code</th>
<th>Database Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>STORE</td>
</tr>
<tr>
<td>14</td>
<td>BIND</td>
</tr>
<tr>
<td>15</td>
<td>ACCEPT</td>
</tr>
<tr>
<td>16</td>
<td>IF</td>
</tr>
<tr>
<td>17</td>
<td>RETURN</td>
</tr>
<tr>
<td>18</td>
<td>COMMIT</td>
</tr>
<tr>
<td>19</td>
<td>ROLLBACK</td>
</tr>
<tr>
<td>20</td>
<td>LRF requests</td>
</tr>
</tbody>
</table>

### Minor DB Status Codes

<table>
<thead>
<tr>
<th>Minor Code</th>
<th>Database Function Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>When this code is combined with a major code of 00, it indicates successful completion of the DML operation. When this code is combined with a nonzero major code, it indicates that the DML operation was not completed successfully due to central version causes, such as time-outs and program checks.</td>
</tr>
<tr>
<td>01</td>
<td>An area has not been readied. When this code is combined with a major code of 16, an IF operation has resulted in a valid false condition.</td>
</tr>
<tr>
<td>02</td>
<td>The database key that is used with a FIND/OBTAIN DB-KEY statement or the direct database key that was suggested for a STORE is not within the page range for the specified record name.</td>
</tr>
<tr>
<td>03</td>
<td>Invalid currency for the named record, set, or area. This code occurs only when a run unit is sharing a transaction with other database sessions. The 03 minor status is returned when the run unit tries to retrieve or update a record by using a currency that has been invalidated because of changes that were made by another database session that is sharing the same transaction.</td>
</tr>
<tr>
<td>04</td>
<td>The occurrence count of a variably occurring element has been specified as less than zero or greater than the maximum number of occurrences that are defined in the control element.</td>
</tr>
<tr>
<td>05</td>
<td>The specified DML function would have violated a duplicates-not-allowed option for a CALC, sorted, or index set.</td>
</tr>
<tr>
<td>06</td>
<td>No currency has been established for the named record, set, or area.</td>
</tr>
<tr>
<td>07</td>
<td>The end of a set, area, or index has been reached or the set is empty.</td>
</tr>
<tr>
<td>08</td>
<td>The specified record, set, procedure, or LR verb is not in the subschema or the specified record is not a member of the set.</td>
</tr>
<tr>
<td>09</td>
<td>The area has been readied with an incorrect usage mode.</td>
</tr>
<tr>
<td>10</td>
<td>An existing access restriction or subschema usage prohibits execution of the specified DML function. For LRF users, the subschema in use allows access to database records only. When this code is combined with a major code of 00, it means that the program has attempted to access a database record, but the subschema in use allows access to logical records only.</td>
</tr>
<tr>
<td>11</td>
<td>The record cannot be stored in the specified area due to insufficient space.</td>
</tr>
<tr>
<td>12</td>
<td></td>
</tr>
</tbody>
</table>

---

15-Jan-2018
<table>
<thead>
<tr>
<th>Minor Code</th>
<th>Database Function Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>A current record of run unit has not been established or has been nullified by a previous ERASE statement.</td>
</tr>
<tr>
<td>14</td>
<td>The CONNECT statement cannot be executed. The requested record has been defined as a mandatory automatic member of the set.</td>
</tr>
<tr>
<td>15</td>
<td>The DISCONNECT statement cannot be executed because the requested record has been defined as a mandatory member of the set.</td>
</tr>
<tr>
<td>16</td>
<td>The record cannot be connected to a set of which it is already a member.</td>
</tr>
<tr>
<td>17</td>
<td>The transaction manager encountered an error.</td>
</tr>
<tr>
<td>18</td>
<td>The record has not been bound.</td>
</tr>
<tr>
<td>19</td>
<td>The transaction for the run unit was forced to back out.</td>
</tr>
<tr>
<td>20</td>
<td>The current record is not the same type as the specified record name.</td>
</tr>
<tr>
<td>21</td>
<td>Not all areas being used have been readied in the correct usage mode.</td>
</tr>
<tr>
<td>22</td>
<td>The specified record name is not currently a member of the specified set name.</td>
</tr>
<tr>
<td>23</td>
<td>The specified area name is not in the subschema or is not an extent area, or the specified record name has not been defined within the specified area name.</td>
</tr>
<tr>
<td>25</td>
<td>No currency has been established for the named set.</td>
</tr>
<tr>
<td>26</td>
<td>No duplicates exist for the named record or the record occurrences cannot be found.</td>
</tr>
<tr>
<td>28</td>
<td>The run unit has attempted to ready an area that has been readied previously.</td>
</tr>
<tr>
<td>29</td>
<td>The run unit has attempted to place a lock on a record that another run unit has locked. A deadlock results. Unless the run unit issued either a FIND/OBTAIN KEEP EXCLUSIVE or a KEEP EXCLUSIVE, the run unit is aborted.</td>
</tr>
<tr>
<td>30</td>
<td>An attempt has been made to erase the owner record of a nonempty set.</td>
</tr>
<tr>
<td>31</td>
<td>The retrieval statement format conflicts with the location mode of the record.</td>
</tr>
<tr>
<td>32</td>
<td>An attempt to retrieve a CALC/DUPLICATE record was unsuccessful. The CALC value in variable storage is not equal to the value of the CALC control element in the current record of run unit.</td>
</tr>
<tr>
<td>33</td>
<td>At least one set in which the record participates has not been included in the subschema.</td>
</tr>
<tr>
<td>40</td>
<td>The WHERE clause in an OBTAIN NEXT logical-record request is inconsistent with a previous OBTAIN FIRST or OBTAIN NEXT command for the same record. Previously specified criteria, such as reference to a key field, have been changed. A path status of LR-ERROR is returned to the LRC block.</td>
</tr>
<tr>
<td>41</td>
<td>The subschema contains no path that matches the WHERE clause in a logical-record request. A path status of LR-ERROR is returned to the LRC block.</td>
</tr>
<tr>
<td>42</td>
<td>An ON clause that is included in the path by the DBA specified return of the LR-ERROR path status to the LRC block. An error has occurred while processing the LRF request.</td>
</tr>
<tr>
<td>43</td>
<td></td>
</tr>
</tbody>
</table>

---
### Minor Code Database Function Status

<table>
<thead>
<tr>
<th>Minor Code</th>
<th>Database Function Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>44</td>
<td>The WHERE clause in a logical-record request does not supply a key element (sort key, CALC key, or db-key) expected by the path. A path status of LR-ERROR is returned to the LRC block.</td>
</tr>
<tr>
<td>45</td>
<td>During evaluation of a WHERE clause, a program check has been recognized because a subscript value is neither greater than 0 nor less than its maximum allowed value plus 1. A path status of LR-ERROR is returned to the LRC block unless the DBA has included an ON clause to override this action in the path.</td>
</tr>
<tr>
<td>46</td>
<td>A program check has revealed an arithmetic exception (for example: overflow, underflow, significance, divide) during evaluation of a WHERE clause. A path status of LR-ERROR is returned to the LRC block unless the DBA has included an ON clause to override this action in the path.</td>
</tr>
<tr>
<td>53</td>
<td>The subschema definition of an indexed set does not match the physical structure of the indexed set in the database.</td>
</tr>
<tr>
<td>54</td>
<td>The prefix length of an SR51 record is less than zero or the data length is less than or equal to zero.</td>
</tr>
<tr>
<td>55</td>
<td>An invalid length has been defined for a variable-length record.</td>
</tr>
<tr>
<td>56</td>
<td>An insufficient amount of memory to accommodate the CA IDMS compression/decompression routines is available.</td>
</tr>
<tr>
<td>57</td>
<td>A retrieval-only run unit has detected an inconsistency in an index that should cause a 1143 abend, but the optional APAR bit 216 has been turned on.</td>
</tr>
<tr>
<td>58</td>
<td>An attempt was made to roll back updates in a local mode program. Updates that are made to an area during the execution of a local mode program cannot be automatically rolled out. The area must be manually recovered.</td>
</tr>
<tr>
<td>60</td>
<td>A record occurrence type is inconsistent with the set named in the ERROR-SET field in the IDMS communications block. This code usually indicates a broken chain.</td>
</tr>
<tr>
<td>61</td>
<td>No record can be found for an internal db-key. This code usually indicates a broken chain.</td>
</tr>
<tr>
<td>62</td>
<td>A system-generated database key points to a record occurrence, but no record with that db-key can be found. This code usually indicates a broken chain.</td>
</tr>
<tr>
<td>63</td>
<td>The DBMS cannot interpret the DML function to be performed. When this code is combined with a major code of 00, it indicates that invalid function parameters have been passed on the call to the DBMS. For LRF users, a WHERE clause includes a keyword that is longer than the 32 characters allowed.</td>
</tr>
<tr>
<td>64</td>
<td>The record cannot be found; the CALC control element has not been defined properly in the subschema.</td>
</tr>
<tr>
<td>65</td>
<td>The database page read was not the page requested.</td>
</tr>
<tr>
<td>66</td>
<td>The specified area is not available in the requested usage mode.</td>
</tr>
<tr>
<td>67</td>
<td>The invoked subschema does not match the subschema object tables.</td>
</tr>
<tr>
<td>68</td>
<td>The CICS interface was not started.</td>
</tr>
<tr>
<td>69</td>
<td></td>
</tr>
</tbody>
</table>
### Minor Code Database Function Status

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>70</td>
<td>The database will not ready properly; a JCL error is the probable cause.</td>
</tr>
<tr>
<td>71</td>
<td>The page range or page group for the area being readied or the requested page cannot be found in the DMCL.</td>
</tr>
<tr>
<td>72</td>
<td>There is insufficient memory to load a subschema or database procedure dynamically.</td>
</tr>
<tr>
<td>73</td>
<td>A central version run unit will exceed the MAXERUS value specified at system generation.</td>
</tr>
<tr>
<td>74</td>
<td>The dynamic load of a module has failed. If operating under the central version, a subschema or database procedure module either was not found in the data dictionary or the load (core image) library or, if loaded, will exceed the number of subschema and database procedures provided for at system generation.</td>
</tr>
<tr>
<td>75</td>
<td>A read error has occurred.</td>
</tr>
<tr>
<td>76</td>
<td>A write error has occurred.</td>
</tr>
<tr>
<td>77</td>
<td>The run unit has not been bound or has been bound twice. When this code is combined with a major code of 00, it means that the program is no longer signed on to the subschema or the variable subschema tables have been overwritten.</td>
</tr>
<tr>
<td>78</td>
<td>An area wait deadlock has occurred.</td>
</tr>
<tr>
<td>79</td>
<td>The run unit has requested more db-key locks than are available to the system.</td>
</tr>
<tr>
<td>80</td>
<td>The target node is either not active or has been disabled.</td>
</tr>
<tr>
<td>81</td>
<td>The converted subschema requires specified database name to be in the DBNAME table.</td>
</tr>
<tr>
<td>82</td>
<td>The subschema must be named in the DBNAME table.</td>
</tr>
<tr>
<td>83</td>
<td>An error has occurred in accessing native VSAM data sets.</td>
</tr>
<tr>
<td>87</td>
<td>The owner and member records for a set to be updated are not in the same page group or do not have the same database key radix.</td>
</tr>
<tr>
<td>91</td>
<td>The subschema requires a DBNAME to do the bind run unit.</td>
</tr>
<tr>
<td>92</td>
<td>No subschema areas map to DMCL.</td>
</tr>
<tr>
<td>93</td>
<td>A subschema area symbolic was not found in DMCL.</td>
</tr>
<tr>
<td>94</td>
<td>The specified dbname is not defined in the DBNAME table, and is not a SEGMENT defined in the DMCL.</td>
</tr>
<tr>
<td>95</td>
<td>The specified subschema failed DBTABLE mapping using the specified dbname.</td>
</tr>
</tbody>
</table>

**Note:** For a complete description of DB runtime status codes, see [CA IDMS Status Codes](https://docops.ca.com/display/IDMSMXX/CA+IDMS+Status+Codes).
DC Status Codes (COBOL)

The following tables list the DC major and minor codes and their meanings.

### Major DC Status Codes

<table>
<thead>
<tr>
<th>Major Code</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>Any DML statement</td>
</tr>
<tr>
<td>30</td>
<td>TRANSFER CONTROL</td>
</tr>
<tr>
<td>31</td>
<td>WAIT/POST</td>
</tr>
<tr>
<td>32</td>
<td>GET STORAGE/FREE STORAGE</td>
</tr>
<tr>
<td>33</td>
<td>SET ABEND EXIT/ABEND CODE</td>
</tr>
<tr>
<td>34</td>
<td>LOAD/DELETE TABLE</td>
</tr>
<tr>
<td>35</td>
<td>GET TIME/SET TIMER</td>
</tr>
<tr>
<td>36</td>
<td>WRITE LOG</td>
</tr>
<tr>
<td>37</td>
<td>ATTACH/CHANGE PRIORITY</td>
</tr>
<tr>
<td>38</td>
<td>BIND/ACCEPT/END TRANSACTION STATISTICS</td>
</tr>
<tr>
<td>39</td>
<td>ENQUEUE/DEQUEUE</td>
</tr>
<tr>
<td>40</td>
<td>SNAP</td>
</tr>
<tr>
<td>43</td>
<td>PUT/GET/DELETE SCRATCH</td>
</tr>
<tr>
<td>44</td>
<td>PUT/GET/DELETE QUEUE</td>
</tr>
<tr>
<td>45</td>
<td>BASIC MODE TERMINAL MANAGEMENT</td>
</tr>
<tr>
<td>46</td>
<td>MAPPING MODE TERMINAL MANAGEMENT</td>
</tr>
<tr>
<td>47</td>
<td>LINE MODE TERMINAL MANAGEMENT</td>
</tr>
<tr>
<td>48</td>
<td>ACCEPT/WRITE PRINTER</td>
</tr>
<tr>
<td>49</td>
<td>SEND MESSAGE</td>
</tr>
<tr>
<td>50</td>
<td>COMMIT TASK/ROLLBACK TASK/FINISH TASK/WRITE JOURNAL</td>
</tr>
<tr>
<td>51</td>
<td>KEEP LONGTERM</td>
</tr>
<tr>
<td>58</td>
<td>SVC SEND/RECEIVE</td>
</tr>
</tbody>
</table>

### Minor DC Status Codes

<table>
<thead>
<tr>
<th>Minor Code</th>
<th>Function Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>When this code is combined with a major code of 00, it indicates that the DML function completed successfully or that all tested resources have been enqueued.</td>
</tr>
<tr>
<td>01</td>
<td>The requested operation cannot be performed immediately, and waiting would cause a deadlock.</td>
</tr>
<tr>
<td>02</td>
<td></td>
</tr>
<tr>
<td>Minor Code</td>
<td>Function Status</td>
</tr>
<tr>
<td>------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>03</td>
<td>The scratch area ID cannot be found.</td>
</tr>
<tr>
<td>04</td>
<td>The queue ID (header) cannot be found or a paging session was in progress when a second STARTPAGE command was received. That is, an implied ENDPAGE was processed before this STARTPAGE was executed successfully.</td>
</tr>
<tr>
<td>05</td>
<td>The specified scratch record ID or queue record cannot be found.</td>
</tr>
<tr>
<td>06</td>
<td>No resource control element (RCE) exists for the queue record; currency has not been established.</td>
</tr>
<tr>
<td>07</td>
<td>Either an I/O error has occurred or the queue upper limit has been reached.</td>
</tr>
<tr>
<td>08</td>
<td>The requested resource is not available.</td>
</tr>
<tr>
<td>09</td>
<td>The requested resource is available.</td>
</tr>
<tr>
<td>10</td>
<td>New storage has been assigned.</td>
</tr>
<tr>
<td>11</td>
<td>A maximum task condition exists.</td>
</tr>
<tr>
<td>12</td>
<td>The named task code is invalid.</td>
</tr>
<tr>
<td>13</td>
<td>The named resource cannot be found.</td>
</tr>
<tr>
<td>14</td>
<td>The requested module is defined as nonconcurrent and is currently in use.</td>
</tr>
<tr>
<td>15</td>
<td>The named module has been overlaid and cannot be reloaded immediately.</td>
</tr>
<tr>
<td>16</td>
<td>The specified interval control element (ICE) address cannot be found.</td>
</tr>
<tr>
<td>17</td>
<td>The record has been replaced.</td>
</tr>
<tr>
<td>18</td>
<td>No printer terminals have been defined for the current DC system.</td>
</tr>
<tr>
<td>19</td>
<td>The return area is too small; data has been truncated.</td>
</tr>
<tr>
<td>20</td>
<td>An I/O, program-not-found, or potential-deadlock status condition exists.</td>
</tr>
<tr>
<td>21</td>
<td>The message destination is undefined, the long-term ID cannot be found, or a nonterminal task issued a KEEP LONGTERM request.</td>
</tr>
<tr>
<td>22</td>
<td>A record already exists for the scratch area specified.</td>
</tr>
<tr>
<td>23</td>
<td>No storage or resource control element (RCE) could be allocated for the reply area.</td>
</tr>
<tr>
<td>24</td>
<td>The maximum number of outstanding replies has been exceeded.</td>
</tr>
<tr>
<td>25</td>
<td>An attention interrupt has been received.</td>
</tr>
<tr>
<td>26</td>
<td>The output data stream contains a logical error.</td>
</tr>
<tr>
<td>27</td>
<td>A permanent I/O error has occurred.</td>
</tr>
<tr>
<td>28</td>
<td>The terminal dial-up line is disconnected.</td>
</tr>
<tr>
<td>29</td>
<td>An invalid parameter has been passed in the list set up by the DML processor.</td>
</tr>
<tr>
<td>30</td>
<td>The named function has not yet been implemented.</td>
</tr>
<tr>
<td>31</td>
<td>An invalid parameter has been passed, the TRB, LRB, or MRB contains an invalid field, or the request is invalid because of a possible logic error in the application program. In a DC-BATCH environment, a possible cause is that the command specifies a record length that exceeds the maximum length based on the packet size.</td>
</tr>
<tr>
<td>Minor Code</td>
<td>Function Status</td>
</tr>
<tr>
<td>------------</td>
<td>----------------</td>
</tr>
<tr>
<td>32</td>
<td>The derived length of the specified variable storage is negative or zero.</td>
</tr>
<tr>
<td>33</td>
<td>Either the named table or the named map cannot be found in the data dictionary load area.</td>
</tr>
<tr>
<td>34</td>
<td>The named variable-storage area must be an 01-level entry in the LINKAGE SECTION.</td>
</tr>
<tr>
<td>35</td>
<td>A GET STORAGE request is invalid because the LINKAGE SECTION variable has already been allocated.</td>
</tr>
<tr>
<td>36</td>
<td>The program either was not defined during system generation or is marked out-of-service.</td>
</tr>
<tr>
<td>37</td>
<td>A GET STORAGE operand is invalid. The specified variable storage area is in the WORKING-STORAGE SECTION instead of the LINKAGE SECTION.</td>
</tr>
<tr>
<td>38</td>
<td>No GET STORAGE operand was specified or the specified LINKAGE SECTION variable has not been allocated.</td>
</tr>
<tr>
<td>39</td>
<td>The terminal device being used is out of service.</td>
</tr>
<tr>
<td>40</td>
<td>NOIO has been specified but the data stream cannot be found.</td>
</tr>
<tr>
<td>41</td>
<td>An IF operation resulted in a valid true condition.</td>
</tr>
<tr>
<td>42</td>
<td>The named map does not support the terminal device in use.</td>
</tr>
<tr>
<td>43</td>
<td>The terminal operator canceled a line I/O session.</td>
</tr>
<tr>
<td>44</td>
<td>The referenced field does not participate in the specified map. A possible cause is an invalid subscript.</td>
</tr>
<tr>
<td>45</td>
<td>An invalid terminal type is associated with the issuing task.</td>
</tr>
<tr>
<td>46</td>
<td>A terminal I/O error has occurred.</td>
</tr>
<tr>
<td>47</td>
<td>The named area has not been readied.</td>
</tr>
<tr>
<td>48</td>
<td>The run unit has not been bound.</td>
</tr>
<tr>
<td>49</td>
<td>NOWAIT has been specified but WAIT is required.</td>
</tr>
<tr>
<td>50</td>
<td>Statistics are not being kept.</td>
</tr>
<tr>
<td>51</td>
<td>A lock manager error occurred during the processing of a KEEP LONGTERM request.</td>
</tr>
<tr>
<td>52</td>
<td>The specified table is missing or invalid.</td>
</tr>
<tr>
<td>53</td>
<td>An error occurred from a user-written edit routine.</td>
</tr>
<tr>
<td>54</td>
<td>Either there is invalid internal data or a data conversion error has occurred.</td>
</tr>
<tr>
<td>55</td>
<td>The user-written edit routine cannot be found.</td>
</tr>
<tr>
<td>56</td>
<td>No DFLDS have been defined for the map.</td>
</tr>
<tr>
<td>57</td>
<td>The ID cannot be found, is not a long-term permanent ID, or is being used by another run unit.</td>
</tr>
<tr>
<td>58</td>
<td>The LRID cannot be found, the maximum number of concurrent task threads was exceeded, or an attempt was made to roll back database changes in local mode.</td>
</tr>
<tr>
<td>59</td>
<td>An error occurred in transferring the KEEP LONGTERM request to IDMSKEEP.</td>
</tr>
<tr>
<td>60</td>
<td>The requested KEEP LONGTERM lock id was already in use with a different page group.</td>
</tr>
<tr>
<td>63</td>
<td>Invalid function parameters have been passed on the call to the DBMS.</td>
</tr>
<tr>
<td>64</td>
<td>No detail exists currently for update; no action has been taken. Alternatively, the requested node for a header or detail is either not present or not updated.</td>
</tr>
</tbody>
</table>
There are no more updated details to MAP IN or the amount of storage that is defined for pageable maps at sysex is insufficient. In the latter case, subsequent MAP OUT DETAIL statements are ignored.

No detail occurrence, footer, or header fields exist for the MAP OUT RESUME command to map out, or the scratch record that contains the requested detail could not be accessed. The latter case is a mapping internal error and should be reported.

The first screen page has been transmitted to the terminal.

Either the program is no longer signed on to the subschema or the variable subschema tables have been overwritten.

The target node is either not active or has been disabled.

An error was encountered processing a syncpoint request. Review the log for details.

An unsupported COBOL compiler option (for example, DEBUG) has been specified for an online program, or a program that is running in a batch region has issued a DML verb that is only valid when running online under CA IDMS/DC/UCF.

An unexpected internal return code has been received; the terminal device is out of service.

Note: For a complete description of DC runtime status codes, see CA IDMS Status Codes (https://docops.ca.com/display/IDMSMXX/CA+IDMS+Status+Codes).

### Condition Names

The following table lists the ERROR-STATUS condition names.

<table>
<thead>
<tr>
<th>Code</th>
<th>Condition name</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>DB-STATUS-OK</td>
<td>No error.</td>
</tr>
<tr>
<td>0307</td>
<td>DB-END-OF-SET</td>
<td>End of set, area, or SPF index.</td>
</tr>
<tr>
<td>0326</td>
<td>DB-REC-NOT-FOUND</td>
<td>No record found.</td>
</tr>
<tr>
<td>0001 to</td>
<td>ANY-ERROR-STATUS</td>
<td>Any nonzero status.</td>
</tr>
<tr>
<td>9999</td>
<td>ANY-STATUS</td>
<td>Any status.</td>
</tr>
<tr>
<td>3101</td>
<td>DC-DEADLOCK</td>
<td>Waiting will cause a deadlock.</td>
</tr>
<tr>
<td>3401</td>
<td>DC-NO-STORAGE</td>
<td>Insufficient space available.</td>
</tr>
<tr>
<td>4303</td>
<td>DC-AREA-ID-UNK</td>
<td>ID cannot be found.</td>
</tr>
</tbody>
</table>
Error Detection (COBOL)

Check the value that is returned to the ERROR-STATUS field after each DML request. When you use the Logical Record Facility, check the LR-STATUS field of the LRC block before checking the ERROR-STATUS field.

CA IDMS provides three aids for error detection:
- AUTOSTATUS Protocols (see page 56)
- IDMS-STATUS Routine (see page 57)
- User-Defined Protocols (see page 61)

AUTOSTATUS Protocols

When the protocol in use includes AUTOSTATUS, the precompiler automatically generates a PERFORM IDMS-STATUS statement after each DML command (except IF). For each standard protocol (for example, BATCH or CICS) that is provided at installation time, an AUTOSTATUS protocol (for example, BATCH-AUTOSTATUS or CICS-AUTOSTATUS) is also provided. (The IDMS DC and DC-BATCH protocols already include AUTOSTATUS.) The DBA determines which protocol to use. Specify this protocol in the ENVIRONMENT DIVISION by using the MODE IS statement. (For more information about protocols, see Precompiler-Directive Statements (see page 63).)

When AUTOSTATUS is in use, the PERFORM IDMS-STATUS statement can still be preceded by a check for a nonzero return code by including an ON clause at the end of the DML command. If the DBMS returns the specified status code to the IDMS communications block, the imperative statement in the ON clause is executed. If the specified status code is not returned, IDMS-STATUS is performed.

Any DML command can include an ON clause. Only one ON clause is allowed per command.

Syntax

```
>>> ON condition-name imperative-statement .
```
Parameters

- **ON**
  Tests whether a nonzero status has been returned as a result of a DML command.

- **condition-name**
  Specifies a preassigned nonzero status condition name. Valid condition names include DB-STATUS-OK, DB-END-OF-SET, DB-REC-NOT-FOUND, ANY-ERROR-STATUS, and any condition names that the DBA has defined.

- **imperative-statement**
  Specifies the program action to take when the DML command results in the specified condition-name.

The following example illustrates the use of the ON clause. A DML source program might contain the following statements:

```
0800-OBTAIN-REC.
  OBTAIN CALC OFFICE ON DB-REC-NOT-FOUND GO TO 0900-NO-REC.
  ;
  ;
0900-NO-REC.
  STORE OFFICE.
```

The precompiler converts the DML statements to comments, translates the ON clause into an IF statement, and generates the following expanded COBOL source code:

```
0800-OBTAIN-REC.
  * OBTAIN CALC OFFICE ON DB-REC-NOT-FOUND
  MOVE 0001 TO DML-SEQUENCE
  CALL 'IDMS' USING SUBSCHEMA-CTRL
  IDBMSCOM (32)
  SR450
  IDBMSCOM (43)
  IF NOT DB-REC-NOT-FOUND PERFORM IDMS-STATUS;
  ELSE
    GO TO 0900-NO-REC.
  ;
  ;
0900-NO-REC.
  * STORE OFFICE.
  MOVE 0002 TO DML-SEQUENCE
  CALL 'IDMS' USING SUBSCHEMA-CTRL
  IDBMSCOM (42)
  SR450
  PERFORM IDMS-STATUS.
```

For more information about the expansion of calls to CA IDMS, see CA IDMS Call Formats (see page 358).

**IDMS-STATUS Routine**

IDMS-STATUS is an error-checking routine that is included in the dictionary. You can copy IDMS-STATUS into your program by coding the following statement at the end of the PROCEDURE DIVISION:
COPY IDMS IDMS-STATUS.

For more information about the use of the COPY IDMS IDMS-STATUS statement, refer to Precompiler-Directive Statements (see page 63).

This section contains the following information:

- IDMS-STATUS Routine Used Under Batch (see page 58)
- IDMS-STATUS Routine Used Under a DC/UCF System (see page 58)
- Pageable Map ERROR-STATUS Condition Names (see page 59)
- Effects of Nonzero Status on IDMS-STATUS (see page 59)
- The IDMS-ABORT Routine (see page 61)

**IDMS-STATUS Routine Used Under Batch**

The COPY IDMS IDMS-STATUS statement copies the following code into batch programs:

```plaintext
******************************************************************
IDMS-STATUS                  SECTION.
******************************************************************
IDMS-STATUS-PARAGRAPH.
IF DB-STATUS-OK GO TO ISABEX.
PERFORM IDMS-ABORT.
DISPLAY '**************************
ABORTING - PROGRAM-NAME
', ' ERROR-STATUS
', ' ERROR-RECORD
**** RECOVER IDMS ****'
UPON CONSOLE.
DISPLAY 'PROGRAM NAME -----' PROGRAM-NAME.
DISPLAY 'ERROR STATUS ------' ERROR-STATUS.
DISPLAY 'ERROR RECORD ------' ERROR-RECORD.
DISPLAY 'ERROR SET --------' ERROR-SET.
DISPLAY 'ERROR AREA -------' ERROR-AREA.
DISPLAY 'LAST GOOD RECORD --' RECORD-NAME.
DISPLAY 'LAST GOOD AREA ----' AREA-NAME.
MOVE 39 TO SSC-IN01-REQ-CODE.
MOVE 0 TO SSC-IN01-REQ-RETURN.
MOVE ' ' TO SSC-STATUS-LABEL.
PERFORM IDMS-STATUS-LOOP UNTIL SSC-IN01-REQ-RETURN > 0.
ROLLBACK.
CALL 'ABORT'.
GO TO ISABEX.
IDMS-STATUS-LOOP.
CALL 'IDMSINI' USING IDBMSCOM(41)
SSC-IN01-REQ-WK
SUBSCHEMA-CTRL
IDBMSCOM(1)
DML-SEQUENCE
SSC-STATUS-LINE.
IF SSC-IN01-REQ-RETURN GREATER THAN 4
DISPLAY 'DML SEQUENCE --------' DML-SEQUENCE
ELSE
DISPLAY SSC-STATUS-LABEL '---' SSC-STATUS-VALUE.
ISABEX. EXIT.
```

**IDMS-STATUS Routine Used Under a DC/UCF System**

The COPY IDMS IDMS-STATUS statement copies the following code into DC/UCF programs:

```plaintext
******************************************************************
IDMS-STATUS                  SECTION.
******************************************************************
IDMS-STATUS-PARAGRAPH.
IF DB-STATUS-OK GO TO ISABEX.
PERFORM IDMS-ABORT.
DISPLAY '**************************
ABORTING - PROGRAM-NAME
', ' ERROR-STATUS
', ' ERROR-RECORD
**** RECOVER IDMS ****'
UPON CONSOLE.
DISPLAY 'PROGRAM NAME -----' PROGRAM-NAME.
DISPLAY 'ERROR STATUS ------' ERROR-STATUS.
DISPLAY 'ERROR RECORD ------' ERROR-RECORD.
DISPLAY 'ERROR SET --------' ERROR-SET.
DISPLAY 'ERROR AREA -------' ERROR-AREA.
DISPLAY 'LAST GOOD RECORD --' RECORD-NAME.
DISPLAY 'LAST GOOD AREA ----' AREA-NAME.
MOVE 39 TO SSC-IN01-REQ-CODE.
MOVE 0 TO SSC-IN01-REQ-RETURN.
MOVE ' ' TO SSC-STATUS-LABEL.
PERFORM IDMS-STATUS-LOOP UNTIL SSC-IN01-REQ-RETURN > 0.
ROLLBACK.
CALL 'ABORT'.
GO TO ISABEX.
IDMS-STATUS-LOOP.
CALL 'IDMSINI' USING IDBMSCOM(41)
SSC-IN01-REQ-WK
SUBSCHEMA-CTRL
IDBMSCOM(1)
DML-SEQUENCE
SSC-STATUS-LINE.
IF SSC-IN01-REQ-RETURN GREATER THAN 4
DISPLAY 'DML SEQUENCE --------' DML-SEQUENCE
ELSE
DISPLAY SSC-STATUS-LABEL '---' SSC-STATUS-VALUE.
ISABEX. EXIT.
```
IF DB-STATUS-OK GO TO ISABEX.
PERFORM IDMS-ABORT.
MOVE ERROR-STATUS TO SSC-ERRSTAT-SAVE
MOVE DML-SEQUENCE TO SSC-DMLSEQ-SAVE
SNAP FROM SUBSCHEMA-CTRL TO SUBSCHEMA-CTRL-END
ON ANY-STATUS NEXT SENTENCE.
ABEND CODE SSC-ERRSTAT-SAVE ON ANY-STATUS NEXT SENTENCE.
ISABEX. EXIT.

IDMS-STATUS abends your program when the ERROR-STATUS field contains a nonzero value. Because some values do not indicate processing errors, your program should check ERROR-STATUS for nonzero values before calling IDMS-STATUS.

Pageable Map ERROR-STATUS Condition Names

The following table lists the condition names that are included automatically when using pageable maps.

⚠️ **Note:** You cannot make checks for these codes within the IDMS-STATUS routine.

<table>
<thead>
<tr>
<th>Code</th>
<th>Condition name</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>4604</td>
<td>DC-SECOND-STARTPAGE</td>
<td>Second consecutive STARTPAGE.</td>
</tr>
<tr>
<td>4664</td>
<td>DC-DETAIL-NOT-FOUND</td>
<td>No current detail.</td>
</tr>
<tr>
<td>4668</td>
<td>DC-NO-MORE-UPD-DETAILS</td>
<td>All details mapped in.</td>
</tr>
<tr>
<td>4668</td>
<td>DC-MAX-SPACE-REACHED</td>
<td>Pageable map space exceeded.</td>
</tr>
<tr>
<td>4672</td>
<td>DC-NO-DETAILS</td>
<td>Nothing to map out.</td>
</tr>
<tr>
<td>4676</td>
<td>DC-FIRST-PAGE-SENT</td>
<td>First page transmitted.</td>
</tr>
<tr>
<td>4680</td>
<td>DC-PAGE-READY</td>
<td>A complete map page was built.</td>
</tr>
</tbody>
</table>

When IDMS-STATUS executes, it exits immediately when the ERROR-STATUS check indicates successful completion of the function (ERROR-STATUS of 0000).

Effects of Nonzero Status on IDMS-STATUS

This section describes the effects of nonzero status conditions on IDMS-STATUS execution. The effects depend on the operating mode (BATCH or IDMS-DC) of the application program.

Effect When the Operating Mode Is BATCH

When the operating mode is BATCH, a nonzero error status causes IDMS-STATUS to perform the following actions:

- Print status information about the unsuccessful function.
- Issue a rollback.
Abend the program.

The following status information is retrieved from the IDMS-DB communications block:

- Program name
- Error status
- Error record
- Error set
- Error area
- Record name (the last record that was successfully accessed)
- Area name (the last area that was successfully accessed)
- Page number and line index of the database key (the last record that the run unit accessed)
- Database key in hexadecimal format
- Page group and database key format (associated with the last record that the run unit accessed)
- DML sequence number

**Effect When the Operating Mode Is IDMS-DC**

When the operating mode is IDMS-DC, a nonzero error status causes IDMS-STATUS to perform the following actions:

- Snap the IDMS-DC communications block (SUBSCHEMA-CTRL).
- Abend the program.

The following status information is retrieved from the IDMS-DC communications block:

- Program name
- Error status
- Error record
- Error set
- Error area
- Record name (the last record that was successfully accessed)
- Area name (the last area that was successfully accessed)
- DML sequence number
The IDMS-ABORT Routine

IDMS-STATUS includes a call to perform a routine named IDMS-ABORT, which you can use for extra error processing. CA IDMS supplies only the PERFORM statement. If you use the IDMS-ABORT routine, you must supply the routine itself by coding the section name and exit. See the following example:

IDMS-ABORT SECTION.
IDMS-ABORT-EXIT.
EXIT.

For example, you can use the IDMS-ABORT SECTION to display information about the LRC block. See the following example:

IDMS-ABORT SECTION.
IF LR-STATUS = ‘LR-ERROR’
  DISPLAY ‘LOGICAL RECORD ERROR’
  ‘LR NAME -- ’ LR-NAME
  ‘LR VERB -- ’ LR-VERB.
IDMS-ABORT-EXIT.
EXIT.

Code the routine directly into the program or copy it in as a module, according to the program requirements. If you are not using the abort routine, delete the reference to IDMS-ABORT from IDMS-STATUS.

User-Defined Protocols

To establish a user-defined protocol, follow these steps:

1. Establish a uniquely named user-defined MODE.

2. Identify an existing CA-supplied protocol that meets the program requirements. Use this protocol, with modifications as needed, to create a new protocol with the same name as the user-defined MODE.

3. Modify the appropriate SUBSCHEMA-CTRL record definition to include the user-defined MODE.

4. Specify the user-defined MODE in the PROTOCOL parameter of the program.

For example, to create a version of the DC-BATCH protocol that does not include AUTOSTATUS, follow these steps:

1. Define the user-defined MODE:
   
   ADD ATTRIBUTE DC-BATCH-NOAUTO WITHIN CLASS MODE.

2. Define the user-defined protocol that is based on the CA-supplied DC-BATCH protocol and edit the DC-BATCH protocol to remove the @AUTOSTATUS references:
   
   ADD MODULE NAME DC-BATCH-NOAUTO VERSION 1 LANGUAGE IS COBOL
   MODE IS DC-BATCH-NOAUTO
   MODULE SOURCE FOLLOWS
3. Modify the SUBSCHEMA-CTRL record for MODE IS DC-BATCH to include the user-defined MODE:

   MODIFY RECORD SUBSCHEMA-CTRL VERSION 1 LANGUAGE COBOL
   MODE IS DC-BATCH-NOAUTO.

4. Specify the user-defined MODE in the program:

   PROTOCOL MODE IS DC-BATCH-NOAUTO
Precompiler-Directive Statements

Compiler-directive statements instruct the precompiler to copy source code from the data dictionary into the COBOL application program. These statements do not produce any executable commands. Compiler-directive statements are coded beginning in columns 8-11 of the IDENTIFICATION and ENVIRONMENT DIVISIONs, and in columns 8-72 of the DATA and PROCEDURE DIVISIONs, as follows:

- **IDENTIFICATION DIVISION** -- The PROGRAM-ID statement specifies a program name and version number.

- **ENVIRONMENT DIVISION** -- The IDMS-CONTROL SECTIOn establishes the operating mode, debug sequencing, and variable storage allocation.

- **DATA DIVISION** -- The following sections are included in the DATA DIVISION:
  
  - **FILE SECTION** -- COPY IDMS FILE statements copy descriptions of non-IDMS files from the data dictionary.
  
  - **SCHEMA SECTION** -- The DB statement identifies the subschema view to be used by the program.
  
  - **MAP SECTION** -- These statements notify the precompiler that mapping mode terminal I/O is being used, define the program's maps, and specify the size of map field lists.

  - **WORKING-STORAGE and LINKAGE SECTIONS** -- **PROCEDURE DIVISION** -- COPY IDMS statements copy source data descriptions or non-IDMS data description code for records from the data dictionary.

- **COPY IDMS statements copy source data for BIND statements or program source modules defined in the data dictionary.**

All compiler-directive statements are optional except the SCHEMA SECTION and DB statement. If a program accesses the database, it must include a SCHEMA SECTION that contains a DB statement identifying the subschema. All other compiler-directive statements can be omitted; the precompiler will generate the required source code components automatically.

If the program does not access the database (that is, does not invoke a subschema and does not issue any DML statements), the SCHEMA SECTION and DB statement can be omitted as well.

The COPY IDMS and other compiler-directive statements are explained separately for each of the following divisions. References to the IDMS communications block apply equally to the IDMS-DC communications block.

This article describes the following information:

- **IDENTIFICATION DIVISION** (see page 64)
- **ENVIRONMENT DIVISION** (see page 64)
- **DATA DIVISION** (see page 67)
- **PROCEDURE DIVISION** (see page 78)
IDENTIFICATION DIVISION

The PROGRAM-ID statement in the IDENTIFICATION DIVISION identifies your program to the precompiler.

Syntax

►►─── IDENTIFICATION DIVISION. ──►
►─── PROGRAM-ID.program-name ─┬──────────────────────────┬───────────────────►◄
└─ VERSION version-number ─┘

Parameters

- **PROGRAM ID**
  Specifies the program.

- **program-name**
  The name of the program. If the program has been previously defined in the data dictionary through IDD facilities, **program-name** must match the name assigned to the program when it was defined in order for the precompiler to recognize it as the same program.

- **VERSION**
  Qualifies **program-name** with a version number (for example, for purposes of testing or development).

- **version-number**
  An integer in the range 1 through 9999. By default, if you do not specify a number, the default is either the highest number defined in the data dictionary for the named program or 1 if the program does not already exist in the data dictionary.

ENVIRONMENT DIVISION

An IDMS-CONTROL SECTION is required in the ENVIRONMENT DIVISION to establish the following:

- **Operating mode** -- The environment in which the program will execute, and the form and content of calling sequences produced by the precompiler

- **Debug sequencing** -- Whether each PROCEDURE DIVISION DML command will be numbered for identification during error reporting

- **Variable storage allocation** -- How source data description code for the IDMS communications block and other DATA DIVISION components will be inserted in the program

Syntax

►►─── ENVIRONMENT DIVISION. ──►
►─────────────────────────────────────────────────────────►
└─ IDMS-CONTROL SECTION. ──►
Parameters

- **PROTOCOL**
  Specifies how CA IDMS CALL statements are generated and whether the debugging sequence option is included.

- **MODE IS**
  Identifies the operating mode used by the precompiler to generate call statements for the program's PROCEDURE DIVISION DML statements.

- **BATCH**
  Specifies to execute the program in batch mode. This is the default. The IDMS communications block is copied into variable storage; standard CALL statements (CALL 'IDMS') are generated in the PROCEDURE DIVISION.

- **IDMS DC**
  Specifies to execute the program in IDMS-DC mode. The IDMS DC communications block is copied into variable storage; CA IDMS CALL statements (CALL 'IDMSCOBI') are generated in the PROCEDURE DIVISION for DC requests.

- **DC-BATCH**
  Specifies to execute the program in DC-BATCH mode. The IDMS-DC communications block is copied into variable storage; DC-BATCH CALL statements (CALL 'IDMSDCCI') are generated in the PROCEDURE DIVISION for DC requests. Specify MODE IS DC-BATCH to access DC queues and printers from batch applications running under the central version.

- **user-mode-name**
  Specifies to execute the program in a special environment (for example, under a teleprocessing monitor or in a user-defined operating mode) as determined by the DBA. The appropriate communications block is copied into variable storage; mode-specific CALL statements (for example, in CICS: CALL 'IDMSINC1' USING DFHCADS) are generated in the PROCEDURE DIVISION. The following list provides the standard operating modes (protocols) available for COBOL programs.
  If **user-mode-name** specifies an AUTOSTATUS protocol (for example, CICS-AUTOSTATUS), the precompiler automatically generates an IDMS-STATUS statement after every DML command.
except IF. When using an AUTOSTATUS protocol, be sure to include the COPY IDMS IDMS-STATUS statement in the PROCEDURE DIVISION. For details on programming under an AUTOSTATUS protocol, see Communications Blocks and Error Detection (https://docops.ca.com/display/IDMSCU19/Communications+Blocks+and+Error+Detection).

- **DEBUG**
  Specifies that a unique DML sequence number is placed in the IDMS communications block for each DML statement. These numbers appear in columns 81-88 of the COBOL compiler output listing in the form DMLCnnn. The precompiler generates numbers to identify the sequence in which DML statements appear in the program. Depending on the error routine defined by the DBA, you can use the DML sequence number to help debug your program.
  If DEBUG is not specified, the precompiler does not associate sequence numbers with source statements.

- **SUBSCHEMA-NAMES LENGTH IS**
  Specifies whether to use a 16-byte or 18-byte communications block.
  For information about 16-byte communications blocks, see Communications Blocks and Error Detection (https://docops.ca.com/display/IDMSCU19/Communications+Blocks+and+Error+Detection).
  For information about 18-byte communications blocks, see 18-Byte Communications Blocks (Assembler) (https://docops.ca.com/pages/viewpage.action?pageId=357363226).

- **IDMS-RECORDS**
  Specifies whether source CA IDMS data description code is inserted into the DATA DIVISION automatically.

- **WITHIN WORKING-STORAGE section**
  Instructs the processor to insert automatically the copied DATA DIVISION components as the last entries in the WORKING-STORAGE SECTION of the source program.
  This is the default.

- **WITHIN LINKAGE section**
  Instructs the processor to automatically insert the copied DATA DIVISION components as the last entries in the LINKAGE SECTION of the source program. Any VALUE clauses present in source code will be dropped automatically.

- **levels INCREMENTED by**
  Varies the level numbers for inserted descriptions from those stored in the data dictionary. If you specify a level number, the first level of code will be inserted to the level specified by count; all other levels will be adjusted accordingly. If you do not specify a level, the descriptions inserted will begin at 01 and have the same level numbers as originally specified in the data dictionary.

- **count**
  An integer in the range 1 through 48.
  Specifies the value by which the DATA DIVISION level numbers (including the 01 level number) of all stored elements are to be incremented.

⚠️ **Note:** Using the LEVELS INCREMENTED BY clause may cause unpredictable results if record fields have been defined with a SYNCHRONIZED clause. Such fields may contain extra bytes (slack bytes) inserted to ensure correct alignment. Because CA IDMS does not recognize slack bytes as functional, it may misinterpret data fields that contain them. Therefore, you should ensure that all fields and records are properly structured.
• MANUAL
Indicates that CA IDMS-related source data description code (for example, SUBSCHEMA-CTRL or SUBSCHEMA-NAMES) will be inserted explicitly into the source program by means of DATA DIVISION COPY IDMS statements. If MANUAL is not specified, the required DATA DIVISION code is inserted automatically by the precompiler.

Standard Modes Available for COBOL Programs

<table>
<thead>
<tr>
<th>Mode</th>
<th>DC-BATCH</th>
<th>TASKMASTER</th>
</tr>
</thead>
<tbody>
<tr>
<td>BATCH</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BATCH-AUTOSTATUS</td>
<td>IDMS-DC</td>
<td>TASKMASTER-AUTO</td>
</tr>
<tr>
<td>CICS</td>
<td>INTERCOMM</td>
<td>UTM</td>
</tr>
<tr>
<td>CICS-AUTOSTATUS</td>
<td>INTERCOMM-AUTO</td>
<td>UTM-AUTOSTATUS</td>
</tr>
<tr>
<td>CICS-EXEC</td>
<td>INTERCOMM-REENT</td>
<td>WESTI</td>
</tr>
<tr>
<td>CICS-EXEC-AUTO</td>
<td>ICOMM-REENT-AUTO</td>
<td>WESTI-AUTOSTATUS</td>
</tr>
<tr>
<td>CICS-STANDARD</td>
<td>SHADOW</td>
<td>WESTI-REENT</td>
</tr>
<tr>
<td>CICS-STD-AUTO</td>
<td>SHAD-AUTOSTATUS</td>
<td>WESTI-REENT-AUTO</td>
</tr>
</tbody>
</table>

The following example illustrates the statements used to code the IDMS-CONTROL SECTION of a program running under DC with DEBUG sequencing and automatic insertion of IDMS-RECORDS in WORKING-STORAGE SECTION:

ENVIRONMENT DIVISION.
IDMS-CONTROL SECTION.
PROTOCOL.
MODE IS IDMS-DC
DEBUG
IDMS-RECORDS WITHIN WORKING-STORAGE SECTION.

DATA DIVISION

Compiler-directive statements can be in the following sections of the DATA DIVISION:

• FILE SECTION -- COPY IDMS statements copy descriptions of non-IDMS files from the data dictionary

• SCHEMA SECTION -- A DB statement identifies the subschema view to be used by the program

• MAP SECTION -- These statements notify the precompiler that mapping mode terminal I/O is being used, define the program's maps, and specify the size of map field lists

• WORKING-STORAGE SECTION -- COPY IDMS statements copy source data description or non-IDMS data description code for records from the data dictionary

• LINKAGE SECTION -- COPY IDMS statements copy source data description or non-IDMS data description code for records from the data dictionary
FILE SECTION

The FILE SECTION can include one or more COPY IDMS statements to copy non-IDMS file descriptions from the data dictionary into the program. Each COPY IDMS statement generates the file definition that includes record size, block size, and recording mode from the data dictionary. Additionally, any records defined within the file through the IDD facilities are also copied.

Syntax

```
FILE SECTION.

COPY IDMS FILE file-name VERSION version-number .
```

Parameters

- **COPY IDMS FILE**
  Copies the description of a non-IDMS file into the DATA DIVISION.

- **file-name**
  Either the primary name or a synonym for a file defined in the data dictionary.

- **VERSION**
  Qualifies `file-name` with a version number.
  If you do not specify a version number, the default is the highest version number defined in the data dictionary for `file-name`.

- **version-number**
  An integer in the range 1 through 9999.

SCHEMA SECTION

For any program that accesses the database, a SCHEMA SECTION is included in the DATA DIVISION to identify a subschema view to the precompiler. The subschema named in the DB statement of the SCHEMA SECTION determines which record descriptions can be copied into the program from the data dictionary. Every DML command issued by the program is checked against the record, set, and area access restrictions specified in this subschema.

Syntax

```
SCHEMA SECTION.

DB subschema-name WITHIN schema-name VERSION version-number .
```

Parameters

- **DB subschema-name**
  The name of the subschema.

- **schema-name**
  The name of the schema containing the subschema.

- **version-number**
  An integer in the range 1 through 9999.
- **DB subschema-name**
  Specifies a subschema defined in the data dictionary. If the DBA has chosen to preregister valid program names for this subschema in the data dictionary, the program named in the IDENTIFICATION DIVISION must be associated with `subschema-name` in the data dictionary.

- **WITHIN schema-name**
  Specifies the schema under which `subschema-name` is compiled.

- **VERSION**
  Qualifies `schema-name` with a version number.
  If you do not specify a version number, the default is the highest version number defined in the data dictionary for `file-name`.

- **version-number**
  An integer in the range 1 through 9999.

### MAP SECTION

The MAP SECTION notifies the precompiler that mapping mode terminal I/O is being used, defines the program's maps, and specifies the size of map field lists.

**Syntax**

```
MAP SECTION.
  MAX FIELD LIST is field-list-size
  MAP map-name VERSION version-number
  TYPE is STANDARD EXTENDED PAGING
```

**Parameters**

- **MAX FIELD LIST is**
  Specifies the size of field lists used in MODIFY MAP and INQUIRE MAP statements.

- **field-list-size**
  The field list size or the size is expressed as a numeric constant.
  The specified size must be at least one greater than the size of the largest field list used by the program. For example, if the largest map field list contains 5 fields, the value of `field-list-size` must be at least 6.
  The MAX FIELD LIST statement must be specified if the program uses a field list in a MODIFY MAP or INQUIRE MAP request.

- **MAP**
  Defines the map used by the program. This parameter can be repeated as necessary to define each map to be used.
map-name
The name of a map used by the program.

VERSION
Qualifies the named map with a version number.

version-number
An integer in the range 1 through 9999.
There is no default for version-number. If your site uses multiple versions, you must specify a version number.

TYPE Is
Specifies whether the map request block (MRB) built for the map is to be standard or extended.

STANDARD
Specifies that the map has standard 3270-type terminal attributes.
This is the default.

EXTENDED
Specifies that the map has extended 3279-type terminal attributes (for example, color, blinking fields, reverse video).

PAGING
Specifies that the named map is a pageable map.

The following example shows the DATA DIVISION statements required to access the EMPSS09 subschema and the EMPMAPLR map; the largest map field list allowed is 4.

DATA DIVISION.
SCHEMA SECTION.
DB EMPSS09 WITHIN EMPSCHM.
MAP SECTION.
MAX FIELD LIST IS 5.
MAP EMPMAPLR VERSION 1 TYPE IS STANDARD.

WORKING-STOREAGE and LINKAGE SECTIONS

COPY IDMS statements can be coded in the WORKING-STOREAGE and LINKAGE SECTIONS, allowing you to explicitly copy source code from the data dictionary into the program. No COPY IDMS statements are required in the DATA DIVISION unless the IDMS-RECORDS MANUAL clause has been specified in the IDMS-CONTROL SECTION of the ENVIRONMENT DIVISION.

If the source code to be copied into the LINKAGE SECTION includes VALUE clauses, these clauses are not copied.

WORKING-STOREAGE SECTION and LINKAGE SECTION source code requirements differ according to the usage mode defined in the program's subschema: DML, LR, or MIXED. These usage modes determine whether the program can access database records only, logical records only, or both database records and logical records. The program should not copy components that conflict with its subschema's usage mode (for example, do not copy SUBSCHEMA-LR-CTRL if the subschema's usage mode is DML).
An explanation of each usage mode and the required source code components in the program is shown below:

- **DML** allows a program to access database records only and requires the following source code components:
  - **SUBSCHEMA-CTRL** -- The IDMS communications block, through which the application program and the DBMS communicate. For more information, see Chapter 4 ([https://docops.ca.com/display/IDMSCU19/Communications+Blocks+and+Error+Detection](https://docops.ca.com/display/IDMSCU19/Communications+Blocks+and+Error+Detection)).
  - **SUBSCHEMA-NAMES** -- The name of the program's subschema and the names of all records, sets, and areas to which the program has access through this subschema. SUBSCHEMA-NAMES is used by the precompiler to generate appropriate CA IDMS CALL statements in the PROCEDURE DIVISION.
  - **SUBSCHEMA-RECORDS** -- The description of all records to which the subschema permits access.

- **LR** allows a program to access logical records only and requires the following source code components:
  - **SUBSCHEMA-CTRL** -- The IDMS communications block, through which LRF and the DBMS communicate. For more information, see Chapter 4 ([https://docops.ca.com/display/IDMSCU19/Communications+Blocks+and+Error+Detection](https://docops.ca.com/display/IDMSCU19/Communications+Blocks+and+Error+Detection)).
  - **SUBSCHEMA-LR-CTRL** -- The logical-record request control (LRC) block, through which the application program and LRF communicate. For more information, see Chapter 4 ([https://docops.ca.com/display/IDMSCU19/Communications+Blocks+and+Error+Detection](https://docops.ca.com/display/IDMSCU19/Communications+Blocks+and+Error+Detection)).
  - **SUBSCHEMA-LR-NAMES** -- The name of the program's subschema and the names of all database areas that can be accessed through the subschema. Logical-record names are not copied into the program; rather, they are moved as literals into the LRC block when needed to process a logical-record request.
  - **SUBSCHEMA-LR-RECORDS** -- The descriptions of all logical records contained in the subschema.

- **MIXED** allows a program to access both database records and logical records; this usage mode requires the following source code components:
  - **SUBSCHEMA-CTRL**
  - **SUBSCHEMA-NAMES**
  - **SUBSCHEMA-RECORDS**
  - **SUBSCHEMA-LR-CTRL**
  - **SUBSCHEMA-LR-RECORDS**

The use of MIXED mode is not recommended for the following reasons:
Issuing both logical-record and database requests requires that the program take into account the database currencies maintained in the paths used to service logical-record requests.

Accessing both logical records and database records in the same program can diminish the program’s independence from the database structure and possibly interfere with the execution of paths invoked to provide requested logical-record access.

Logical-record path processing can interfere with program access to database records. You may need to insert a DML statement after a logical-record request to reestablish the appropriate currency.

The precompiler inserts the required data descriptions into the program automatically unless IDMS RECORDS MANUAL is specified in the IDMS-CONTROL SECTION of the ENVIRONMENT DIVISION. If IDMS RECORDS MANUAL is specified, you must explicitly copy the required components, as outlined above, by coding COPY IDMS statements in the DATA DIVISION.

**UTM modes only:** You must include SUBSCHEMA-CTRL and all subschema records in the LINKAGE SECTION. You must include SUBSCHEMA-NAMES in the WORKING-STORAGE SECTION.

**Syntax**

```
| WORKING STORAGE SECTION. |
| LINKAGE SECTION. |
| level-number |
| COPY-IDMS |
| SUBSCHEMA-DML-LR-DESCRIPTION |
| SUBSCHEMA-DESCRIPTION |
| SUBSCHEMA-CTRL |
| SUBSCHEMA-NAMES |
| SUBSCHEMA-SSNAME |
| SUBSCHEMA-RECNAMES |
| SUBSCHEMA-SETNAMES |
| SUBSCHEMA-AREANAMES |
| SUBSCHEMA-RECORDS |
| RECORD rec-name |
| TRANSACTION-STATISTICS |
| SUBSCHEMA-LR-DESCRIPTION |
| SUBSCHEMA-LR-CONTROL |
| SUBSCHEMA-LR-CTRL |
| SUBSCHEMA-LR-RECORDS |
| LR logical-record-name |
| MAPS |
| MAP map-name |
| MAP CONTROLS |
| MAP CONTROL map-name |
| MAP RECORDS |
```

**Parameters**

- **level-number**
  An integer in the range 01 through 48.
Instructs the precompiler to copy the descriptions into the program at a level other than that originally specified for the description in the data dictionary. If you specify a level number, the first level of code will be copied to the specified level; all other levels will be adjusted accordingly. If you do not specify a level, the descriptions copied will begin at 01 and have the same level numbers as originally specified in the data dictionary.

**Note:** Using the `level-number` clause can cause unpredictable results if record fields have been defined with a SYNCHRONIZED clause. Such fields may contain slack bytes, inserted to ensure correct alignment. Because CA IDMS does not regard slack bytes as functional, it may misrepresent fields that contain such bytes. Therefore, you should ensure that all fields and records are properly structured.

- **COPY IDMS**
  Requests that the specified source data description code be copied into the DATA DIVISION at the location of the COPY IDMS statement.

- **SUBSCHEMA-DML-LR-DESCRIPTION**
  Copies all components required to access both database and logical records (SUBSCHEMA-CTRL, SUBSCHEMA-NAMES, SUBSCHEMA-RECORDS, SUBSCHEMA-LR-CTRL, SUBSCHEMA-LR-RECORDS). SUBSCHEMA-DML-LR-DESCRIPTION should be specified only when the subschema’s usage mode is MIXED; do not specify SUBSCHEMA-DML-LR-DESCRIPTION if the usage mode is DML or LR.

- **SUBSCHEMA-DESCRIPTION**
  Copies all components required to access database records (SUBSCHEMA-CTRL, SUBSCHEMA-NAMES, and SUBSCHEMA-RECORDS). Do not specify SUBSCHEMA-DESCRIPTION if the subschema’s usage mode is LR.

- **SUBSCHEMA-CONTROL**
  Copies both the SUBSCHEMA-CTRL and SUBSCHEMA-NAMES components. Do not specify SUBSCHEMA-CONTROL if the subschema’s usage mode is LR.

- **SUBSCHEMA-CTRL**
  Copies the IDMS communications block data description; if the operating mode is IDMS-DC or DC-BATCH, SUBSCHEMA-CTRL copies the IDMS-DC communications block.

- **SUBSCHEMA-NAMES**
  Copies the eight-character literal name of the subschema and the literal names of all database records, sets, and areas contained in the subschema. SUBSCHEMA-NAMES includes SUBSCHEMA-SSNAME, SUBSCHEMA-RECNAMES, SUBSCHEMA-SETNAMES, and SUBSCHEMA-AREANAMES. Do not specify SUBSCHEMA-NAMES if the subschema’s usage mode is LR.

- **SUBSCHEMA-SSNAME**
  Copies the eight-character literal name of the program’s subschema. Do not specify SUBSCHEMA-SSNAME if the subschema’s usage mode is LR.

- **SUBSCHEMA-RECNAMES**
  Copies the literal names of all database records contained in the subschema. Do not specify SUBSCHEMA-RECNAMES if the subschema’s usage mode is LR.
- **SUBSCHEMA-SETNAMES**
  Copies the literal names of all sets contained in the subschema. Do not specify SUBSCHEMA-SETNAMES if the subschema's usage mode is LR.

- **SUBSCHEMA-AREANAMES**
  Copies the literal names of all database areas that can be accessed through the subschema. Do not specify SUBSCHEMA-AREANAMES if the subschema's usage mode is LR.

- **SUBSCHEMA-RECORDS**
  Copies the descriptions of all records contained in the subschema. COBOL synonyms defined for the subschema records in the data dictionary may be copied into the program, according to the rules of synonym usage. Do not specify SUBSCHEMA-RECORDS if the subschema's usage mode is LR.

  **Note:** The OCCURS DEPENDING ON clause will be commented out for all schema-owned records. Therefore, although the maximum length of variable storage will be reserved, only the correct amount of data will be transferred to variable storage at run time.

Since COBOL will doubleword align an 01 level record, the precompiler adds up to seven bytes, if necessary, to make the record length divisible by eight when copying in a schema-owned record to an 01 level.

- **RECORD**
  Copies the description of a record defined in the data dictionary. If the subschema's usage mode is LR, only copy in IDD work records.

  **rec-name**
  The name of the record to be copied. Either the primary name or a synonym for a record stored in the data dictionary.
  Schema-owned records cannot be copied into non-IDMS programs (that is, programs that do not use a subschema and that do not access the database). However, a synonym defined for a schema-owned record can be copied into a non-IDMS program (use the VERSION clause to identify the synonym).
  **IDD records:** If an operating mode is associated with record-name in the data dictionary, it must agree with the mode in effect for the program (see "ENVIRONMENT DIVISION" earlier in this section).

- **VERSION**
  Optionally qualifies IDD records (but not schema-owned records) with a version number.
  If you do not specify a version number, the default is the highest version number defined in the data dictionary for the language and operating mode under which the program is being compiled.
  When copying a record that is schema owned using a synonym name, a version clause is needed, even if the synonym is not schema owned. The only time the version clause may be left off when copying a record using a synonym name is when the record is IDD owned. Once a record becomes schema owned, version clauses are needed.

  **vers-num**
  An integer in the range 1 through 9999.
  You cannot specify a version number for a rec-name specified in the subschema named in the DB subschema-name statement. The precompiler will automatically copy the correct version into the program.
• **REDEFINES**
  Copies a record description to an area previously defined by another record description. Two record descriptions can thus provide alternative definitions of the same storage location.

  • **rec-name**
    The name of the record to be redefined.

• **TRANSACTION-STATISTICS**
  Copies the definition of the transaction statistics block (TSB) with a length of 560 bytes. This block can be used in the ACCEPT TRANSACTION STATISTICS or END TRANSACTION STATISTICS DML statements.

• **SUBSCHEMA-LR-DESCRIPTION**
  Copies all components required to access logical records (SUBSCHEMA-CTRL, SUBSCHEMA-LR-CTRL, SUBSCHEMA-LR-NAMES, and SUBSCHEMA-LR-RECORDS). Do not include SUBSCHEMA-LR-DESCRIPTION if the subschema's usage mode is DML.

• **SUBSCHEMA-LR-CONTROL**
  Copies the SUBSCHEMA-CTRL, SUBSCHEMA-LR-CTRL, and SUBSCHEMA-LR-NAMES components. Do not include SUBSCHEMA-LR-CONTROL if the subschema's usage mode is DML.

• **SUBSCHEMA-LR-CTRL**
  Copies the LRC block data description.

• **SIZE IS**
  Specifies the size of that portion of the LRC block that contains information about the logical-request request WHERE clause (PXE).
  If included, this parameter should specify a size large enough to accommodate the most complex WHERE clause in the program. The default, 512, is large enough to include approximately 32 operators, operands, and literals.
  Do not include SUBSCHEMA-LR-CTRL if the subschema's usage mode is DML.

  • **lrc-block-size**
    A positive integer in the range 0 through 9999.
    Calculate the size as follows:
    - Multiply the greatest number of operands and operators that will be included in a single WHERE clause by 16 bytes
    - Add the number of bytes, rounded up to the nearest multiple of 8, associated with the data field for each operand that is a keyword or a program variable or logical-record field named in the OF LR clause
    - Add the length, rounded up to the nearest multiple of eight, of each operand that is a character literal
    - Add 12 bytes for each operand that is a numeric literal
    - Do not specify a block size if none of the logical-record requests issued by the program will include WHERE clauses.
- **SUBSCHEMA-LR-NAMES**
  Copies the literal name of the program's subschema and the literal names of all database areas that can be accessed through the subschema. Logical-record names are not copied into the program. Do not include SUBSCHEMA-LR-NAMES if the subschema's usage mode is DML.

- **SUBSCHEMA-LR-RECORDS**
  Copies the descriptions of all logical records defined in the subschema. All participating database records become 02-level group fields, permitting the program to reference as a group field that portion of a logical record that corresponds to a database record. Do not include SUBSCHEMA-LR-RECORDS if the subschema's usage mode is DML.

  **Note:** The OCCURS DEPENDING ON clause will be commented out for all schema-owned records. Therefore, although the maximum length of variable storage will be reserved, only the correct amount of data will be transferred to variable storage at runtime.

When copying a schema-owned record to a level other than 01, the precompiler adds up to seven bytes, if necessary, to make the record length divisible by eight for doubleword alignment.

- **LR**
  Copies the description of an individual logical record contained in the subschema.

- **logical-record-name**
  The name of the logical record to copy.

- **REDEFINES**
  Copies a redefinition of the data contained in another logical record, a database record, or a non-IDMS record, while maintaining the same location in variable storage. Do not include this statement if the subschema's usage mode is DML.

- **record-name**
  The name of the record to be redefined.

- **MAPS**
  Copies the map request block (MRB) and map records associated with all maps defined in the MAP SECTION.

- **MAP map-name**
  Copies the MRB and map records associated with the named map. The map version number defaults to the version specified for the map in the MAP SECTION.

- **MAP-CONTROLS**
  Copies the MRBs associated with all maps specified in the MAP SECTION.

- **MAP-CONTROL map-name**
  Copies the MRB for the named map. The map version number defaults to the version specified for the map in the MAP SECTION.

- **MAP-RECORDS**
  Copies the map records associated with all maps specified in the MAP SECTION.
Results of COPY IDMS Specifications

The following figure shows the code copied into the DATA DIVISION as a result of COPY IDMS specifications.
PROCEDURE DIVISION

The COPY IDMS statements in the PROCEDURE DIVISION allow inclusion into the source program of BIND statements for CA IDMS records and for procedure source statements defined as modules in the data dictionary by the DBA.

Syntax

\[
\begin{align*}
\text{PROCEDURE DIVISION.} & \quad \text{COPY IDMS SUBSCHEMA-BINDS.} \\
\text{COPY IDMS SUBSCHEMA-RECORD-BINDS.} & \\
\text{COPY IDMS MAP-BINDS.} & \\
\text{COPY IDMS module module-name.} & \quad \text{VERSION version-number} \\
\end{align*}
\]

Parameters

- **COPY IDMS SUBSCHEMA-BINDS**
  Initializes the PROGRAM-NAME field in the IDMS communications block and copies a standard BIND RUN-UNIT statement and appropriate standard BIND record-name commands for each CA IDMS record in the program’s DATA DIVISION. COPY IDMS SUBSCHEMA-BINDS does not generate BIND RECORD statements for logical records, nor are any needed.
  In cases where more than one copy of a given database record description (including synonyms) is present in the program, COPY IDMS SUBSCHEMA-BINDS will not automatically generate bind record statements. Individual bind record statements must be issued to bind the record to the correct location.
  If IDMS-RECORDS MANUAL has been specified in the ENVIRONMENT DIVISION, the COPY IDMS SUBSCHEMA-BINDS statement generates BINDS only for subschema records explicitly copied into the DATA DIVISION by means of COPY IDMS statements; it does not automatically generate BINDS for all subschema records.
  Do not use the COPY IDMS SUBSCHEMA-BINDS statement when binding several records to the same location. Instead, code DML BIND statements in the PROCEDURE DIVISION for each record.
  For more information, see BIND RECORD (COBOL) (see page 108).

**Note:** If AUTOSTATUS is in use, a PERFORM IDMS-STATUS occurs automatically after each BIND generated by a COPY IDMS SUBSCHEMA-BINDS statement. If AUTOSTATUS is not in use, you should explicitly code the BIND RUN-UNIT and BIND RECORD statements so that a PERFORM IDMS-STATUS can be coded after each BIND.

For more information about AUTOSTATUS, see AUTOSTATUS Protocols (see page 56).
COPY IDMS SUBSCHEMA-RECORD-BINDS
Copies appropriate standard BIND record-name commands for each CA IDMS record in the program's DATA DIVISION.
In cases where more than one copy of a given database record description (including synonyms) is present in the program, COPY IDMS SUBSCHEMA-RECORD-BINDS will not automatically generate bind record statements. Individual bind record statements must be issued to bind the record to the correct location.
If IDMS-RECORDS MANUAL has been specified in the ENVIRONMENT DIVISION, the COPY IDMS SUBSCHEMA-RECORD-BINDS statement generates BINDS only for subschema records explicitly copied into the DATA DIVISION by means of COPY IDMS statements; it does not automatically generate BINDS for all subschema records.
Do not use the COPY IDMS SUBSCHEMA-RECORD-BINDS statement when binding several records to the same location. Instead, code DML BIND statements in the PROCEDURE DIVISION for each record.
Note: If AUTOSTATUS is in use, a PERFORM IDMS-STATUS occurs automatically after each BIND generated by a COPY IDMS SUBSCHEMA-BINDS statement. If AUTOSTATUS is not in use, you should explicitly code the BIND RUN-UNIT and BIND RECORD statements so that a PERFORM IDMS-STATUS can be coded after each BIND.
COPY IDMS MAP-BINDS
Copies map- and map-record-specific BIND MAP statements for all maps in the program's MAP SECTION. For more information see BIND MAP (COBOL) (see page 105).
COPY IDMS module
Copies source statements from a module stored in the data dictionary into the source program. The unmodified module is placed into the program by the precompiler at the location of the request. The module can, but need not, contain DML statements. Any DML statements will be examined and expanded within the context of the program's subschema view and compile mode as if they were coded directly.
COPY IDMS MODULE statements can be nested (that is, code invoked by a COPY IDMS MODULE entry can itself contain a COPY IDMS MODULE statement). However, you must ensure that a copied module does not, in turn, copy itself.
module-name
The name of a module previously defined by the DBA by means of the IDD DDDL compiler. The following standard modules are available for COBOL programs:
- IDMS-STATUS

Note: The IDMS-STATUS module must be copied into the program if an AUTOSTATUS protocol is in effect, as specified in the IDMS-CONTROL SECTION of the ENVIRONMENT DIVISION.
- IDMS-STATUS (BATCH-AUTOSTATUS)
- IDMS-STATUS (DC)
- IDMS-WAIT (DC)
- IDMS-WAIT (CICS)
IDMS-WAIT (CICS STANDARD)

IDMS-WAIT (CICS AUTOSTATUS)

IDMS-WAIT (CICS STANDARD AUTOSTATUS)

⚠️ **Note:** The IDMS-STATUS module must be copied into the program if an AUTOSTATUS protocol is in effect, as specified in the IDMS-CONTROL SECTION of the ENVIRONMENT DIVISION.

- **VERSION**
  Optionally qualifies *module-name* with a version number.
  If you do not specify a version number, the default is the highest version number defined in the data dictionary for the language mode under which the program is being compiled (for example, BATCH or IDMS-DC).
  If no mode-specific version exists for *module-name*, the non-mode-specific version (if present) is copied. If neither a mode-specific entry nor a non-mode-specific entry for *module-name* has been established, an error results. The same rules apply to the module's language (that is, *version-number* defaults to the highest value defined in the data dictionary for the language in which the program is written).

- **version-number**
  An integer in the range 1 through 9999.
  By default, if you do not specify a version number, the highest value defined in the data dictionary will be used.
COBOL Data Manipulation Language (DML) Statements

CA IDMS data manipulation language (DML) consists of statements that enable you to access the database management system (DBMS) and to request Logical Record Facility (LRF) and DC system services. The following table lists the DML statements by function.

**Note:** Consider the following items:

- All DML operands are positional
- (o) applies to online environment
- (dcb) applies to DC-BATCH
- (o,dcb) applies to online environment or DC-BATCH

<table>
<thead>
<tr>
<th>DML Function</th>
<th>Description</th>
<th>Statements</th>
</tr>
</thead>
</table>
| Accept       | Move special information such as database keys, storage addresses, and statistics from the DBMS to program variable storage. | | - ACCEPT DB-KEY FROM CURRENCY  
Saves the db-key of the current record of run unit, record type, set, or area.  
- ACCEPT DB-KEY RELATIVE TO CURRENCY  
Saves the db-key of the next, prior, or owner record relative to the current record of a set.  
- ACCEPT IDMS STATISTICS  
Returns system run-time statistics to the program.  
- ACCEPT BIND RECORD  
Returns the bind address of a record to the program.  
- ACCEPT PROCEDURE  
Returns information from the application program information block associated with a database procedure to the program |
| Control      | Initiate and terminate processing.  
Effect recovery.  
Prevent concurrent retrieval and update of database records.  
Evaluate set conditions. | | - BIND RUN-UNIT  
Signs on the application program to the DBMS.  
- BIND TASK  
Establishes a connection with the DC/UCF system from a batch program. Also allows certain online functions, such as writing to queues or printing to a printer controlled by the DC/UCF system (dcb). |
<table>
<thead>
<tr>
<th>DML Function</th>
<th>Description</th>
<th>Statements</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIND RECORD</td>
<td>Establishes addressability in variable storage for one or more records that are included in the program subschema.</td>
<td></td>
</tr>
<tr>
<td>BIND PROCEDURE</td>
<td>Establishes communication between the application program and a DBA-defined database procedure.</td>
<td></td>
</tr>
<tr>
<td>READY</td>
<td>Prepares database areas for processing.</td>
<td></td>
</tr>
<tr>
<td>FINISH</td>
<td>Commits changes that were made to the database through an individual run unit or through all database sessions that are associated with a task.</td>
<td></td>
</tr>
<tr>
<td>IF</td>
<td>Evaluates the presence of records in a set or the membership status of a record. Also specifies what action to take based on the outcome.</td>
<td></td>
</tr>
<tr>
<td>COMMIT</td>
<td>Commits changes that were made to the database through an individual run unit or through all database sessions that are associated with a task.</td>
<td></td>
</tr>
<tr>
<td>ROLLBACK</td>
<td>Rolls back uncommitted changes that were made to the database through an individual run unit or through all database sessions that are associated with a task.</td>
<td></td>
</tr>
<tr>
<td>KEEP CURRENT</td>
<td>Places an explicit shared or exclusive lock on a record that is current of run unit, record, set, or area.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Logical-record</th>
<th>Retrieve, modify, store, and erase logical records.</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERASE</td>
<td>Deletes a logical record.</td>
</tr>
<tr>
<td>MODIFY</td>
<td>Modifies a logical record.</td>
</tr>
<tr>
<td>OBTAIN</td>
<td>Accesses a logical record.</td>
</tr>
<tr>
<td>STORE</td>
<td>Stores a logical record.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Modification</th>
<th>Add new records to the database and modify and delete existing records.</th>
</tr>
</thead>
<tbody>
<tr>
<td>STORE</td>
<td>Adds a new record to the database.</td>
</tr>
<tr>
<td>MODIFY</td>
<td>Changes the contents of an existing record.</td>
</tr>
<tr>
<td>DML Function</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------</td>
</tr>
<tr>
<td>CONNECT</td>
<td>Links a record to a set.</td>
</tr>
<tr>
<td>DISCONNECT</td>
<td>Removes a member record from a set.</td>
</tr>
<tr>
<td>ERASE</td>
<td>Deletes a record from the database.</td>
</tr>
</tbody>
</table>

**Program Management**
- Pass and return control from one program to another.
- Load and delete programs and tables.
- Define exit routines to perform before an abnormal program termination (abend).
- Force an abend condition.
- TRANSFER CONTROL (LINK)
  Passes control to another program with the expectation of receiving it back (o).
- TRANSFER CONTROL (XCTL)
  Passes control to another program with no expectation of receiving it back (o).
- DC RETURN
  Returns control to the next higher level calling program (o).
- LOAD TABLE
  Loads a program or table into the CA IDMS system program pool (o).
- DELETE TABLE
  Signals that a program has finished using a program or a table in the program pool (o).
- SET ABEND EXIT (STAE)
  Establishes linkage to a program or routine that receives control when an abend occurs (o).
- ABEND
  Abnormally terminates the issuing task (o).

**Queue Management**
- Create, delete, or retrieve records from the queue area.
- PUT QUEUE
  Stores a queue record (o,dcb).
- GET QUEUE
  Retrieves a queue record (o,dcb).
- DELETE QUEUE
  Deletes a queue record (o,dcb).

**Recovery**
- Perform the following functions relating to database, scratch, and queue area recovery when a system failure occurs:
  - Establish checkpoints in the journal file for database, scratch, and queue records that the issuing task is using.
  - Roll back user database, scratch, and queue areas to the last established checkpoint.
- COMMIT
  Commits changes that were made to the database through an individual run unit or through all database sessions that are associated with a task.
- FINISH
  Commits changes that were made to the database through an individual run unit or through all database sessions that are associated with a task.
### DML Function Description

<table>
<thead>
<tr>
<th>DML Function</th>
<th>Description</th>
<th>Statements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Establish an end-of-task checkpoint and relinquish control of all database, scratch, and queue areas that are associated with the issuing task.</td>
<td>ROLLBACK</td>
<td>Rolls back uncommitted changes that were made to the database through an individual run unit or through all database sessions that are associated with a task.</td>
</tr>
<tr>
<td>Write user-defined records to the journal file.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| Retrieval | Locate records in the database and make them available to the application program. | | |
|------------|-------------------------------------------------|-----------------|
| FIND/OBTAIN DB-KEY | Accesses a record using a db-key that the program previously saved. | FIND/OBTAIN CURRENT | Accesses a record using previously established currencies. |
| FIND/OBTAIN CURRENT | Accesses a record using previously established currencies. | FIND/OBTAIN WITHIN SET/AREA | Accesses a record based on its logical location within a set or its physical location within an area. |
| FIND/OBTAIN OWNER | Accesses the owner record of a set occurrence. | FIND/OBTAIN CALC/DUPLICATE | Accesses a record using its CALC-key value. |
| FIND/OBTAIN USING SORT KEY | Accesses a record in a sorted set using its sort-key value. | GET | Moves all data that is associated with a previously located record into program variable storage. |
| RETURN | Retrieves the database key and symbolic key of an indexed record entry. | | |

| Scratch management | Create, delete, or retrieve records from the scratch area. | | |
|-------------------|-------------------------------------------------|-----------------|
| PUT SCRATCH | Stores a scratch record (o). | GET SCRATCH | Retrieves a scratch record (o). |
| DELETE SCRATCH | Deletes a scratch record (o). | | |

| Storage management | Allocate and release variable storage. | | |
|--------------------|-------------------------------------------------|-----------------|
| GET STORAGE | Allocates variable storage from a CA IDMS system storage pool (o). | FREE STORAGE | Frees all or part of a block of variable storage (o). |

- Initiate a new task.
## Task Management

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Statements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attach</td>
<td>Change the dispatching priority of the issuing task.</td>
<td>ATTACH</td>
</tr>
<tr>
<td></td>
<td>Enqueue and dequeue system resources.</td>
<td>CHANGE PRIORITY</td>
</tr>
<tr>
<td></td>
<td>Signal that a task is to wait pending completion of an event.</td>
<td>ENQUEUE</td>
</tr>
<tr>
<td></td>
<td>Post an event control block (ECB) indicating completion of an event.</td>
<td>DEQUEUE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>WAIT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>POST</td>
</tr>
</tbody>
</table>

## Terminal Management (Basic Mode)

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Statements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read Terminal</td>
<td>Requests a synchronous or asynchronous data transfer from the terminal to program variable storage (o).</td>
<td>READ TERMINAL</td>
</tr>
<tr>
<td>Write Terminal</td>
<td>Requests a synchronous or asynchronous data transfer from program variable storage to the terminal buffer (o).</td>
<td>WRITE TERMINAL</td>
</tr>
<tr>
<td>Write Then Read Terminal</td>
<td>Requests a synchronous or asynchronous data transfer from program variable storage to the terminal buffer and, upon an operator signal, back to variable storage (o).</td>
<td>WRITE THEN READ TERMINAL</td>
</tr>
<tr>
<td>Check Terminal</td>
<td>Ensures that a previously issued asynchronous I/O operation is complete (o).</td>
<td>CHECK TERMINAL</td>
</tr>
</tbody>
</table>

## Terminal Management (Line Mode)

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Statements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read Line From Terminal</td>
<td>Requests a synchronous data transfer from the terminal to the issuing program (o).</td>
<td>READ LINE FROM TERMINAL</td>
</tr>
<tr>
<td>Write Line To Terminal</td>
<td>Requests a synchronous or asynchronous data transfer from the issuing program to the terminal (o).</td>
<td>WRITE LINE TO TERMINAL</td>
</tr>
<tr>
<td>End Line Terminal Session</td>
<td>Terminates the current line I/O session (o).</td>
<td>END LINE TERMINAL SESSION</td>
</tr>
<tr>
<td>Write Printer</td>
<td>Requests transmission of data from a task to a printer (o, dcb).</td>
<td>WRITE PRINTER</td>
</tr>
</tbody>
</table>

## Transfer data between the application program and the terminal.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Statements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bind Map</td>
<td>Identifies the location of the map request block (MRB) and initializes the MRB fields (o).</td>
<td>BIND MAP</td>
</tr>
<tr>
<td>DML Function</td>
<td>Description</td>
<td>Statements</td>
</tr>
<tr>
<td>------------------------------------</td>
<td>------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Terminal Management (Mapping Mode)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MAP IN</td>
<td>Requests a transfer of data</td>
<td>MAP IN Requests a transfer of data from the terminal to program variable</td>
</tr>
<tr>
<td>GET TIME</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MAP OUT</td>
<td>Requests a transfer of data</td>
<td>MAP OUT Requests a transfer of data from program variable storage to the</td>
</tr>
<tr>
<td>SET TIMER</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MAP OUTIN</td>
<td>Requests a transfer of data</td>
<td>MAP OUTIN Requests a transfer of data from program variable storage to the</td>
</tr>
<tr>
<td>ENDPAGE</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>INQUIRE MAP Obtains information or tests conditions concerning the previous</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>MODIFY MAP Requests modifications of mapping options for a map (o).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>STARTPAGE Begins a map paging session and specifies options for that</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>ENDPAGE Terminates a map paging session (o).</td>
</tr>
<tr>
<td>Time Management</td>
<td>Obtain the time and date,</td>
<td>GET TIME Obtains the time and date from the system.</td>
</tr>
<tr>
<td></td>
<td>and define time-related</td>
<td>SET TIMER Defines a time-delayed event (o).</td>
</tr>
<tr>
<td></td>
<td>events. These events</td>
<td></td>
</tr>
<tr>
<td></td>
<td>include:</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></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Utility function</td>
<td>Request retrieval of task-</td>
<td>ACCEPT Retrieves task-related information (o).</td>
</tr>
<tr>
<td></td>
<td>related information.</td>
<td>SNAP Requests a memory dump of selected parts of storage (o).</td>
</tr>
<tr>
<td></td>
<td>Request a memory dump of</td>
<td>SEND MESSAGE Sends a message to a user, logical terminal, or list of users</td>
</tr>
<tr>
<td></td>
<td>selected parts of storage.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Retrieve and send a predefined</td>
<td>BIND TRANSACTION STATISTICS Defines the beginning of a transaction for the</td>
</tr>
<tr>
<td></td>
<td>message that is stored in the</td>
<td></td>
</tr>
<tr>
<td></td>
<td>data dictionary.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Send a specified message to</td>
<td>ACCEPT TRANSACTION STATISTICS Returns the contents of the transaction</td>
</tr>
<tr>
<td></td>
<td>one or more users or logical</td>
<td></td>
</tr>
<tr>
<td></td>
<td>terminals.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Collect, retrieve, and write</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CA IDMS statistics on a</td>
<td></td>
</tr>
<tr>
<td></td>
<td>transaction basis.</td>
<td></td>
</tr>
</tbody>
</table>
DML Function | Description | Statements
--- | --- | ---
 | - Establish long-term database locks and monitor access to the database records that are used across tasks during a pseudo-conversational transaction. | - END TRANSACTION STATISTICS
- Defines the end of a transaction (o).
- KEEP LONGTERM
- Modifies a prior KEEP LONGTERM request, or enables database long-term locks or database monitoring for records, sets, or areas.
- WRITE JOURNAL
- Writes user-defined records to the journal file.
- WRITE LOG
- Retrieves a message from the data dictionary and sends it to a predefined destination (o).

ABEND (COBOL)

The ABEND statement terminates the issuing task abnormally. This statement also invokes or ignores previously established abend exits and writes a task dump to the log file. After completion of the ABEND function, processing control is returned to the CA IDMS program control module.

Syntax

```
ABEND CODE abend-code
       DUMP NODUMP       EXITS INVOKED IGNORED
```

Parameters

- **ABEND CODE abend-code**
  Specifies a four-character user-defined abend code. The code can be the symbolic name of a variable-storage field that contains the abend code. You can also specify the abend code itself, enclosed in quotation marks.

  **Note:** Do not use CA IDMS system abend codes. The abend code that you specify here appears in the system log and also at the task's terminal.

- **DUMP | NODUMP**
  Specifies whether to write a formatted task dump to the log file. The default value is NODUMP.

- **EXITS INVOKED | IGNORED**
  Specifies whether to invoke or ignore abend routines established by SET ABEND EXIT (STAE) requests. The default value is INVOKED.
Example

The following syntax terminates the issuing task abnormally, writes a task dump to the log file, and ignores any abend exits:

```plaintext
ABEND CODE 'U876'
DUMP
EXITS IGNORED.
```

Status Codes

Because control is passed to the CA IDMS program control module, the program does not check the ERROR-STATUS field.

ACCEPT (COBOL)

The ACCEPT statement retrieves the following task-related information:

- Current task code
- Task identifier
- Logical terminal identifier
- Physical terminal identifier
- CA IDMS system version
- User identifier (the ID of the user signed on to the task's logical terminal)
- Physical terminal screen dimensions
- System ID

Syntax

```
ACCEPT TASK CODE INTO return-location.
```

Parameters

- **ACCEPT**
  Retrieves the specified information. The following values are valid:
CA IDMS Reference - 19.0

- **TASK CODE**
  Retrieves the one- to eight-character code that is used to invoke the current task.

- **TASK ID**
  Retrieves the task identifier that CA IDMS assigned. The task identifier is a unique sequence number stored in a PIC S9(8) COMP SYNC (fullword) field. At system startup, the identifier is set to zero. Each time a task is executed, the ID increments by one.

- **LTERM ID**
  Retrieves the one- to eight-character identifier of the logical terminal that is associated with the current task. If the current task is not associated with a terminal, a null value of all spaces is returned.

- **PTERM ID**
  Retrieves the one- to eight-character identifier of the physical terminal that is associated with the current task. If the current task is not associated with a terminal, a null value of all spaces is returned.

- **SYSVERSION**
  Retrieves the version of the current CA IDMS system. The version number is an integer from 0 to 9999 stored in a PIC S9(4) COMP (halfword) field.

- **USER ID**
  Retrieves the 32-character identifier of the user that is signed on to the logical terminal associated with the current task. If no user is signed on, a null value of all spaces is returned.

- **SCREENSIZE**
  Retrieves the screen dimensions of the physical terminal that is associated with the current task. The screen size is returned to a field that is divided into two PIC S9(4) COMP (halfword) fields. The first halfword contains the row. The second halfword contains the column. For example, a 24-line by 80-character screen is represented by a value of 24 in the first halfword and a value of 80 in the second halfword. If the current task is not associated with a terminal, a null value of zero is returned.

- **SYSTEM ID**
  Specifies the eight-character name (nodename) by which the DC/UCF system is known to other nodes in the DC/UCF communications network.

- **INTO return-location**
  Specifies where CA IDMS returns the requested task-related information. The *return-location* is the symbolic name of a user-defined field. The picture and usage of this field must be compatible with the picture and usage of the requested data.

**Example**

The following syntax retrieves the ID of the current task and the user ID that is signed on to the logical terminal associated with that task:

```
ACCEPT TASK ID INTO TASK-ID.
ACCEPT USER ID INTO USER-ID.
```
Status Codes

After completion of the ACCEPT function, the ERROR-STATUS field in the IDMS-DC communications block indicates the outcome:

<table>
<thead>
<tr>
<th>Status code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>The request has been serviced successfully.</td>
</tr>
<tr>
<td>4829</td>
<td>An invalid parameter has been passed from the program.</td>
</tr>
</tbody>
</table>

ACCEPT BIND ADDRESS (COBOL)

The ACCEPT BIND ADDRESS statement moves the bind address of a record to a specified location in program variable storage. This statement is typically requested by a subprogram that requires the address of a record to access it.

**Currency:** The ACCEPT BIND ADDRESS statement updates no currencies and requires no currencies to be set relative to the specified record.

Syntax

```plaintext
►►─── ACCEPT bind-address FROM record-name BIND . ───────────────────────────►◄
```

Parameters

- **ACCEPT bind-address**
  Specifies where to copy the bind address of the specified record. Specify a PIC 9(8) COMP SYNC (fullword) field.

- **FROM ... BIND**
  Specifies the record whose bind address will be copied into the specified location in variable storage. The following value is valid:

  - **record-name**
    Specifies the name of a record that was previously bound by the run unit.

Example

The following statement moves the bind address for EMPLOYEE to a location identified as REG1 in the requesting subprogram:

```
ACCEPT REG1 FROM EMPLOYEE BIND.
```
Status Codes

After completion of the ACCEPT BIND ADDRESS function, the ERROR-STATUS field in the IDMS communications block indicates the outcome:

<table>
<thead>
<tr>
<th>Status code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>The request has been serviced successfully.</td>
</tr>
<tr>
<td>1508</td>
<td>The named record is not in the specified subschema.</td>
</tr>
</tbody>
</table>

ACCEPT DATABASE STATISTICS (COBOL)

The ACCEPT DATABASE STATISTICS statement copies system runtime statistics from the IDMS statistics block to program variable storage. This statement can be issued multiple times during the execution of a run unit. For example, you can request database statistics after storing a variable-length record to determine whether the entire record was stored in one place or whether fragments were placed in an overflow area.

The ACCEPT DATABASE STATISTICS statement does not reset any statistics fields to zero. Resetting of the IDMS statistics block fields occurs only upon issuing a FINISH command.

The ACCEPT DATABASE STATISTICS statement is used in both navigational and the nonnavigational environments.

Syntax

```plaintext
ACCEPT db-statistics FROM IDMS-STATISTICS .
EXTENDED (db-stat-extended) ;
```

Parameters

- **db-statistics**
  
  Specifies a fullword-aligned 100-byte field in program variable storage. The data that is copied from IDMS-STATISTICS to *db-statistics* is formatted as follows:

```plaintext
01 DB-STATISTICS
  03 DATE-TODAY   PIC X(8).
  03 TIME-TODAY   PIC X(8).
  03 PAGES-READ   PIC S9(8) COMP.
  03 PAGES-WRITTEN PIC S9(8) COMP.
  03 PAGES-REQUESTED PIC S9(8) COMP.
  03 CALC-TARGET  PIC S9(8) COMP.
  03 CALC-OVERFLOW PIC S9(8) COMP.
  03 VIA-TARGET   PIC S9(8) COMP.
  03 VIA-OVERFLOW PIC S9(8) COMP.
  03 LINES-REQUESTED PIC S9(8) COMP.
  03 RECS-CURRENT PIC S9(8) COMP.
  03 CALLS-TO-IDMS PIC S9(8) COMP.
```
The LOCAL-ID field consists of the 4-byte identifier of the interface in which the run unit originated (for example, BATC, DBDC, or CICS) and a unique identifier (fullword) assigned to the run unit by that interface. For batch and CMS run units, this identifier specifies the internal machine time. For the CICS run units, this identifier specifies the CICS transaction number assigned to the run unit.

To display the originating interface identifier and the run unit identifier for a program, move LOCAL-ID to a work field:

```
01 WORK-LOCAL-ID.
 02 WORK-LOCAL-ORIGIN PIC X(4).
 02 WORK-LOCAL-NUMBER PIC S9(8) COMP.
```

Alternatively, your DBA can modify the DB-STATISTICS record from the data dictionary to define two subordinate fields for the LOCAL-ID field. The DB-STATISTICS record describes the IDMS statistics block. To use this record, code the following statement in program variable storage:

```
01 COPY IDMS DB-STATISTICS.
```

- **db-stat-extended**
  
  Specifies the name of a fullword-aligned 100-byte field in program variable storage. The data that is copied from IDMS-STATISTICS to *db-stat-extended* is formatted as follows:

```
01 DB-STAT-EXTENDED
 03 SR8-SPLITS PIC S9(8) COMP.
 03 SR8-SPAWNNS PIC S9(8) COMP.
 03 SR8-STORES PIC S9(8) COMP.
 03 SR8-ERASES PIC S9(8) COMP.
 03 SR7-STORES PIC S9(8) COMP.
 03 SR7-ERASES PIC S9(8) COMP.
 03 BINARY-SEARCHES-TOTAL PIC S9(8) COMP.
 03 LEVELS-SEARCHED-TOTAL PIC S9(8) COMP.
 03 ORPHANS-ADOPTED PIC S9(8) COMP.
 03 LEVELS-SEARCHED-BEST PIC S9(4) COMP.
 03 LEVELS-SEARCHED-WORST PIC S9(4) COMP.
 03 FILLER PIC X(60).
```

This record layout can be copied from the data dictionary. Code the following statement in program variable storage:

```
01 INCLUDE IDMS (DB_STAT_EXTENDED).
```

**Note:** For more information about the CA IDMS statistics blocks, see the *CA IDMS Database Administration* section.
Example

The following statements perform the following tasks:

- Establish currency for the sets in which a new EXPERTISE record will participate as a member.
- Store the EXPERTISE record.
- Move statistics regarding the stored EXPERTISE record to the DB-STATISTICS location in main storage.

```
MOVE EMP-ID-IN TO EMP-ID-0415.
FIND CALC EMPLOYEE.
MOVE SKILL-ID-IN TO SKILL-ID-0455.
FIND CALC SKILL.
STORE EXPERTISE.
ACCEPT DB-STATISTICS FROM IDMS-STATISTICS.
```

Status Codes

After completion of the ACCEPT DATABASE STATISTICS function, the ERROR-STATUS field in the IDMS communications block indicates the outcome:

<table>
<thead>
<tr>
<th>Status code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>The request has been serviced successfully.</td>
</tr>
<tr>
<td>1518</td>
<td>The database statistics location has not been bound properly.</td>
</tr>
</tbody>
</table>

ACCEPT DB-KEY FROM CURRENCY (COBOL)

The ACCEPT DB-KEY FROM CURRENCY statement moves the database key of the current record of run unit, record type, set, or area to a specified location in program variable storage. Records whose database keys are saved in this manner are available for subsequent direct access by using a FIND /OBTAIN DB-KEY statement.

⚠️ **Note:** Establish currency before using this statement. If no currency has been established, the DBMS returns 0000 to the ERROR-STATUS field and -1 to the `db-key` field.

**Currency:** ACCEPT DB-KEY FROM CURRENCY does not update any currencies.

Syntax

```
| ACCEPT db-key-location FROM record-name | set-name | area-name |
```


Parameters

- **ACCEPT db-key-location**
  Specifies where to save the database key of the specified records. This value must be a PIC S9(8) COMP SYNC (fullword) field.

- **FROM ... CURRENCY**
  Specifies which database key to save in the specified location. By default, if you omit a record, set, or area qualifier, the database key of the record that is current of the run unit is saved. The following values are valid:
  - **record-name**
    Saves the database key of the record that is current of the specified record type.
  - **set-name**
    Saves the database key of the record that is current of the specified set.
  - **area-name**
    Saves the database key of the record that is current of the specified area.

- **PAGE-INFO page-info-location**
  Collects the page information for the specified record and records it into the specified location. Specify a PIC S9(8) COMP SYNC (fullword) field.

Example

The following statements establish a DEPARTMENT record as current of run unit and save its database key in location SAVE-DB-KEY:

```
MOVE '8683' TO DEPT-ID-0410.
FIND CALC DEPARTMENT.
ACCEPT SAVE-DB-KEY FROM CURRENCY.
```

**Note:** The same results can be accomplished by using the following COBOL MOVE statement:

```
MOVE DB-KEY TO SAVE-DB-KEY.
```

Status Codes

After completion of the ACCEPT DB-KEY FROM CURRENCY function, the ERROR-STATUS field in the IDMS communications block indicates the outcome:
CA IDMS Reference - 19.0

<table>
<thead>
<tr>
<th>Status code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>The request has been serviced successfully.</td>
</tr>
<tr>
<td>1503</td>
<td>The database key that is the object of an ACCEPT has been invalidated. This outcome can occur only when a run unit is sharing a transaction with other database sessions. The 03 minor status is returned when the run unit tries to retrieve a key and a currency has been invalidated because of changes that were made by another database session that is sharing the same transaction.</td>
</tr>
<tr>
<td>1508</td>
<td>The named record or set is not in the subschema. The program has probably invoked the wrong subschema.</td>
</tr>
<tr>
<td>1523</td>
<td>The named area is not in the subschema.</td>
</tr>
</tbody>
</table>

### ACCEPT DB-KEY RELATIVE TO CURRENCY (COBOL)

The ACCEPT DB-KEY RELATIVE TO CURRENCY statement moves a selected database key to a specified location in program variable storage. You can move the database key of the next, prior, or owner record relative to the current record of set.

This version of the ACCEPT statement lets you save the database key of a record within a set without accessing the record. Records whose database keys are saved in this manner are available for subsequent direct access by using a FIND/OBTAIN DB-KEY statement.

⚠️ **Note:** Establish currency before using this statement. If no set currency has been established, the DBMS returns 0000 to the ERROR-STATUS field and -1 to the `db-key-location` field.

**Currency:** ACCEPT DB-KEY RELATIVE TO CURRENCY does not update any currencies.

### Syntax

```plaintext
ACCEPT db-key-location FROM set-name NEXT PRIOR OWNER CURRENCY
```

### Parameters

- **ACCEPT db-key-location**
  
  Specifies where to save the database key of the requested record. Specify a PIC S9(8) COMP SYNC (fullword) field.
FROM ... CURRENCY
Specifies which record database key to move into the specified location. The following values are valid:

- set-name
  Specifies a set that is included in the subschema.

- NEXT
  Saves the database key of the next record relative to the record that is current of the specified set. NEXT CURRENCY cannot be requested unless the specified set has prior pointers. These pointers ensure that the next pointer in the prefix of the current record does not point to a logically deleted record.

  **Note:** NEXT cannot be requested for sets that are defined for native VSAM records.

- PRIOR
  Saves the database key of the prior record relative to the record that is current of the specified set. PRIOR CURRENCY cannot be requested unless the specified set has prior pointers.

  **Notes:**
  No indication of an end-of-set condition is possible for an ACCEPT NEXT or PRIOR. A retrieval command must be issued to determine whether the next or prior record in the set occurrence is the owner record.
  
  PRIOR cannot be requested for sets that are defined for native VSAM records.

- OWNER
  Saves the database key of the owner of the record that is current of the specified set. OWNER CURRENCY cannot be requested unless the specified set has owner pointers. However, if the current record of the named set is the owner record occurrence, OWNER CURRENCY returns the database key of the record itself, even when this set does not have owner pointers.

  **Notes:**
  When a record that is declared as an optional or manual set member is accessed, it is not established as current of set when it is not currently connected to an occurrence of the specified set. A subsequent attempt to access the owner record locates the owner of the current record of set instead. In such cases, determine whether the retrieved record is actually a set member before executing ACCEPT DB-KEY FROM OWNER CURRENCY. The IF statement (see page 173) can be used for this purpose.
  
  OWNER cannot be requested for sets that are defined for native VSAM records.
• **PAGE-INFO page-info-location**
  Collects the page-info of the specified record and records it into the specified page-info-location. Specify a PIC S9(8) COMP SYNC (fullword) field for this value.

**Example**

The following statements access the EMP-EXPERTISE set and save the database key of the owner record of the SKILL-EXPERTISE set:

MOVE '0119' TO EMP-ID-0415.
FIND CALC EMPLOYEE.
FIND FIRST WITHIN EMP-EXPERTISE.
ACCEPT SAVE-DB-KEY FROM SKILL-EXPERTISE OWNER CURRENCY.

**Status Codes**

After completion of the ACCEPT DB-KEY RELATIVE TO CURRENCY function, the ERROR-STATUS field in the IDMS communications block indicates the outcome:

<table>
<thead>
<tr>
<th>Status code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>The request has been serviced successfully.</td>
</tr>
<tr>
<td>1503</td>
<td>The database key that is the object of an ACCEPT has been invalidated. This outcome occurs only when a run unit is sharing a transaction with other database sessions. The 03 minor status is returned when the run unit tries to retrieve a database key and a currency has been invalidated because of changes that were made by another database session that is sharing the same transaction.</td>
</tr>
<tr>
<td>1508</td>
<td>The named set is not in the subschema. The program has probably invoked the wrong subschema.</td>
</tr>
</tbody>
</table>

**ACCEPT page-info-location (COBOL)**

The ACCEPT page-info-location statement moves the page information for a given record to a specified location in program variable storage. Page information that is saved in this manner is available for subsequent direct access by using a FIND/OBTAIN DB-KEY statement. 

The database key radix portion of the page information can be used when interpreting a database key for display purposes. This portion can also be used when formatting a database key from page and line numbers. The database key radix represents the number of bits within a database key value that are reserved for the line number of a record. By default, this value is 8, meaning that up to 255 records can be stored on a single page of the area. Given a database key, you can determine its associated page number and line number by using the following formulas:

\[
\text{Page-number} = \frac{\text{dbkey value}}{2^{\text{dbkey radix}}} \\
\text{Line-number} = \text{dbkey value} - (\text{page-number} \times (2^{\text{dbkey radix}}))
\]
Syntax

```
ACCEPT page-info-location FOR record-name.
```

Parameters

- **ACCEPT page-info-location**
  Retrieves the specified information. Specify a 4-byte field that is defined as a group field or as a fullword field (PIC S9(8) COMP). This value identifies the location in variable storage that contains page information for the specified record. Upon successful completion of this statement, the first 2 bytes of the field contain the page group number. The last 2 bytes contain a value that can be used to interpret database keys.

- **FOR record-name**
  Specifies the record whose page information to place in the specified location.

Example

The following example retrieves the page information for DEPARTMENT and uses the database key format information to transform a page number into a database key.

```
01 W-PG-INFO.
  02 W-GRP-NUM PIC S9(4) COMP.
  02 W-DBK-FORMAT PIC 9(4) COMP.

ACCEPT W_PG_INFO FOR DEPARTMENT.
MOVE W-PAGE TO W-DBKEY.
PERFORM ADJUST-PAGE W-DBK-FORMAT TIMES.

ADJUST-PAGE SECTION.
  MULTIPY W-DBKEY BY 2.}
```

Status Codes

After completion of the ACCEPT page-info-location function, the ERROR-STATUS field in the IDMS-DC communications block indicates the outcome:

<table>
<thead>
<tr>
<th>Status code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>The request has been serviced successfully.</td>
</tr>
<tr>
<td>1508</td>
<td>The named record is not in the subschema. The program probably invoked the wrong subschema.</td>
</tr>
</tbody>
</table>
ACCEPT PROCEDURE CONTROL LOCATION (COBOL)

The ACCEPT PROCEDURE CONTROL LOCATION statement copies the 256-byte application program information block that was associated with a previously defined database procedure to a specified location in program variable storage. A BIND PROCEDURE statement (see page 107) previously placed information into this block; the procedure may have updated this information subsequently.

The ACCEPT PROCEDURE CONTROL LOCATION statement should be used only by programs running under, but in a different partition from, the central version.

Syntax

```plaintext
ACCEPT procedure-control-location FROM procedure-name PROCEDURE .
```

Parameters

- **ACCEPT procedure-control-location**
  Specifies where to copy the application program information block. Specify a fullword-aligned 256-byte location in variable storage.

- **FROM procedure-name PROCEDURE**
  Specifies which database procedure whose application program information block to copy into variable storage. This value must refer to an eight-character field in variable storage.

Example

The following statement copies the application program information block used by the CHECKALL procedure to the CHECK-IT location in main storage:

```plaintext
ACCEPT CHECK-IT FROM CHECKALL PROCEDURE .
```

Status Codes

After completion of the ACCEPT PROCEDURE CONTROL LOCATION function, the ERROR-STATUS field in the IDMS communications block indicates the outcome:

<table>
<thead>
<tr>
<th>Status code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>The request has been serviced successfully.</td>
</tr>
<tr>
<td>1508</td>
<td>The named procedure is not in the specified subschema.</td>
</tr>
<tr>
<td>1518</td>
<td>The procedure control location has not been bound properly.</td>
</tr>
</tbody>
</table>
ACCEPt TRANSACTION STATISTICS (COBOL)

The ACCEPT TRANSACTION STATISTICS statement copies the contents of the transaction statistics block (TSB) to a location in program variable storage. Optionally, the statement can also write the TSB to the DC system log file. You can define the TSB length.

Syntax

```
ACCEPT TRANSACTION STATISTICS
   WRITE | NOWRITE
   INTO return-stat-data-location
   LENGTH 388
```

Parameters

- **WRITE|NOWRITE**
  Specifies whether to write the TSB to the DC system log file.
  **Default:** WRITE

- **INTO return-stat-data-location**
  Specifies the WORKING-STORAGE SECTION or LINKAGE SECTION data area into which to return the TSB. Specify a fullword-aligned 388-byte field. (You can customize the length by using the LENGTH parameter.)
  The data that is copied from the TSB to `return-stat-data-location` is formatted as follows:

```
01 STATISTICS-BLOCK.
   03 SYS-INTERNAL PIC X(8)    SYSTEM INTERNAL USE ONLY
   03 PROG-CALL   PIC S9(8) COMP # OF PROGRAMS CALLED
   03 PROG-LOAD   PIC S9(8) COMP # OF PROGRAMS LOADED
   03 TERM-READ   PIC S9(8) COMP # OF TERMINAL READS
   03 TERM-WRITE  PIC S9(8) COMP # OF TERMINAL WRITES
   03 TERM-ERROR  PIC S9(8) COMP # OF TERMINAL ERRORS
   03 STORAGE-GET PIC S9(8) COMP # OF STORAGE GETS
   03 SCRATCH-GET PIC S9(8) COMP # OF SCRATCH GETS
   03 SCRATCH-PUT PIC S9(8) COMP # OF SCRATCH PUTS
   03 SCRATCH-DEL PIC S9(8) COMP # OF SCRATCH DELETES
   03 QUEUE-GET   PIC S9(8) COMP # OF QUEUE GETS
   03 QUEUE-PUT   PIC S9(8) COMP # OF QUEUE PUTS
   03 QUEUE-DEL   PIC S9(8) COMP # OF QUEUE DELETES
   03 GET-TIME    PIC S9(8) COMP # OF GET TIMES
   03 SET-TIME    PIC S9(8) COMP # OF SET TIMES
   03 DB-SRVREQ   PIC S9(8) COMP # OF DB SERVICE REQUESTS
   03 MAX-STACK   PIC S9(8) COMP MAX WORDS USED IN STACK
   03 USER-TIME   PIC S9(8) COMP USER MODE TIME (10**-4 SEC)
   03 SYS-TIME    PIC S9(8) COMP SYS MODE TIME (10**-4 SEC)
   03 WALT-TIME   PIC S9(8) COMP WALT TIME (10**-4 SEC)
   03 MAX-RCE-USED PIC S9(8) COMP MAXIMUM NUMBER OF RCES USED
   03 MAX-RLE-USED PIC S9(8) COMP MAXIMUM NUMBER OF RLES USED
   03 MAX-DPE-USED PIC S9(8) COMP MAXIMUM NUMBER OF DPES USED
   03 STG-HI-MARK PIC S9(8) COMP STORAGE HIGH WATER MARK
   03 FREESTG-REQ PIC S9(8) COMP # OF FREE STORAGE REQUESTS
   03 SYS-SERV    PIC S9(8) COMP # OF SYSTEM SERVICE REQUEST
   03 SYS-INTERN2 PIC X(8)   SYSTEM INTERNAL USE ONLY
   03 PAGES-READ  PIC S9(8) COMP # OF PAGES READ
```
If you extend the length to 560 bytes, the full TRANSACTION-STATISTICS are also included. The following block can be expanded using the COPY IDMS TRANSACTION-STATISTICS statement:

```
01 TRANSACTION-STATISTICS.
  03 TSB-STATS-R18    PIC X(560).
  03 TSB-STATS-R17    REDEFINES TSB-STATS-R18.
  04 TSB-DC-STATS     PIC X(108).
  04 TSB-DC-STATS1    REDEFINES TSB-DC-STATS.
  05 SYS-INTERN1      PIC X(8).
  05 PROG-CALL        PIC S9(8) COMP.
  05 PROG-LOAD        PIC S9(8) COMP.
  05 TERM-READ        PIC S9(8) COMP.
  05 TERM-WRITE       PIC S9(8) COMP.
  05 TERM-ERROR       PIC S9(8) COMP.
  05 STORAGE-GET      PIC S9(8) COMP.
  05 SCRATCH-GET      PIC S9(8) COMP.
  05 SCRATCH-PUT      PIC S9(8) COMP.
  05 SCRATCH-DEL      PIC S9(8) COMP.
  05 QUEUE-GET        PIC S9(8) COMP.
  05 QUEUE-PUT        PIC S9(8) COMP.
  05 QUEUE-DEL        PIC S9(8) COMP.
```
05 GET-TIME PIC S9(8) COMP.
05 SET-TIME PIC S9(8) COMP.
05 DB-SRVREQ PIC S9(8) COMP.
05 MAX-STACK PIC S9(8) COMP.
05 USER-TIME PIC S9(8) COMP.
05 SYS-TIME PIC S9(8) COMP.
05 WAIT-TIME PIC S9(8) COMP.
05 MAX-RCE-USED PIC S9(8) COMP.
05 MAX-RLE-USED PIC S9(8) COMP.
05 MAX-DPE-USED PIC S9(8) COMP.
05 STG-HI-MARK PIC S9(8) COMP.
05 FREESTG-REQ PIC S9(8) COMP.
05 SYS-SERV PIC S9(8) COMP.
04 TSB-DB-STATS PIC X(72).
04 TSB-DB-STATS1 REDEFINES TSB-DB-STATS.
  05 SYS-INTERN2 PIC X(8).
  05 PAGES-READ PIC S9(8) COMP.
  05 PAGES-WRIT PIC S9(8) COMP.
  05 PAGES-REQ PIC S9(8) COMP.
  05 CALC-NO PIC S9(8) COMP.
  05 CALC-OF PIC S9(8) COMP.
  05 VIA-NO PIC S9(8) COMP.
  05 VIA-OF PIC S9(8) COMP.
  05 RECS-REQ PIC S9(8) COMP.
  05 RECS-CURR PIC S9(8) COMP.
  05 DB-CALLS PIC S9(8) COMP.
  05 FRAG-STORED PIC S9(8) COMP.
  05 RECS-RELO PIC S9(8) COMP.
  05 TOT-LOCKS PIC S9(8) COMP.
  05 SHR-LOCKS PIC S9(8) COMP.
  05 NSH-LOCKS PIC S9(8) COMP.
  05 LOCKS-FREED PIC S9(8) COMP.
04 TSB-IX-STATS PIC X(40).
04 TSB-IX-STATS1 REDEFINES TSB-IX-STATS.
  05 SR8-SPLITS PIC S9(8) COMP.
  05 SR8-SPAWN PIC S9(8) COMP.
  05 SR8-STORE PIC S9(8) COMP.
  05 SR8-ERASE PIC S9(8) COMP.
  05 SR7-STORE PIC S9(8) COMP.
  05 SR7-ERASE PIC S9(8) COMP.
  05 BTREE-SRCH PIC S9(8) COMP.
  05 BTREE-LEVEL PIC S9(8) COMP.
  05 ORPHANS PIC S9(8) COMP.
  05 BTREE-LEV-B PIC S9(4) COMP.
  05 BTREE-LEV-W PIC S9(4) COMP.
04 TSB-DB-STATS-EXTENDED PIC X(20).
04 TSB-DB-STATS-EXTENDED1 REDEFINES TSB-DB-STATS-EXTENDED.
  05 RECS-UPD PIC S9(8) COMP.
  05 PAGE-INCACHE PIC S9(8) COMP.
  05 PAGE-INPREFET PIC S9(8) COMP.
  05 RESERVED PIC X(8).
04 TSB-HDR PIC X(68).
04 TSB-HDR1 REDEFINES TSB-HDR.
  05 SYS-INTERN3 PIC X(8).
  05 USER-ID PIC X(32).
  05 LTERM-ID PIC X(8).
  05 USER-SUPP-ID PIC X(8).
  05 BIND-DATE PIC S9(7) COMP-3.
  05 BIND-TIME PIC S9(8) COMP.
  05 TRANSTAT-FLGS PIC S9(8) COMP.
04 TSB-SQL-STATS PIC X(80).
04 TSB-SQL-STATS1 REDEFINES TSB-SQL-STATS.
  05 SYS-INTERN4 PIC X(8).
  05 SQL-COMMANDS PIC S9(8) COMP.
  05 SQL-FETCH PIC S9(8) COMP.
  05 SQL-INSERT PIC S9(8) COMP.
  05 SQL-UPDATE PIC S9(8) COMP.
  05 SQL-DELETE PIC S9(8) COMP.
  05 SQL-SORTS PIC S9(8) COMP.
  05 SQL-ROWSORT PIC S9(8) COMP.
05 SQL-MINRSORT PIC S9(8) COMP.
05 SQL-MAXRSORT PIC S9(8) COMP.
05 SQL-AMCMPL PIC S9(8) COMP.
05 SQL-RESERVED PIC X(32).
04 TSB-STATS-DCX PIC X(168).
04 TSB-STATS-DCX1 REDEFINES TSB-STATS-DCX.
05 TSB-STATS-DCX-FILLER PIC X(8).
05 TSB-SYS-MODE-CPU-TOD PIC 9(18) COMP.
05 TSB-SYS-ZIIP-ON-CP-TOD PIC 9(18) COMP.
05 TSB-SYS-ZIIP-ON-ZIIP-TOD PIC 9(18) COMP.
05 TSB-USER-MODE-CPU-TOD PIC 9(18) COMP.
05 TSB-TCB-CPU-TIME-TOD PIC 9(18) COMP.
05 TSB-SRB-CPU-TIME-TOD PIC 9(18) COMP.
05 TSB-STATS-DCX-FILL01 PIC X(112).

**LENGTH**
Specifies the length of the returned TSB. To retrieve all statistics, including the DC extended statistics section that records CPU times in the Time of Day (TOD) format, specify LENGTH as 560.

**len-return-TSB**
Specifies the symbolic name of a user-defined field that contains the TSB length, or the length expressed as a numeric constant.

**Limits:** Integer of 388 or greater

**Default:** If you do not specify *len-return-TSB*, the first 388 bytes of the TSB are returned.

**Example**
The following statement returns the contents of the TSB to STATISTICS-BLOCK and writes the transaction statistics to the log file:

```plaintext
ACCEPT TRANSACTION STATISTICS
WRITE INTO STATISTICS-BLOCK.
```

**Status Codes**

After completion of the ACCEPT TRANSACTION STATISTICS function, the ERROR-STATUS field in the IDMS-DC communications block indicates the outcome:

<table>
<thead>
<tr>
<th>Status code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>The request has been serviced successfully.</td>
</tr>
<tr>
<td>3801</td>
<td>Storage for the transaction statistics block is not available; to wait would cause a deadlock.</td>
</tr>
<tr>
<td>3813</td>
<td>No transaction statistics block exists; a BIND TRANSACTION STATISTICS request has not been issued.</td>
</tr>
<tr>
<td>3831</td>
<td>The parameter list is invalid, or no logical terminal element (LTE) is associated with the issuing task.</td>
</tr>
<tr>
<td>3850</td>
<td>The collection of transaction statistics or task statistics has not been enabled during system generation.</td>
</tr>
</tbody>
</table>
ATTACH (COBOL)

The ATTACH statement initiates a new task by acquiring the necessary control blocks and storage and then adding the task to its dispatching list. CA IDMS initializes the attached task and queues it up for execution. The issuing program receives control in accordance with normal dispatching priority.

Syntax

```
ATTACH TASK CODE 'task-code' PRIORITY priority WAIT | NOWAIT
```

Parameters

- **ATTACH TASK CODE 'task-code'**
  Specifies the symbolic name of a user-defined field containing the task code, or the code itself enclosed in quotation marks. The referenced task code must have been defined during system generation or dynamically with the DCMT VARY DYNAMIC TASK command.

- **PRIORITY priority**
  Specifies the dispatching priority of the attached task. Specify the symbolic name of a user-defined field that contains the dispatching priority, or the priority itself expressed as a numeric constant. The numeric constant must be from 000 to 240. By default, if you do not specify a priority or its location, the priority that was established during system generation for the specified task code, terminal, and user is used.

- **WAIT|NOWAIT**
  Specifies whether the issuing task waits until the maximum task condition no longer exists and the specified task can be attached. When NOWAIT is specified, the program should check the ERROR-STATUS field in the CA IDMS communications block to determine whether the ATTACH request has completed. If the ERROR-STATUS value is 3711, indicating that a maximum task condition exists, the request has not been serviced. The program should perform alternative processing before reissuing the ATTACH request.
  **Default:** WAIT

Example

The following syntax initiates task TASKATCH and assigns it a dispatching priority of 199:

```
ATTACH TASK CODE 'TASKATCH'
PRIORITY 199
NOWAIT.
```
Status Codes

After completion of the ATTACH function, the ERROR-STATUS field of the IDMS-DC communications block indicates the outcome:

<table>
<thead>
<tr>
<th>Status code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>The request has been serviced successfully.</td>
</tr>
<tr>
<td>3711</td>
<td>The task cannot be attached because the maximum number of tasks has already been attached.</td>
</tr>
<tr>
<td>3712</td>
<td>The specified task code is not known to the CA IDMS system.</td>
</tr>
<tr>
<td>3758</td>
<td>The task cannot be attached because the maximum number of concurrent tasks threads was exceeded.</td>
</tr>
<tr>
<td>3799</td>
<td>The requested task could not be attached because the current user is not authorized to execute the task.</td>
</tr>
</tbody>
</table>

**BIND MAP (COBOL)**

The BIND MAP statement identifies the location of a map request block (MRB) and initializes the MRB fields. For each MRB that is used by a program, code a BIND MAP statement. For each record that is defined to a map, code a BIND MAP RECORD statement.

BIND MAP statements can be global or record-specific:

- A *global* BIND MAP statement applies to the map as a whole. The statement initializes the entire MRB and fills in the fields that apply to the map in general.

- A *record-specific* BIND MAP statement applies only to the named map record. The statement initializes the variable storage address of the named record in the MRB.

Typically, a program issues a global BIND MAP statement for each map, followed by BIND MAP statements for each map record that is used by the program.

You can include global and record-specific BIND MAP statements automatically by using a COPY IDMS MAP-BINDS statement (see Precompiler-Directive Statements (see page 63)). This statement includes the necessary BINDS for all maps and map records that are defined for the program.

The program can alter the storage address for a map record at any time by issuing another BIND MAP statement for that record. After the initial global bind (BIND MAP), all map records are considered unbound. Map operations that use those records have no effect on storage. After a map record is bound to a storage address (BIND MAP RECORD), subsequent map operations use that address to access the record. To unbind a map record, issue a record-specific BIND MAP statement that specifies the TO NULL option.
Syntax

```
BIND MAP map-name
  RECORD rec-name
  TO NULL
  rec-location
```

Parameters

- **BIND MAP map-name**
  Specifies an existing map. The map version defaults to the version that is specified for the map in the MAP SECTION of the program.

- **RECORD rec-name**
  Specifies a record that is used by the map. This value initializes the variable storage address of the named record in the MRB.

- **TO**
  Specifies whether to bind the record to a specified address. The following values are valid:
  - **NULL**
    Leaves the record unbound.
  - **rec-location**
    Binds the record to the specified address. Specify the symbolic name of a user-defined field that contains the address. Rec-location defaults to rec-name. Subsequent I/O operations use this area of storage for any operation that is associated with the record.

Example

The following statements bind the EMPMAPLR map and its five associated map records:

```
BIND MAP EMPMAPLR.
BIND MAP EMPMAPLR RECORD EMPLOYEE.
BIND MAP EMPMAPLR RECORD DEPARTMENT.
BIND MAP EMPMAPLR RECORD JOB.
BIND MAP EMPMAPLR RECORD OFFICE.
BIND MAP EMPMAPLR RECORD EMP-DATE-WORK-REC.
```

Status Codes

After completion of the BIND MAP function, the ERROR-STATUS field in the IDMS-DC communications block indicates the outcome:

<table>
<thead>
<tr>
<th>Status code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>The request has been serviced successfully.</td>
</tr>
</tbody>
</table>
**BIND PROCEDURE (COBOL)**

The BIND PROCEDURE statement establishes communication between a program and a DBA-written database procedure (for example, a security routine). Use this statement only when the application program must pass more information to the procedure than the DBMS provides. Such instances are unusual. Usually, you will not be aware of which procedures gain control before or after various DML functions.

The BIND PROCEDURE statement is used in both the navigational and the non-navigational environments.

**Syntax**

```
BIND PROCEDURE FOR procedure-name TO procedure-control-location .
```

**Parameters**

- **BIND PROCEDURE FOR procedure-name**
  Specifies which database procedure in program variable storage to make available to the program.

- **TO procedure-control-location**
  Specifies where to bind the named procedure. Specify a 256-byte (fixed-length) location. A program that runs in a different partition from the central version may need to pass certain information to the database procedure. When the DBMS invokes the database procedure, this information is copied from the `procedure-control-location` to the IDMS application program information block. The program passes the information that was in `procedure-control-location` during the BIND PROCEDURE, not the information that was present at the time of the procedure call.

**Example**

The following statement binds the procedure with the variable name PROGCHEK to the 256-byte area PROC-CTL:

```
BIND PROCEDURE FOR PROGCHEK TO PROC-CTL.
```

**Status Codes**

After completion of the BIND PROCEDURE function, the ERROR-STATUS field in the IDMS communications block indicates the outcome:
## Status code

<table>
<thead>
<tr>
<th>Status code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>The request has been serviced successfully.</td>
</tr>
<tr>
<td>1400</td>
<td>The BIND PROCEDURE statement cannot be recognized. This code usually indicates that the IDMS communications block (SUBSCHEMA-CTRL) is not aligned on a fullword boundary.</td>
</tr>
<tr>
<td>1408</td>
<td>The named procedure is not in the specified subschema.</td>
</tr>
<tr>
<td>1418</td>
<td>The procedure has been bound improperly to location 0.</td>
</tr>
<tr>
<td>1472</td>
<td>The available memory is insufficient to load the database procedure dynamically.</td>
</tr>
<tr>
<td>1474</td>
<td>An attempt to load a module from the load/core-image library or DDLDCLOD has failed.</td>
</tr>
</tbody>
</table>

## BIND RECORD (COBOL)

The BIND RECORD statement establishes addressability for a record in program variable storage. Usually, you need not issue individual BIND RECORD statements because the necessary statements are generated as a group by the COPY IDMS SUBSCHEMA-BINDS statement (see Precompiler-Directive Statements (see page 63)). However, you can issue BIND RECORD commands separately as necessary. (For example, to bind several records to the same storage location.) In any case, addressability must be established for each subschema record to be used by the program.

The program should perform the IDMS-STATUS routine after each BIND RECORD statement to ensure that the statement was executed successfully. When AUTOSTATUS is in use (see AUTOSTATUS Protocols (see page)), a PERFORM IDMS-STATUS operation occurs automatically after each BIND RECORD statement. This operation occurs even when the BIND RECORD statements are generated as a group by a COPY IDMS SUBSCHEMA-BINDS statement. Use COPY IDMS SUBSCHEMA-BINDS only when AUTOSTATUS is in use.

### Syntax

```
BIND record-name TO record-location.
record-location WITH record-name.
```

### Parameters

- **BIND record-name**
  
  Specifies the record to bind to a location in variable storage. The specified record must be included in the subschema.

- **TO record-location**
  
  Specifies where to bind the record. The location corresponds to the record description that was copied into the program manually or automatically through DATA DIVISION statements. The `record-location` must be the same length as `record-name`. 

Note: Exercise caution when using the TO record-location option. A source-object mismapping can result from improper use. When more than one copy of a given database record description is present in the program, ensure that the proper record description is bound at the proper time.

- record-location WITH record-name
  Binds a record name literal (record-name) with a variable storage record description (record-location). Record-name must specify a record that is included in the subschema.

Example

The following statement binds the EMPLOYEE record:

BIND EMPLOYEE.

Status Codes

After completion of the BIND RECORD function, the ERROR-STATUS field in the IDMS communications block indicates the outcome:

<table>
<thead>
<tr>
<th>Status code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>The request has been serviced successfully.</td>
</tr>
<tr>
<td>1400</td>
<td>The BIND RECORD statement cannot be recognized. This code usually indicates that the IDMS communications block (SUBSCHEMA-CTRL) is not aligned on a fullword boundary.</td>
</tr>
<tr>
<td>1408</td>
<td>The named record is not in the subschema. The program has probably invoked the wrong subschema.</td>
</tr>
<tr>
<td>1418</td>
<td>The record has been bound improperly to location 0.</td>
</tr>
<tr>
<td>1472</td>
<td>The available memory is insufficient to load a database procedure dynamically.</td>
</tr>
<tr>
<td>1474</td>
<td>An attempt to load a module from the load/core-image library or DDLDCLOD has failed.</td>
</tr>
</tbody>
</table>

BIND RUN-UNIT (COBOL)

The BIND RUN-UNIT statement establishes a run unit for accessing the database, identifies the location of the IDMS communications block being used, and names the subschema to be loaded for the run unit. BIND RUN-UNIT can also name the node under which the run unit executes and identify the database to access. BIND RUN-UNIT must be the first functional DML call passed to the DBMS at execution time. This statement must logically precede all other DML statements (for example, BIND RECORD, READY, FIND) in the PROCEDURE DIVISION of the program.

UTM modes only: Before you issue the BIND RUN-UNIT statement, move LOW VALUES to SUBSCHEMA-CTRL.
When AUTOSTATUS is in use, COPY IDMS SUBSCHEMA BINDS can automatically invoke the BIND RUN-UNIT statement and the appropriate BIND RECORD statements (see Precompiler-Directive Statements (see page 63)).

Program registration requires that all programs be registered in the data dictionary before compilation. If program registration is in effect, the program must initialize the PROGRAM-NAME field of the IDMS communications block automatically or manually:

- **Automatic initialization:** COPY IDMS SUBSCHEMA-BINDS automatically generates a COBOL MOVE statement that moves the program name (stated in the IDENTIFICATION DIVISION) to the PROGRAM-NAME field.

- **Manual initialization:** The programmer manually codes a COBOL MOVE statement before the BIND RUN-UNIT statement is executed. See the following example:

  ```cobol
  MOVE 'EMPDISP' TO PROGRAM-NAME.
  ```

The BIND RUN-UNIT statement is used in both the navigational and the non-navigational environments.

### Syntax

```
BIND RUN-UNIT [FOR subschema-name]
```

```
[DBNODE nodename] [DBNAME database-name]
```

```
[DICTNODE nodename] [DICTNAME dictionary-name]
```

### Parameters

- **FOR subschema-name**
  Identifies a subschema view other than the one that was specified in the DB clause of the SCHEMA SECTION. This value must be the symbolic name of a user-defined eight-character field in variable storage.
  By default, if you do not specify a subschema, the run unit uses the subschema that is named in the DB clause of the SCHEMA SECTION.

  **Note:** Exercise care when using the FOR subschema-name option. Improper use can lead to mismappings between the named subschema and record descriptions in variable storage.

- **DBNODE nodename**
  Specifies the node where the database resides. Specify the symbolic name of a user-defined eight-character field in variable storage, or the database name itself enclosed in quotation marks.
- **DBNAME database-name**
  Specifies which database to access. Specify the symbolic name of a user-defined eight-character field in variable storage, or the database name itself enclosed in quotation marks.

- **DICTNODE nodename**
  Specifies the node that controls the dictionary where the subschema resides. Specify the symbolic name of a user-defined eight-character field in variable storage, or the node name itself enclosed in quotation marks.

- **DICTNAME dictionary-name**
  Specifies the dictionary where the subschema resides. Specify the symbolic name of a user-defined eight-character field in variable storage, or the dictionary name itself enclosed in quotation marks.

**Notes:**
- The DBNODE, DBNAME, DICTNODE, and DICTNAME parameters can be overridden at runtime by the DCUF SET DBNODE/DBNAME and DCUF SET DICTNODE/DICTNAME commands.
- **Batch users:** The DBNODE AND DBNAME parameters can be overridden at runtime when the IDMSOPTI module or the SYSCTL clause in the system generation SYSTEM statement specifies a nodename or a dbname with the ALWAYS option. For more information, see [SYSTEM Statement Parameters](https://docops.ca.com/display/IDMS19/SYSTEM+Statement+Parameters+Information).

**Example**

The following statement binds the run unit to the DBMS:

BIND RUN-UNIT.

**Status Codes**

After completion of the BIND RUN-UNIT function, the ERROR-STATUS field in the IDMS communications block indicates the outcome:

<table>
<thead>
<tr>
<th>Status code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>The request has been serviced successfully.</td>
</tr>
<tr>
<td>1400</td>
<td>The BIND RUN-UNIT statement cannot be recognized. This code usually indicates that the IDMS communications block (SUBSCHEMA-CTRL) is not aligned on a fullword boundary.</td>
</tr>
<tr>
<td>1410</td>
<td></td>
</tr>
</tbody>
</table>
A security violation has been detected. An existing access restriction or subschema usage prohibits execution of the specified DML function. For LRF users, the subschema in use allows access to database records only. When this code is combined with a major code of 00, the program has attempted to access a database record, but the subschema in use allows access to logical records only.

1417 The transaction manager encountered an error. See the log for additional information.

1467 The subschema load module is invalid. The invoked subschema does not match the subschema object tables.

1469 The run unit is not bound to the DBMS. The central version is not active, the central version is not accepting new run units, or the run unit’s connection to the central version is broken due to timeout or other factors, as noted on the CV log.

1470 A journal file will not open (local mode only). Under OS, the most probable cause is that a DD statement for the journal file is missing in the JCL.

1472 There is insufficient memory to load a subschema or database procedure dynamically.

1473 The central version is not accepting new run units.

1474 The subschema was not found in the dictionary load area or in the load library.

1477 The run unit has been bound previously.

1480 The specified node is not active or has been disabled.

1481 The converted subschema requires the specified database name to be in the DBNAME table.

1482 The subschema must be named in the DBNAME table.

1483 The available memory is insufficient to allocate native VSAM work areas.

1491 The subschema requires a DBNAME to do the bind run unit.

1492 No subschema areas map to DMCL.

1493 A subschema area symbolic was not found in DMCL.

1494 The specified dbname is not a dbname defined in the DBNAME table, or a SEGMENT defined in the DMCL.

1495 The specified subschema failed DBTABLE mapping using the specified dbname.

### BIND TASK (COBOL)

The BIND TASK statement initiates a CA IDMS task when the operating mode is DC-BATCH. This statement establishes communication with the DC system and, when accessing CA IDMS queues, allocates a packet-data movement buffer to contain the queue data. After a task is started, the program can issue any number of consecutive BIND-READY-FINISH sequences.

⚠️ **Note:** Do not issue this command in local mode.
Syntax

```plaintext
BIND TASK
  NODENAME (nodename)
```

Parameters

- **NODENAME (nodename)**
  Specifies which node to bind the task to. Specify the symbolic name of a user-defined field that contains the nodename, or the nodename itself enclosed in quotation marks. The node name must match the node named in the DDS statement at system generation.

Example

The following statement establishes communication with a DC system:

```plaintext
BIND TASK.
```

Status Codes

After completion of the BIND TASK function, the ERROR-STATUS field in the IDMS-DC communications block indicates the outcome:

<table>
<thead>
<tr>
<th>Status code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>The request has been serviced successfully.</td>
</tr>
</tbody>
</table>

**BIND TRANSACTION STATISTICS (COBOL)**

The BIND TRANSACTION STATISTICS statement defines the beginning of a transaction for the purposes of collecting transaction statistics. CA IDMS allocates a block of storage to accumulate these statistics. Because this block is owned by the logical terminal that is associated with the current task, the BIND TRANSACTION STATISTICS statement cannot be used with nonterminal tasks.

⚠️ **Note:** If a transaction statistics block (TSB) is already allocated for the logical terminal that is associated with the current task, the BIND request clears the block and writes any previously accumulated transaction statistics to the log file.

When a BIND TRANSACTION STATISTICS request is issued, the transaction is assigned a 40-character identifier. The first 32 characters are the identifier of the signed-on user (if any). The last eight characters are the identifier of the logical terminal that is associated with the current task.
Syntax

BIND TRANSACTION STATISTICS.

Example

The following example illustrates the BIND TRANSACTION STATISTICS statement:

BIND TRANSACTION STATISTICS.

Status Codes

After completion of the BIND TRANSACTION STATISTICS function, the ERROR-STATUS field in the IDMS-DC communications block indicates the outcome:

<table>
<thead>
<tr>
<th>Status code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>The request has been serviced successfully. Any existing transaction statistics block was written to the log file before being cleared.</td>
</tr>
<tr>
<td>3801</td>
<td>Storage for the transaction statistics block is not available; to wait would cause a deadlock.</td>
</tr>
<tr>
<td>3810</td>
<td>A new transaction statistics block has been allocated.</td>
</tr>
<tr>
<td>3831</td>
<td>The parameter list is invalid or no logical terminal element (LTE) is associated with the issuing task.</td>
</tr>
<tr>
<td>3850</td>
<td>The collection of transaction statistics or task statistics has not been enabled during system generation.</td>
</tr>
</tbody>
</table>

CHANGE PRIORITY (COBOL)

The CHANGE PRIORITY statement changes the dispatching priority of the issuing task. The new dispatching priority applies only to the current execution of the task. CHANGE PRIORITY does not relinquish control to another task and cannot be used to alter the priority of other tasks.

Syntax

CHANGE PRIORITY to \texttt{priority}.

Parameters

- CHANGE PRIORITY to \texttt{priority}
  
  Specifies the new dispatching priority for the issuing task. Specify the symbolic name of a user-defined field that contains the priority value, or the value itself expressed as a numeric constant in the range 0 through 240.
Example

The following example changes the dispatching priority of the issuing task to the value contained in the PRIORITY-210 field:

```plaintext
CHANGE PRIORITY TO PRIORITY-210.
```

Status Codes

After completion of the CHANGE PRIORITY function, the ERROR-STATUS field in the IDMS-DC communications block indicates the outcome:

<table>
<thead>
<tr>
<th>Status code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>The request has been serviced successfully.</td>
</tr>
</tbody>
</table>

CHECK TERMINAL (COBOL)

The CHECK TERMINAL statement delays task processing until a previously issued I/O request has completed.

If a READ TERMINAL, WRITE TERMINAL, or WRITE THEN READ TERMINAL request specifies the NOWAIT option, the program must issue a CHECK TERMINAL request before specifying any other I/O operation. If the I/O operation is not complete, the task execution is suspended. When the I/O operation is complete, the task resumes execution according to its established dispatching priority.

Syntax

```plaintext
CHECK TERMINAL
▶ GET STORAGE
▶ INTO input-data-location
   TO end-input-data-location
   MAX LENGTH input-data-max-length
▶ RETURN LENGTH into input-data-actual-length.
```

Parameters

- **GET STORAGE**
  Acquires an input buffer for the data being read into the program. CA IDMS allocates the required storage when the read operation is complete. This parameter applies only to asynchronous requests.

- **INTO input-data-location**
  Specifies the data area that is reserved for the input data stream. Specify the symbolic name of a user-defined field. If GET STORAGE is specified, the data area must be an unallocated 01-level LINKAGE SECTION entry. If GET STORAGE is not specified, the data area must be a previously allocated WORKING-STORAGE SECTION or LINKAGE SECTION entry.
- **TO end-input-data-location**
  Specifies the end of the data area that is reserved for the input. Specify the symbolic name of a user-defined dummy byte field. Or, specify a field that contains a data item which is not associated with the data area reserved for the input data stream.

- **MAX LENGTH input-data-max-length**
  Defines the length, in bytes, of the data area that is reserved for the input data stream. Specify the symbolic name of a user-defined field that contains the length of the data area, or the length itself expressed as a numeric constant. If the input data stream is larger than the data area that is reserved in the WORKING-STORAGE SECTION or LINKAGE SECTION, the data stream is truncated to fit the available space.

- **RETURN LENGTH INTO input-data-actual-length**
  Specifies the location to return the actual length of the input data stream. Specify the symbolic name of a user-defined field. If the data stream has been truncated, `input-data-actual-length` contains the original length before truncation.

### Example

The following statement determines whether an I/O operation is complete, acquires an input buffer, and reads 72 bytes of data into TERM-LINE:

```plaintext
CHECK TERMINAL
GET STORAGE
INTO TERM-LINE MAX LENGTH 72.
```

### Status Codes

After completion of the CHECK TERMINAL function, the ERROR-STATUS field in the IDMS-DC communications block indicates the outcome:

<table>
<thead>
<tr>
<th>Status code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>The request has been serviced successfully.</td>
</tr>
<tr>
<td>4519</td>
<td>The input area that was specified for the return of data is too small. The returned data has been truncated to fit the available space.</td>
</tr>
<tr>
<td>4525</td>
<td>The output operation has been interrupted. The terminal operator has pressed ATTENTION or BREAK.</td>
</tr>
<tr>
<td>4526</td>
<td>A logical error (for example, an invalid control character) has been encountered in the output data stream.</td>
</tr>
<tr>
<td>4527</td>
<td>A permanent I/O error has occurred during processing.</td>
</tr>
<tr>
<td>4528</td>
<td>The dial-up line for the terminal being used has been disconnected.</td>
</tr>
<tr>
<td>4531</td>
<td>The terminal request block (TRB) contains an invalid field, indicating a possible error in the program parameters.</td>
</tr>
<tr>
<td>4535</td>
<td>Storage for the input buffer cannot be acquired because the specified 01-level LINKAGE SECTION entry has been allocated.</td>
</tr>
<tr>
<td>Status code</td>
<td>Meaning</td>
</tr>
<tr>
<td>-------------</td>
<td>-------------------------------------------------------------------------</td>
</tr>
<tr>
<td>4537</td>
<td>Storage for the input buffer cannot be acquired. The specified data area is defined in the WORKING-STORAGE SECTION instead of the LINKAGE SECTION.</td>
</tr>
<tr>
<td>4538</td>
<td>The specified 01-level LINKAGE SECTION entry has not been allocated and the GET STORAGE option has not been specified. No I/O has been performed.</td>
</tr>
<tr>
<td>4539</td>
<td>The terminal device that is associated with the issuing task is out of service.</td>
</tr>
</tbody>
</table>

**COMMIT (COBOL)**

The COMMIT statement commits changes that were made to the database through an individual run unit or through all database sessions associated with a task. A task-level commit also commits all changes that were made with scratch, queue, and print activity.

If the commit applies to an individual run unit and the run unit is sharing its transaction with another database session, the run unit changes may not be committed when the COMMIT statement is executed.

⚠️ **Note:** For more information about the impact of transaction sharing, see [Sharing Transactions Among Sessions](https://docops.ca.com/display/IDMSCU19/Sharing+Transactions+Among+Sessions).

Run units (and SQL sessions) impacted by the COMMIT statement remain active after the operation is complete.

The COMMIT statement is used in both the navigational and logical record facility environments. The COMMIT TASK statement is also used in an SQL programming environment.

**Currency:** Use of the ALL option, as in COMMIT ALL, sets all currencies to null.

**Syntax**

```
COMMIT [ TASK | ALL ] .
```

**Parameters**

- **TASK**
  Commits the changes that were made by all scratch, queue, and print activity and all top-level run units that are associated with the current task. The impact on SQL sessions that are associated with the task depends on whether those sessions are suspended and whether their transactions are eligible to be shared.

⚠️
Notes:

- For more information about the impact of a COMMIT TASK statement on SQL sessions, see Effect of Teleprocessing Statement and Events (https://docops.ca.com/display/IDMS19/Effect+of+Teleprocessing+Statements+and+Events).
- For more information about run units and the impact of COMMIT TASK, see Sharing Transactions Among Sessions (https://docops.ca.com/display/IDMSCU19/Sharing+Transactions+Among+Sessions).

- **ALL**
  Releases all currency locks on records in database, scratch, and queue areas that are associated with the issuing task (COMMIT TASK ALL) or run unit (COMMIT ALL). ALL also sets all currencies to null.

Example

The following statement commits changes that were made by the run unit through which it is issued:

```plaintext
COMMIT.
```

Status Codes

After completion of the COMMIT function, the ERROR-STATUS field in the IDMS communications block indicates the outcome:

<table>
<thead>
<tr>
<th>Status code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>The request has been serviced successfully.</td>
</tr>
<tr>
<td>5031</td>
<td>The specified request is invalid. The program may contain a logic error.</td>
</tr>
<tr>
<td>5097</td>
<td>An error was encountered while processing a syncpoint request. See the log for details.</td>
</tr>
</tbody>
</table>

CONNECT (COBOL)

The CONNECT statement establishes a record occurrence as a member of a set occurrence. The specified record must be defined as an optional automatic, optional manual, or mandatory manual member of the set.

**Native VSAM users:** The CONNECT statement is not valid because all sets in native VSAM data sets must be defined as mandatory automatic.

Before the CONNECT statement executes, the following conditions must be satisfied:

- All areas that are affected either explicitly or implicitly by the CONNECT statement must be readied in one of the update usage modes (see READY (COBOL) (see page 228)).
The specified record must be established as current of its record type.

The occurrence of the set into which the specified record will be connected must be established. The current record of set determines the set occurrence and, when set order is NEXT or PRIOR, the position at which the specified record will be connected within the set.

**Currency:** After a CONNECT statement executes successfully, the specified record is current of run unit, its record type, its area, and all sets in which it currently participates.

**Syntax**

```
CONNECT record-name TO set-name.
```

**Parameters**

- **CONNECT record-name**
  Specifies the record whose current occurrence is to be connected to the current occurrence of the specified set. This record must be included in the subschema and must be defined as an optional automatic, optional manual, or mandatory manual member of the set to which it is being connected.

- **TO set-name**
  Specifies the set to which the member record is to be connected. The set must be included in the subschema. The record is connected to the set in accordance with the ordering rules that are defined for that set in the schema.

**Example**

The following figure illustrates the steps that are required to connect an EMPLOYEE record to an occurrence of the OFFICE-EMPLOYEE set.

To connect EMPLOYEE 459 to OFFICE 1 in the OFFICE-EMPLOYEE set, establish EMPLOYEE 459 as current of record type, locate the proper occurrence of the OFFICE record, and issue the CONNECT command.

![Diagram illustrating the steps to connect EMPLOYEE 459 to OFFICE 1 in the OFFICE-EMPLOYEE set.](image-url)
After completion of the CONNECT function, the ERROR-STATUS field in the IDMS communications block indicates the outcome:

<table>
<thead>
<tr>
<th>Status code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>The request has been serviced successfully.</td>
</tr>
<tr>
<td>0705</td>
<td>The CONNECT would violate a duplicates-not-allowed option.</td>
</tr>
<tr>
<td>0706</td>
<td>Currency has not been established for the named record or set.</td>
</tr>
<tr>
<td>0708</td>
<td>The named record is not in the subschema. The program has probably invoked the wrong subschema.</td>
</tr>
<tr>
<td>0709</td>
<td>The area of the named record has not been readied in one of the update usage modes.</td>
</tr>
<tr>
<td>0710</td>
<td>The subschema specifies an access restriction that prohibits connecting the named record in the named set.</td>
</tr>
<tr>
<td>0714</td>
<td>The CONNECT statement cannot be executed. The named record has been defined as a mandatory automatic member of the set.</td>
</tr>
<tr>
<td>0716</td>
<td>The record cannot be connected to a set in which it is already a member.</td>
</tr>
<tr>
<td>0721</td>
<td>An area other than the area of the named record has been readied with an incorrect usage mode.</td>
</tr>
<tr>
<td>0725</td>
<td>Currency has not been established for the named set type.</td>
</tr>
</tbody>
</table>
DC RETURN (COBOL)

The DC RETURN statement returns control to a program at the next higher level within a task. (If the issuing program is the highest level program, control returns to CA IDMS.) You can also use the DC RETURN statement to specify the following items:

- The next task to initiate on the same terminal
- Recovery procedures for abend routines that are established by SET ABEND EXIT functions
- The action to take when the user fails to initiate the next task

DC RETURN Processing

The following figure illustrates how a task is executed when DC RETURN statements within three programs specify the NEXT TASK CODE option.

This figure shows the following processing flow:

1. Task A invokes program A.
2. Program A links to program B, which links to program C.
3. Program C issues a DC RETURN NEXT TASK CODE 'Z' request, which returns control to program B.
4. Program B contains a DC RETURN NEXT TASK CODE 'Y' request, which takes precedence over the program C DC RETURN specification.
5. Control returns to program A, which issues a DC RETURN NEXT TASK CODE 'X' request.
6. Because program A is at the highest level in the task, task X is invoked.
When CA IDMS receives control from the highest level program that issued a DC RETURN NEXT TASK CODE request, the task code attribute controls when the specified task is executed:

- If the task code was assigned the NOINPUT attribute during system generation, the task executes immediately.
- If the task code was assigned the INPUT attribute, the task executes only when the user presses an attention identifier (AID) key. Typical AID keys include all PA and PF keys, ENTER, and CLEAR.

**Syntax**

```
DC RETURN
  NEXT TASK CODE next-task-code

  NORMAL
  ABORT
  CONTINUE
  IMMEDIATE

  TIMEOUT
    INTERVAL timeout-interval
    PROGRAM timeout-program

  NEXT TASK INTERVAL start-interval EVENT TYPE
    INTERNAL
    EXTERNAL

  EVENT ecb
  EVENT NAME ecb-id
```

**Parameters**

- **NEXT TASK CODE next-task-code**
  Specifies the next task to initiate on the same terminal. Specify the symbolic name of a user-defined field that contains the task code, or the task code itself enclosed in quotation marks. The task code must be defined to the DC system under which it is running, either during system generation or at runtime.

  **Note:** The position of the issuing program within the task governs whether the specified task actually receives control.
• **NORMAL|ABORT|CONTINUE|IMMEDIATE**
  Specifies the recovery action to take within the program logic (CA IDMS recovery occurs automatically), and whether to execute abend routines for higher-level programs. These options apply only to DC RETURN statements from abend routines that are established by SET ABEND EXIT (STAE) functions.

  - **NORMAL**
    Specifies to not attempt recovery and execute all abend routines established for programs at higher task levels. This value is the default.

  - **ABORT**
    Specifies to not attempt recovery and abort the task immediately without executing any abend routines established for programs at higher task levels.

  - **CONTINUE**
    Specifies to return control to the program that failed at an address established in the abend control element (ACE) for the program.

  - **IMMEDIATE**
    Is ignored when issued from ABEND routine. This option is only applied when it is not issued from an ABEND routine.

• **TIMEOUT**
  Specifies the action to take when the user fails to enter the data that is required to initiate a task. This parameter overrides the resource timeout interval and program specifications that were established during system generation.

• **INTERVAL timeout-interval**
  Specifies the time, in seconds, that can elapse before the resources that are held by the terminal on which the task is executing are released. Specify the symbolic name of a user-defined PIC S9(4) COMP SYNC (halfword) field that contains the timeout interval, or the interval itself expressed as a numeric constant.

• **PROGRAM timeout-program**
  Specifies which program to invoke to handle and release resources that are held by the terminal on which the task is executing when the specified timeout interval has been reached. Specify the symbolic name of a user-defined field that contains the program name, or the name itself enclosed in quotation marks. The program must be defined to the DC system either during system generation or at runtime.

• **NEXT TASK INTERVAL start-interval**
  Specify the symbolic name of a user-defined PIC S9(4) COMP SYNC (halfword) field that contains the start interval, or the interval itself expressed as a numeric constant.

---

**Note:** When NEXT TASK INTERVAL is specified alone, it causes the task to be initiated after *start-interval*. When NEXT TASK INTERVAL is specified with EVENT/EVENT NAME, the task is initiated after *start-interval* or after posting of the events or event names, whichever occurs first.
EVENT TYPE INTERNAL | EXTERNAL
Specifies events that happen either internal or external to the system.

- INTERNAL
  An event that occurs within IDMS-DC, such as waiting for space in a storage pool, or waiting for a completed task.

- EXTERNAL
  An event that occurs outside the control of the system, such as waiting for a file to be read, or waiting for an I/O to complete.

EVENT ecb
Specifies one or more ECBs upon which the task waits. Specify the symbolic name of a user-defined area that contains three PIC 9(8) COMP SYNC (fullword) fields. If you specify multiple EVENT parameters, separate them with at least one blank.

EVENT NAME ecb-id
Specifies the ECB upon which the task waits. Specify the symbolic name of a user-defined field that contains the ECB ID, or the ID itself enclosed in quotation marks.

⚠️ Note: When NEXT TASK INTERVAL is specified alone, it causes the task to be initiated after start-interval. When NEXT TASK INTERVAL is specified with EVENT/EVENT NAME, the task is initiated after start-interval or after posting of the events or event names, whichever occurs first.

Example
The following statement illustrates the use of DC RETURN. If the task code that is associated with MENU-TASK-CODE was defined with the INPUT attribute, it is invoked when the user presses an AID key. If the task code was defined with the NOINPUT attribute, it is invoked immediately:

```
DC RETURN
  NEXT TASK CODE MENU-TASK-CODE.
```

Status Codes
Because control is returned to the next-higher level, the ERROR-STATUS field is not checked.

DELETE QUEUE (COBOL)
The DELETE QUEUE statement deletes all or part of a queue. If only one queue record is deleted, CA IDMS maintains queue currency by saving the next and prior currencies of the deleted record.
Syntax

```
DELETE QUEUE ID queue-id CURRENT ALL
```

Parameters

- **ID queue-id**
  Specifies which queue contains the record to delete. Specify the symbolic name of a user-defined field that contains the ID, or the ID itself enclosed in quotation marks. If you do not specify an ID, a blank ID is assumed.

- **CURRENT**
  Deletes the current record of the queue that is associated with the requesting task. This value is the default.

- **ALL**
  Deletes all records in the queue and the queue header ID.

Example

The following example deletes the current record in the RES-Q queue:

```
DELETE QUEUE
  ID 'RES-Q'
  CURRENT.
```

Status Codes

After completion of the DELETE QUEUE function, the ERROR-STATUS field in the IDMS-DC communications block indicates the outcome:

<table>
<thead>
<tr>
<th>Status code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>The request has been serviced successfully.</td>
</tr>
<tr>
<td>4404</td>
<td>The requested queue header record cannot be found.</td>
</tr>
<tr>
<td>4405</td>
<td>The requested queue record cannot be found.</td>
</tr>
<tr>
<td>4406</td>
<td>The queue record does not have a resource control element (RCE), which indicates that currency has not been established.</td>
</tr>
<tr>
<td>4407</td>
<td>A database error occurred during the queue processing. A common cause is a DBKEY deadlock. For a PUT QUEUE operation, this code can also mean that the queue upper limit has been reached.</td>
</tr>
</tbody>
</table>
If a database error has occurred, the CA-IDMS/DC/UCF log usually has other messages indicating that a problem was encountered in RHDCRUAL, the internal Run Unit Manager. If a deadlock has occurred, messages DC001000 and DC001002 are also produced.

4431  The parameter list is invalid.

### DELETE SCRATCH (COBOL)

The DELETE SCRATCH statement deletes one scratch record or all records in the scratch area.

#### Syntax

```cobol
DELETE SCRATCH
  AREA ID scratch-area-id
  CURRENT
  FIRST
  LAST
  NEXT
  PRIOR
  ALL
  RECORD ID scratch-record-id
  RETURN RECORD ID into return-scratch-record
```

#### Parameters

- **AREA ID scratch-area-id**
  Specifies the scratch area that is associated with the scratch records being deleted. Specify the symbolic name of a user-defined field that contains the scratch area ID, or the ID itself enclosed in quotation marks. If you do not specify an AREA ID, an area ID of eight blanks is assumed.

- **CURRENT**
  Deletes the current record in the scratch area (the record that was most recently referenced by another scratch function). This value is the default.

- **FIRST**
  Deletes the first record in the specified scratch area.

- **LAST**
  Deletes the last record in the specified scratch area.

- **NEXT**
  Deletes the next record in the specified scratch area.

- **PRIOR**
  Deletes the prior record in the specified scratch area.
• **ALL**
  Deletes all records in the specified scratch area.

• **RECORD ID scratch-record-id**
  Deletes the specified record. Specify the symbolic name of a user-defined field that contains the ID.

• **RETURN RECORD ID into return-scratch-record**
  Specifies the location in the program in which to return the ID of the last record that was deleted by the DELETE SCRATCH function. Specify the symbolic name of a user-defined 4-byte field.

**Example**

The following example deletes the scratch record that is prior to the current scratch record and returns the ID of the deleted record to the SCR-REC-ID field:

```
DELETE SCRATCH
PRIOR
RETURN RECORD ID INTO SCR-REC-ID.
```

**Status Codes**

After completion of the DELETE SCRATCH function, the ERROR-STATUS field in the IDMS-DC communications block indicates the outcome:

<table>
<thead>
<tr>
<th>Status code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>The request has been serviced successfully.</td>
</tr>
<tr>
<td>4303</td>
<td>The requested scratch area ID cannot be found.</td>
</tr>
<tr>
<td>4305</td>
<td>The requested scratch record ID cannot be found.</td>
</tr>
<tr>
<td>4307</td>
<td>An I/O error has occurred during processing.</td>
</tr>
<tr>
<td>4331</td>
<td>The parameter list is invalid.</td>
</tr>
</tbody>
</table>

**DELETE TABLE (COBOL)**

The DELETE TABLE statement notifies CA IDMS that the issuing task has finished using a table that was loaded into the program pool by using the LOAD TABLE function. DELETE TABLE does not physically delete reusable tables from the program pool. Rather, this statement decrements the in-use count that CA IDMS maintains. An in-use count of 0 signals to reuse the space occupied by the table.

**Syntax**

```
DELETE TABLE from 01-level-program-location .
DICTNODE nodename    DICTNAME dictionary-name
```
Parameters

- **DELETE TABLE from 01-level-program-location**
  Specifies the LINKAGE SECTION entry of the 01-level record area that was specified in the associated LOAD TABLE request.

- **DICTNODE nodename**
  Specifies the node that controls the dictionary where the subschema containing the table resides. Specify the symbolic name of a user-defined eight-character field in variable storage.

- **DICTNAME dictionary-name**
  Specifies the dictionary where the subschema containing the table resides. Specify the symbolic name of a user-defined eight-character field in variable storage.

- **LOADLIB library-name**
  Specifies the load library containing the table. Specify the symbolic name of a user-defined eight-character field in variable storage.

Example

The following example releases a previously loaded table from the location in variable storage that was identified by RATE-TABLE:

```
DELETE TABLE FROM RATE-TABLE.
```

Status Codes

After completion of the DELETE TABLE function, the ERROR-STATUS field in the IDMS-DC communications block indicates the outcome:

<table>
<thead>
<tr>
<th>Status code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>The request has been serviced successfully.</td>
</tr>
<tr>
<td>3433</td>
<td>The specified table was not loaded by the task.</td>
</tr>
</tbody>
</table>

DEQUEUE (COBOL)

The DEQUEUE statement releases resources that were acquired by the issuing task with an ENQUEUE request. Acquired resources that are not released explicitly with a DEQUEUE request are released automatically at task termination.
Syntax

```
DEQUEUE ALL
  NAME resource-id LENGTH resource-id-length
```

Parameters

- **ALL**
  Releases all resources that were acquired by the issuing task by means of ENQUEUE requests.

- **NAME resource-id**
  Specifies a resource to dequeue. Specify the symbolic name of a user-defined field that contains the resource ID. If you specify multiple resources, separate them with at least one blank.

- **LENGTH resource-id-length**
  Specifies the length of the resource. Specify the symbolic name of a PIC S9(8) COMP SYNC (fullword) field that contains the length of the resource ID, or the length itself expressed as a numeric constant.

Example

The following statement releases all resources that were enqueued by the issuing task:

```
DEQUEUE PAYROLL-LOCK LENGTH 16.
```

Status Codes

After completion of the DEQUEUE function, the ERROR-STATUS field in the IDMS-DC communications block indicates the outcome:

<table>
<thead>
<tr>
<th>Status code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>The request has been serviced successfully.</td>
</tr>
<tr>
<td>3913</td>
<td>At least one resource ID cannot be found. All resources that were located have been dequeued.</td>
</tr>
<tr>
<td>3931</td>
<td>The parameter list is invalid.</td>
</tr>
</tbody>
</table>

**DISCONNECT (COBOL)**

The DISCONNECT statement cancels the current membership of a record occurrence in a set occurrence. The named record must be defined as an optional member of the named set.
Native VSAM users: The DISCONNECT statement is not valid because all sets in native VSAM data sets must be defined as mandatory automatic.

Before the DISCONNECT statement executes, the following conditions must be satisfied:

- All areas that are affected explicitly or implicitly by the DISCONNECT statement must be readied with one of the three update usage modes. (See READY (COBOL) [see page 228].)

- The named record must be established as current of its record type.

- The named record must currently participate as a member in an occurrence of the named set.

Following successful execution of the DISCONNECT statement, the named record can no longer be accessed through the set for which membership was cancelled. The disconnected record can still be accessed by a complete scan of the area in which it participates, or directly through its database key. A disconnected record can also be accessed through any other sets in which it participates as a member, or when it has a location mode of CALC.

Currency: A successfully executed DISCONNECT statement nullifies currency in the specified set. However, the next, prior, and owner of set are maintained, enabling continued access within the set. The disconnected record is current of run unit, its record type, its area, and any other sets in which it participates.

Syntax

```
►►─── DISCONNECT record-name FROM set-name . ────────────────────────────►◄
```

Parameters

- **DISCONNECT record-name**
  Specifies the record to disconnect. This record must be included in the subschema, and it must be defined as an optional member of the specified set.

- **FROM set-name**
  Specifies the set from which to disconnect the record. The set must be included in the subschema.

Example

The following figure illustrates how to disconnect an EMPLOYEE record from an occurrence of the OFFICE-EMPLOYEE set.

To disconnect EMPLOYEE 4 from the OFFICE 1 of the OFFICE-EMPLOYEE set, follow these steps:

1. Enter the database on OFFICE.
2. Establish EMPLOYEE 4 as current of the EMPLOYEE record type
3. Disconnect EMPLOYEE 4 from the OFFICE-EMPLOYEE set.
Disconnecting a Record from a Set

**Status Codes**

After completion of the DISCONNECT function, the ERROR-STATUS field in the IDMS communications block indicates the outcome:

<table>
<thead>
<tr>
<th>Status code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>The request has been serviced successfully.</td>
</tr>
<tr>
<td>1106</td>
<td>Currency has not been established for the named record.</td>
</tr>
<tr>
<td>1108</td>
<td>The named record is not in the subschema. The program has probably invoked the wrong subschema.</td>
</tr>
<tr>
<td>1109</td>
<td>The area for the named record has not been readied in one of the update usage modes.</td>
</tr>
<tr>
<td>1110</td>
<td>The subschema specifies an access restriction that prohibits use of the DISCONNECT statement.</td>
</tr>
<tr>
<td>Status code</td>
<td>Meaning</td>
</tr>
<tr>
<td>-------------</td>
<td>---------</td>
</tr>
<tr>
<td>1115</td>
<td>The DISCONNECT statement cannot be executed because the named record has been defined as a mandatory member of the set.</td>
</tr>
<tr>
<td>1121</td>
<td>An area other than the area that contains the named record has been readied with an incorrect usage mode.</td>
</tr>
<tr>
<td>1122</td>
<td>The named record is not currently a member of the specified set.</td>
</tr>
</tbody>
</table>

**END LINE TERMINAL SESSION (COBOL)**

The END LINE TERMINAL SESSION statement terminates the current line I/O session. All output data lines that remain in the current buffer and all pages that are queued for asynchronous I/O operations are deleted.

**Syntax**

```
END LINE TERMINAL SESSION.
```

**Example**

The following statement terminates a line mode I/O session:

```
END LINE TERMINAL SESSION.
```

**Status Codes**

No status codes are associated with the END LINE TERMINAL SESSION command.

**END TRANSACTION STATISTICS (COBOL)**

The END TRANSACTION STATISTICS statement defines the end of a transaction. The transaction typically ends when the issuing task terminates. Optionally, END TRANSACTION STATISTICS can write the transaction statistics block (TSB) to the DC system log file and return the TSB to a preallocated location in variable storage. You can define the TSB length.

**Syntax**

```
END TRANSACTION STATISTICS WRITE
     NOWRITE
     INTO return-stat-data-location
```
Parameters

- **WRITE|NOWRITE**
  Specifies whether to write the TSB to the DC system log file when the task terminates.
  **Default:** WRITE

- **INTO return-stat-data-location**
  Specifies the WORKING-STOREAGE SECTION or LINKAGE SECTION data area into which to return the TSB. Specify a fullword-aligned 388-byte field. (You can customize the length by using the LENGTH parameter.)

- **LENGTH len-return-TSB**
  Specifies the length of the returned TSB. Specify either the symbolic name of a user-defined field that contains the length of the TSB, or the length expressed as a numeric constant. To retrieve all statistics, including the DC extended statistics section that records CPU times in the Time of Day (TOD) format, specify LENGTH as 560.
  **Limits:** Integer of 388 or greater
  **Default:** If you do not specify *len-return-TSB*, the first 388 bytes of the TSB are returned.

Example

The following statement ends a transaction, writes statistics to the log file, and returns a copy of the TSB to the STATISTICS-BLOCK field:

```plaintext
END TRANSACTION STATISTICS
  WRITE
  INTO STATISTICS-BLOCK.
```

Status Codes

After completion of the END TRANSACTION STATISTICS function, the ERROR-STATUS field in the IDMS-DC communications block indicates the outcome:

<table>
<thead>
<tr>
<th>Status code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>The request has been serviced successfully.</td>
</tr>
<tr>
<td>3801</td>
<td>Storage for the transaction statistics block is not available; to wait would cause a deadlock.</td>
</tr>
<tr>
<td>3813</td>
<td>No transaction statistics block exists. A BIND TRANSACTION STATISTICS request has not been issued.</td>
</tr>
<tr>
<td>3831</td>
<td>The parameter list is invalid or no logical terminal element (LTE) is associated with the issuing task.</td>
</tr>
<tr>
<td>3850</td>
<td>The collection of transaction statistics or task statistics has not been enabled during system generation.</td>
</tr>
</tbody>
</table>
ENDPAGE (COBOL)

The ENDPAGE statement terminates a map paging session, clears the scratch record for the session, and clears the map paging options for the completed session. A STARTPAGE (COBOL) (see page 243) /ENDPAGE pair encloses the commands that handle a pageable map at runtime.

Syntax

```
       ENDPAGE session .
```

Example

The following statement ends a map paging session:

```
   ENDPAGE session .
```

Status Codes

After completion of the ENDPAGE function, the ERROR-STATUS field in the IDMS-DC communications block indicates the outcome:

<table>
<thead>
<tr>
<th>Status code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>The request has been serviced successfully.</td>
</tr>
</tbody>
</table>

ENQUEUE (COBOL)

The ENQUEUE statement acquires or tests resource availability. Resources are defined during installation and system generation. Resources typically include storage areas, common routines, queues, and processor time.

An enqueued resource can be exclusive or shared:

- **Exclusive** resources are owned exclusively by the issuing task and are not available to any other tasks. CA IDMS prohibits other tasks from obtaining resources that have been enqueued exclusively.

  \[\textbf{Note:} \text{An exclusive ENQUEUE request prohibits another task from enqueuing a resource by name. However, an exclusive enqueue does not prohibit the use of the resource by another task. For true resource protection, enqueue and dequeue resources consistently.}\]

- **Shared** resources are available to all tasks. CA IDMS allows other tasks to issue nonexclusive ENQUEUE requests for the resources, permitting the resources to be shared.
Syntax

ENQUEUE ▶► ──▼─WAIT ◄─NOWAIT ──▼─TEST ──▼─NAME resource-id LENGTH resource-length

Parameters

- **WAIT**
  Waits for all resources to be freed, if the request cannot be serviced immediately. This value is the default.

- **NOWAIT**
  Does not wait to acquire resources that are currently unavailable. If NOWAIT is specified, the program should check the ERROR-STATUS field in the IDMS-DC communications block to determine whether the function has completed. If the ERROR-STATUS value is 3901, a resource could not be obtained immediately, the request has not been serviced, and the program should perform alternative processing before reissuing the NOWAIT request.

- **TEST**
  Tests the availability of the specified resources. If TEST is specified, the program should check the ERROR-STATUS field in the IDMS-DC communications block to determine the outcome of the test.

- **NAME resource-id**
  Specifies the resource ID. Specify the symbolic name of a user-defined field that contains the resource name. The resource name is a 1- to 256-byte character string that identifies the resource. Any character string can be defined as long as all programs that access the resource use the same name, and as long as the name is unique relative to all other resource names within the CV. If you specify multiple resources, separate them with at least one blank.

- **LENGTH resource-id-length**
  Specifies the resource length. Specify the symbolic name of a user-defined field that contains the length of the resource ID, or the length itself expressed as a numeric constant.

- **EXCLUSIVE**
  Assigns the exclusive attribute to the named resource. This value is the default.

- **SHARED**
  Assigns the shared attribute to the named resource.

Examples

The examples below illustrate the use of the ENQUEUE statement:
Example 1
The following statement enqueues the CODE-VALUE and PAYROLL-LOCK resources. CODE-VALUE is reserved for exclusive use by the issuing task. PAYROLL-LOCK can be shared.

ENQUEUE
  WAIT
  NAME CODE-VALUE LENGTH 10
  NAME PAYROLL-LOCK LENGTH 16 SHARED.

Example 2
The following statement tests the availability of the resource that is identified in the RESOURCE-NAME field:

ENQUEUE
  TEST
  NAME RESOURCE-NAME LENGTH RESOURCE-LENGTH.

Status Codes

After completion of an ENQUEUE function to acquire resources, the ERROR-STATUS field in the IDMS DC communications block indicates the outcome:

<table>
<thead>
<tr>
<th>Status code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>The request has been serviced successfully.</td>
</tr>
<tr>
<td>3901</td>
<td>At least one requested resource cannot be enqueued immediately; to wait would cause a deadlock. No new resources have been acquired.</td>
</tr>
<tr>
<td>3908</td>
<td>At least one requested exclusive resource is currently owned by another task. No new resources have been acquired.</td>
</tr>
<tr>
<td>3931</td>
<td>The parameter list is invalid.</td>
</tr>
</tbody>
</table>

After completion of an ENQUEUE function to test resources, the ERROR-STATUS field in the IDMS DC communications block indicates the outcome:

<table>
<thead>
<tr>
<th>Status code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>All requested resources are available.</td>
</tr>
<tr>
<td>3908</td>
<td>At least one tested resource is already owned by another task.</td>
</tr>
<tr>
<td>3909</td>
<td>At least one tested resource is not yet owned by another task and is available to the issuing task.</td>
</tr>
<tr>
<td>3931</td>
<td>The parameter list is invalid.</td>
</tr>
</tbody>
</table>

ERASE (COBOL)

The ERASE statement performs the following functions:
• Disconnects the specified record from all set occurrences in which it participates as a member and logically or physically deletes the record from the database

• Optionally erases all records that are mandatory members of set occurrences owned by the specified record

• Optionally disconnects or erases all records that are optional members of set occurrences owned by the specified record

ERASE is a two-step procedure that cancels the existing membership of the named record in specific set occurrences and then releases for reuse the space that was occupied by the named record and its database key. Erased records are unavailable for further processing by any DML statement.

Before the ERASE statement executes, the following conditions must be satisfied:

• All areas that are affected implicitly or explicitly must be readied in one of the update usage modes. (See READY (COBOL) (see page 228).)

• All sets in which the specified record participates as owner either directly or indirectly (for example, as owner of a set with a member that is owner of another set) and all member record types in those sets must be included in the subschema in use.

• The specified record must be established as current of run unit.

Currency: Following successful execution of an ERASE statement, currency is nullified for all record types that are involved in the erase both explicitly and implicitly. Run unit and area currency remain unchanged. The next, prior, and owner currencies are preserved for sets from which the last record occurrence was erased. These currencies enable you to retrieve the next or prior records within the area or the next, prior, or owner records within the set in which the erased record participated. An attempt to retrieve erased records results in a non-zero status condition.

Syntax

```
ERASE record-name PERMANENT MEMBERS
```  
```
ERASE record-name SELECTIVE MEMBERS
```  
```
ERASE record-name ALL MEMBERS
```

Parameters

• **ERASE record-name**
  Erases the specified record. The record must be included in the subschema. The current of record-name must be current of run unit.
  Unless PERMANENT, SELECTIVE, or ALL qualification follows, a non-zero status condition results when the named record is the owner of any nonempty set occurrences.
  **Native VSAM users:** ERASE record-name is the only form of the ERASE statement that is valid for records in a native VSAM KSDS or RRDS. The ERASE statement is not valid for a native VSAM ESDS.

• **PERMANENT MEMBERS**
  Erases the specified record and all mandatory member record occurrences that are owned by the specified record. Optional member records are disconnected. If any of the erased mandatory members are themselves the owner of any set occurrences, the ERASE statement is executed on
those records as if they were directly the object record of an ERASE PERMANENT statement. (That is, all mandatory members of such sets are also erased.) This process continues until all direct and indirect members have been processed.

- **SELECTIVE MEMBERS**
  Erases the specified record and all mandatory member record occurrences that are owned by the specified record. Optional member records are erased if they do not currently participate as members in other set occurrences. All erased member records that are themselves the owners of any set occurrences are treated as if they were the object of an ERASE SELECTIVE statement.

- **ALL MEMBERS**
  Erases the specified record and all mandatory and optional member record occurrences that are owned by the specified record. All erased member records that are themselves the owners of any set occurrences are treated as if they were the object record of an ERASE ALL statement.

**Example**

The following figure illustrates the use of the three ERASE statement parameters.

The outcome of the ERASE statement varies based on the specified qualifier (PERMANENT, SELECTIVE, or ALL). Although all three qualifiers cause all mandatory members that are owned by the specified record to be erased, they differ in their effect on optional members.

Because the sample employee database provides no appropriate examples of these parameters, this figure and the one after it use a sample high school database instead.
ERASE DEPT PERMANENT
(assuming that FOREIGN LANGUAGES is current of run unit)

The Foreign Languages Department can no longer be funded, so it is deleted from the database along with its subjects and classes. The teachers will be reassigned to other departments.

Erases the foreign language record and all mandatory members; disconnects optional members.

ERASE TCHR SELECTIVE
(assuming that WON HAN is current of the run unit)

Won Han has quit in the middle of the semester. His classes will be finished by another teacher, so only Won Han is erased. (Remember that an unqualified ERASE command cannot be used to erase the owner of a non-empty set.)

Erases the TCHR record occurrence, mandatory members (none, TCHR-CLASS is OA), and optional orphans (none, CHI is in the SUBJ-CLASS set).
ERASE (4)

The following figure shows the effect each parameter has on currency.

<table>
<thead>
<tr>
<th>CURRENCIES: RUN UNIT, RECORD, SET, AREA</th>
</tr>
</thead>
<tbody>
<tr>
<td>RUN UNIT</td>
</tr>
<tr>
<td>-----------</td>
</tr>
<tr>
<td>ESTABLISHED CURRENCIES</td>
</tr>
<tr>
<td>ERASE DEPT PERMANENT</td>
</tr>
<tr>
<td>ESTABLISHED CURRENCIES</td>
</tr>
<tr>
<td>ERASE TOHR SELECTIVE</td>
</tr>
<tr>
<td>ESTABLISHED CURRENCIES</td>
</tr>
<tr>
<td>ERASE TOHR ALL</td>
</tr>
</tbody>
</table>

Status Codes

After completion of the ERASE function, the ERROR-STATUS field in the IDMS communications block indicates the outcome:
<table>
<thead>
<tr>
<th>Status code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>The request has been serviced successfully.</td>
</tr>
<tr>
<td>0203</td>
<td>Invalid currency for the named record to ERASE. This outcome occurs only when a run unit is sharing a transaction with other database sessions. The 03 minor status is returned when the run unit tries to erase a record using a currency that has been invalidated because of changes made by another database session that is sharing the same transaction.</td>
</tr>
<tr>
<td>0208</td>
<td>The object record is not in the specified subschema.</td>
</tr>
<tr>
<td>0209</td>
<td>The are of the named record has not been readied in one of the three update usage modes.</td>
</tr>
<tr>
<td>0210</td>
<td>The subschema specifies an access restriction that prohibits use of the ERASE statement. For SPF users, this code can also indicate the use of an invalid form of the ERASE statement.</td>
</tr>
<tr>
<td>0213</td>
<td>A current record of run unit has not been established or has been nullified by a previous ERASE statement.</td>
</tr>
<tr>
<td>0217</td>
<td>A database key has been encountered that contains a long-term permanent lock.</td>
</tr>
<tr>
<td>0220</td>
<td>The current record of run unit is not the same record type as the named record.</td>
</tr>
<tr>
<td>0221</td>
<td>An area other than the area of the specified record has been readied with an incorrect usage mode.</td>
</tr>
<tr>
<td>0225</td>
<td>Currency has not been established.</td>
</tr>
<tr>
<td>0226</td>
<td>A broken chain has been encountered in the process of executing an ERASE ALL, PERMANENT, or SELECTIVE.</td>
</tr>
<tr>
<td>0230</td>
<td>An attempt has been made to erase the owner record of a nonempty set.</td>
</tr>
<tr>
<td>0233</td>
<td>Erasure of the record occurrence is not allowed in this subschema, or all sets in which the record participates have not been included in the subschema.</td>
</tr>
<tr>
<td>0260</td>
<td>A record occurrence has been encountered whose type is inconsistent with the set named in the ERROR-SET field of the IDMS communications block. Probable causes include a broken chain and improper database description.</td>
</tr>
<tr>
<td>0261</td>
<td>The record cannot be erased because of broken chains in the database.</td>
</tr>
</tbody>
</table>

**ERASE (LRF) (COBOL)**

The ERASE statement deletes a logical-record occurrence. The ERASE statement does not necessarily result in the deletion of all or any of the database records that are used to create the specified logical record. The path that is selected to service an ERASE logical-record request performs whatever database access operations the DBA has specified to service the request. For example, if a DEPARTMENT loses an employee, the EMP-JOB-LR logical record that contains information about that employee would be erased. However, only the information about the former employee would be erased from the database, not all the information about the department. That is, EMPLOYEE information would be erased, but not DEPARTMENT, JOB, or OFFICE information.

LRF uses field values that are present in the variable-storage location reserved for the logical record to update the database. You can specify an alternative storage location from which LRF is to take field values to make the appropriate updates to the database.
Syntax

```plaintext
ERASE logical-record-name

FROM alt-logical-record-location

WHERE boolean-expression

ON path-status imperative-statement
```

Parameters

- **ERASE logical-record-name**
  Erases the specified logical record. The logical record must be defined in the subschema. Unless you specify FROM, LRF uses the field values that are present in the variable-storage location reserved for the logical record to make any necessary updates to the database.

- **FROM alt-logical-record-location**
  Names an alternative variable-storage location from which LRF is to obtain field values to perform the appropriate database updates. When you erase a logical record that has previously been retrieved into an alternative storage location, use the FROM clause to name the same location that was specified in the OBTAIN request. The alternate record location must be defined in the WORKING-STORAGE/LINKAGE SECTION.

- **WHERE boolean-expression**
  Specifies the selection criteria to apply to the logical record. For details on coding this clause, see Logical-Record Clauses (COBOL) (see page 272).

- **ON path-status imperative-statement**
  Specifies the action to take when `path-status` is returned to the LR-STATUS field in the LRC block. For details on coding this clause, see Logical-Record Clauses (COBOL) (see page 272).

Example

The following example erases all occurrences of the EMP-INSURANCE-LR logical record for a former employee. The DBA-designated path status ALL-ERASED indicates that all occurrences of the EMP-INSURANCE-LR logical record have been erased.

```plaintext
ERASE EMP-INSURANCE-LR WHERE EMP-ID-0415 EQ '0316'
ON ALL-ERASED PERFORM EMP-INS-DELETION-RPT.
```

As defined by the DBA, the ERASE EMP-INSURANCE-LR path group logically deletes all of the specified EMP-INSURANCE-LR occurrences but physically deletes only the COVERAGE records, as illustrated by the following figure:
FIND/OBTAIN (COBOL)

The FIND statement locates a record occurrence in the database. The OBTAIN statement locates a record and moves its associated data to the record buffers. Because the FIND and OBTAIN command statements have identical formats, they are discussed together. The FIND/OBTAIN statement has six formats:

- **FIND/OBTAIN CALC** (see page 144) accesses a record occurrence by using its CALC key value.
- **FIND/OBTAIN CURRENT (COBOL)** (see page 145) accesses a record occurrence by using established currencies.
- **FIND/OBTAIN DB-KEY (COBOL)** (see page 148) accesses a record occurrence by using its database key.
- **FIND/OBTAIN OWNER (COBOL)** (see page 150) accesses the owner record of a set occurrence.
- **FIND/OBTAIN WITHIN SET/AREA (COBOL)** (see page 152) accesses a record occurrence based on its logical location within a set or on its physical location within an area.
- **FIND/OBTAIN WITHIN SET USING SORT KEY (COBOL)** (see page 157) accesses a record occurrence in a sorted set by using its sort key value.

You can place locks on located record occurrences by using the KEEP clause of a FIND/OBTAIN statement. The KEEP clause sets a shared or exclusive lock:

- **KEEP** places a shared lock on the located record occurrence. Other concurrently executing run units can access the record but cannot update it.
- **KEEP EXCLUSIVE** places an exclusive lock on the located record occurrence. Other concurrently executing run units cannot access or update the record.

---

**CA IDMS Reference - 19.0**

**Logical-Record Occurrences**

<table>
<thead>
<tr>
<th>EMPLOYEE</th>
<th>INS-PLAN</th>
<th>COVERAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>316</td>
<td>001</td>
<td>D</td>
</tr>
<tr>
<td>316</td>
<td>002</td>
<td>M</td>
</tr>
<tr>
<td>316</td>
<td>001</td>
<td>F</td>
</tr>
</tbody>
</table>

**Physically Deleted**
FIND/OBTAIN CALC/DUPLICATE (COBOL)

The FIND/OBTAIN CALC/DUPLICATE statement locates a record based on the value of a CALC key in the record. The specified record must be stored in the database with a location mode of CALC. Before you issue the FIND/OBTAIN CALC/DUPLICATE statement, initialize a field in program variable storage with the CALC key value.

If a FIND/OBTAIN CALC statement has previously accessed an occurrence of the same record type, you can use the DUPLICATE option to access duplicate records with the same CALC key value as the record that is current of record type.

Currency: Following successful execution of a FIND/OBTAIN CALC/DUPLICATE statement, the accessed record becomes the current record of run unit, its record type, its area, and all sets in which it currently participates as member or owner.

Syntax

```
FIND/OBTAIN KEEP CALC record-name .
```

Parameters

- **KEEP**
  Places a shared lock on the accessed record.

- **EXCLUSIVE**
  Places an exclusive lock on the accessed record.

- **CALC|ANY**
  Locates the first or only occurrence of the specified record type whose CALC key matches the CALC data item in program variable storage. CALC and ANY are synonyms and can be used interchangeably.

- **DUPLICATE**
  Locates the next record with the same CALC key value as the current of the specified record type. The DUPLICATE option requires prior selection of an occurrence of the same record type with the CALC option. If the value of the CALC key in variable storage is not equal to the CALC key field of the current of record type, a status code of 0332 is returned.

- **record-name**
  Specifies the record type to locate.
Example

To retrieve an occurrence of the EMPLOYEE record by using the FIND/OBTAIN CALC/DUPLICATE statement, you must first initialize the variable-storage field that contains the CALC control element. The following statements initialize the CALC field EMP-ID-0415 and retrieve an occurrence of the EMPLOYEE record:

MOVE EMP-ID-IN TO EMP-ID-0415.
OBTAIN CALC EMPLOYEE.

Status Codes

After completion of the FIND/OBTAIN CALC/DUPLICATE function, the ERROR-STATUS field in the IDMS communications block indicates the outcome:

<table>
<thead>
<tr>
<th>Status code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>The request has been serviced successfully.</td>
</tr>
<tr>
<td>0301</td>
<td>The area in which the named record participates has not been readied.</td>
</tr>
<tr>
<td>0306</td>
<td>A successful FIND/OBTAIN CALC has not yet been executed. This code applies only to the DUPLICATE option.</td>
</tr>
<tr>
<td>0308</td>
<td>The named record is not in the subschema. The program probably invoked the wrong subschema.</td>
</tr>
<tr>
<td>0310</td>
<td>The subschema specifies an access restriction that prohibits retrieval of the named record.</td>
</tr>
<tr>
<td>0318</td>
<td>The record has not been bound.</td>
</tr>
<tr>
<td>0326</td>
<td>The record cannot be found or no duplicates exist for the named record.</td>
</tr>
<tr>
<td>0331</td>
<td>The retrieval statement format conflicts with the location mode of the record.</td>
</tr>
<tr>
<td>0332</td>
<td>The value of the CALC data item in program variable storage does not equal the value of the CALC data item in the current record. This code applies only to the DUPLICATE option.</td>
</tr>
<tr>
<td>0364</td>
<td>The CALC control element has not been described correctly in the program or in the subschema.</td>
</tr>
<tr>
<td>0370</td>
<td>A database file will not open properly.</td>
</tr>
</tbody>
</table>

If the FIND/OBTAIN statement includes an explicit KEEP, a major code of 03 is returned when an error occurs during FIND processing. A major code of 06 is returned when the error occurs during KEEP processing.

FIND/OBTAIN CURRENT (COBOL)

The FIND/OBTAIN CURRENT statement locates the record that is current of its record type, set, or area. This statement efficiently establishes the appropriate record as current of run unit before executing a DML statement that uses run-unit currency (for example, ACCEPT, IF, GET, MODIFY, ERASE).
Currency: Following successful execution of a FIND/OBTAIN CURRENT statement, the accessed record is current of run unit, its record type, its area, and all sets in which it currently participates as member or owner.

Syntax

```
FIND OBTAIN KEEP EXCLUSIVE CURRENT record-name WITHIN set-name WITHIN area-name
```

Parameters

- **KEEP**
  Places a shared lock on the accessed record.

- **EXCLUSIVE**
  Places an exclusive lock on the accessed record.

- **CURRENT**
  Locates the current record occurrence of a specified record type, set, or area.

- **record-name**
  Accesses the current record of the specified record type.

- **WITHIN set-name**
  Accesses the current record of the specified set.

- **WITHIN area-name**
  Accesses the current record of the specified area.

Example

The following figure uses the FIND/OBTAIN CURRENT statement to establish the proper record as current of run unit before the record is modified.

Enter the database on DEPARTMENT 5100 by using CALC retrieval. Then examine EMPLOYEE 466 by using within set retrieval and obtain further information from its owner OFFICE record (OFFICE 8). OFFICE 8 becomes current of run unit. Before modifying EMPLOYEE 466, you must issue the FIND CURRENT statement to reestablish EMPLOYEE 466 as current of run unit.

For a complete description of the MODIFY statement and its use, see MODIFY (COBOL) (see page 205).
After completion of the FIND/OBTAIN CURRENT function, the ERROR-STATUS field in the IDMS communications block indicates the outcome:

<table>
<thead>
<tr>
<th>Status code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>The request has been serviced successfully.</td>
</tr>
<tr>
<td>0301</td>
<td>The area in which the named record participates has not been readied.</td>
</tr>
<tr>
<td>0303</td>
<td></td>
</tr>
</tbody>
</table>
Status code | Meaning
--- | ---
0306 | Currency has not been established for the named record, set, or area.
0308 | The named record or set is not in the subschema. The program has probably invoked the wrong subschema.
0310 | The subschema specifies an access restriction that prohibits retrieval of the named record.
0313 | A current record of run unit either has not been established or has been nullified by a previous ERASE statement.
0323 | The specified area name has not been included in the subschema invoked.

If the FIND/OBTAIN statement includes an explicit KEEP, major code 03 is returned when an error occurs during FIND processing. Major code 06 is returned when the error occurs during KEEP processing.

**FIND/OBTAIN DB-KEY (COBOL)**

The FIND/OBTAIN DB-KEY statement locates a record occurrence directly by using a database key that the program previously stored. The DML `ACCEPT (COBOL)` (see page 88) statement or the COBOL MOVE statement can be used to save a database key. Any record in the program subschema can be accessed directly in this manner, regardless of its location mode.

**Native VSAM users:** This statement is not valid for accessing data records in a native VSAM key-sequenced data set (KSDS).

**Currency:** After successful execution of a FIND/OBTAIN DB-KEY statement, the accessed record becomes the current record of run unit, its record type, its area, and all sets in which it currently participates as member or owner. In addition, the RECORD-NAME field of the IDMS communications block is updated with the name of the accessed record. Currency is not used to determine the specified record of the FIND/OBTAIN DB-KEY statement; the record is identified by its database key and, optionally, by its record type.

**Syntax**

```
FIND OBTAIN [KEEP [EXCLUSIVE]] [DB-KEY is db-key] [PAGE-INFO page-info] [rec-name]
```

**Parameters**

- `KEEP`
  Places a shared lock on the accessed record.
EXCLUSIVE
Places an exclusive lock on the accessed record.

DB-KEY is db-key
Locates a record directly by using a database key value that is contained in program variable storage. Specify a field that identifies the location within program variable storage that contains a previously saved database key. If a record name has been specified, db-key must contain the database key of an occurrence of the named record type. If a record name has not been specified, db-key can contain the database key of an occurrence of any record type in the subschema.

PAGE-INFO page-info
Specifies the location in variable storage that contains the page information previously saved by the program. This page information is used to determine the area with which the database key is associated. Specify a 4-byte field, which can be defined either as a group field or as a fullword field (PIC S9(8) COMP). If this value is not specified, the page information that is associated with the record that is current of run unit is used.

Notes:
Page information is used only when the subschema includes areas with mixed page groups; otherwise, it is ignored.
Page information can also be returned by using an ACCEPT PAGE-INFO statement.

rec-name
Specifies the record type of the requested record. The record must be included in the subschema.

Example
The following statement locates the occurrence of the HOSPITAL-CLAIM record whose database key matches the value of a field in program variable storage called SAVED-KEY:

FIND HOSPITAL-CLAIM DB-KEY IS SAVED-KEY.

The located record becomes current of run unit, current of the HOSPITAL-CLAIM record type, current of the INS-DEMO-REGION area, and current of the COVERAGE-CLAIMS set.

Status Codes
After completion of the FIND/OBTAIN DB-KEY function, the ERROR-STATUS field in the IDMS communications block indicates the outcome:

<table>
<thead>
<tr>
<th>Status code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>The request has been serviced successfully.</td>
</tr>
<tr>
<td>0301</td>
<td>The area in which the named record participates has not been readied.</td>
</tr>
<tr>
<td>0302</td>
<td>The database key is inconsistent with the area in which the record is stored. The database key has not been initialized properly or the record name is incorrect.</td>
</tr>
</tbody>
</table>
| 0303        | }
<table>
<thead>
<tr>
<th>Status code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0308</td>
<td>The named record is not in the subschema. The program has probably invoked the wrong subschema.</td>
</tr>
<tr>
<td>0310</td>
<td>The subschema specifies an access restriction that prohibits retrieval of the named record.</td>
</tr>
<tr>
<td>0326</td>
<td>The record cannot be found. The record occurrence is not the correct type.</td>
</tr>
<tr>
<td>0370</td>
<td>A database file will not open properly.</td>
</tr>
<tr>
<td>0371</td>
<td>The requested page cannot be found in the DMCL.</td>
</tr>
</tbody>
</table>

Invalid currency for a record to be retrieved on a FIND. This code occurs only when a run unit is sharing a transaction with other database sessions. The 03 minor status is returned when the run unit tries to find a record using a currency that has been invalidated because of changes that were made by another database session that is sharing the same transaction.

If the FIND/OBTAIN statement includes an explicit KEEP, major code 03 is returned when an error occurs during FIND processing. Major code 06 is returned when the error occurs during KEEP processing.

**FIND/OBTAIN OWNER (COBOL)**

The FIND/OBTAIN OWNER statement locates the owner record of the current occurrence of a set. This statement can retrieve the owner record of any set regardless of whether that set has been assigned owner pointers.

**Native VSAM users:** The FIND/OBTAIN OWNER statement is not valid because owner records are not defined in native VSAM data sets.

**Currency:** To execute a FIND/OBTAIN OWNER statement, currency must be established for the specified set.

⚠️ **Note:** When a record that is declared as an optional or manual member of a set is retrieved, it is *not* established as current of set if it is not currently connected to the specified set. A subsequent attempt to retrieve the owner record locates the owner of the current record of set instead. In such cases, determine whether the retrieved record is actually a member in the specified set before executing the FIND/OBTAIN OWNER statement. The IF MEMBER (see page 173) statement can be used for this purpose.

Following successful execution of a FIND/OBTAIN OWNER statement, the accessed record becomes the current record of run unit, its record type, its area, and all sets in which it currently participates as member or owner. If the current record of set is the owner record when the statement is executed, currency within the specified set remains unchanged.

**Syntax**

```
FIND/OBTAIN OWNER WITHIN set-name .
```
Parameters

- **KEEP**
  Places a shared lock on the accessed record.

- **EXCLUSIVE**
  Places an exclusive lock on the accessed record.

- **OWNER**
  Locates the owner record of the specified set.

- **WITHIN set-name**
  Specifies a set that is included in the subschema.

Example

The following figure illustrates the use of the FIND/OBTAIN OWNER statement.
Status Codes

After completion of the FIND/OBTAIN OWNER function, the ERROR-STATUS field in the IDMS communications block indicates the outcome:

<table>
<thead>
<tr>
<th>Status code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>The request has been serviced successfully.</td>
</tr>
<tr>
<td>0301</td>
<td>The area in which the object record participates has not been readied.</td>
</tr>
<tr>
<td>0303</td>
<td>Invalid currency for a record to be retrieved on a FIND. This code occurs only when a run unit shares a transaction with other database sessions. The 03 minor status is returned when the run unit tries to find a record using a currency that has been invalidated because of changes that were made by another database session that is sharing the same transaction.</td>
</tr>
<tr>
<td>0306</td>
<td>Currency has not been established for the record, set, or area.</td>
</tr>
<tr>
<td>0308</td>
<td>The named set is not in the subschema. The program has probably invoked the wrong subschema.</td>
</tr>
<tr>
<td>0310</td>
<td>The subschema specifies an access restriction that prohibits retrieval of the object record.</td>
</tr>
<tr>
<td>0360</td>
<td>A record occurrence has been encountered whose record type is not a member or owner of the set as it is defined in the subschema.</td>
</tr>
<tr>
<td>0370</td>
<td>A database file will not open properly.</td>
</tr>
</tbody>
</table>

If the FIND/OBTAIN statement includes an explicit KEEP, major code 03 is returned when an error occurs during FIND processing. Major code 06 is returned when the error occurs during KEEP processing.

FIND/OBTAIN WITHIN SET/AREA (COBOL)

The FIND/OBTAIN WITHIN SET/AREA statement locates records logically, based on set relationships, or physically, based on database location. This statement lets you access each record in a set or area serially, or select specific occurrences of a given record type within the set or area.

The following rules apply to the selection of member records within a set:

- The set occurrence that is used as the basis for the operation is determined by the current record of the specified set. Set currency must be established before accessing records within a set.
- The next or prior record within a set is the subsequent or previous record relative to the current record of the named set in the logical order of the set. The prior record in a set can be retrieved only when the set has been assigned prior pointers.
- The first or last record within a set is the first or last member occurrence in terms of the logical order of the set. The selected record is the same one that would be selected if the current of set were the owner record and the next or prior record had been requested. The last record in a set can be retrieved only when the set has prior pointers.
The $n$th occurrence of a record within a set can be retrieved by specifying a sequence number that identifies the position of the record in the set. The DBMS begins its search with the owner of the current of set for the specified set and continues until it locates the $n$th record or encounters an end-of-set condition. If the specified sequence number is negative, the search proceeds in the prior direction within the set. A negative sequence number can be used only when the set has prior pointers; a sequence number of 0 produces a status code of 0304.

When an end-of-set condition occurs, the owner record occurrence of the set becomes the current record of run unit, current of its record type, current of its area, and current record of only the set involved in this operation. Currency of other sets in which the specified record participates as owner or member remains unaffected.

**Note:** If OBTAIN has been specified, the contents of the owner record are not moved to program variable storage. That is, OBTAIN under these circumstances is treated as a FIND.

**Native VSAM users:** When an end-of-set condition occurs, all currencies remain unchanged.

The following rules apply to the selection of records within an area:

- The first record occurrence within an area is the one with the lowest database key. The last record is the one with the highest database key.

- The next record within an area is the one with the next higher database key relative to the current record of the named area. The prior record is the one with the next lower database key relative to the current of area.

- The first or last or $n$th record in an area must be retrieved to establish the correct starting position before next or prior records are requested.

**Currency:** Following successful execution of a FIND/OBTAIN WITHIN SET/AREA statement, the accessed record becomes the current record of run unit, its record type, its area, and all sets in which it currently participates as member or owner.

### Syntax

```
FIND OBTAIN    KEEP     EXCLUSIVE
NEXT PRIOR FIRST LAST
record-name    WITHIN set-name area-name
```

### Parameters

- **KEEP**
  Places a shared lock on the accessed record.

- **EXCLUSIVE**
  Places an exclusive lock on the accessed record.
- **NEXT**
  Accesses the next record in the specified set or area relative to the current record.

- **PRIOR**
  Accesses the prior record in the specified set or area relative to the current record. The specified set must have prior pointers.

- **FIRST**
  Accesses the first record in the specified set or area.

- **LAST**
  Accesses the last record in the specified set or area. The specified set must have prior pointers.

- **number**
  Accesses the indicated record number in the specified set or area. *Number* must be a nonzero number or the symbolic name of a numeric field that contains a nonzero value. If the number is negative, the specified set must have prior pointers.

- **record-name**
  Specifies which occurrences of the named record type to access within a set or area. *Record-name* must be defined as a member of the specified set or contained within the specified area.

- **WITHIN**
  Locates a record based on its location within a set or area.

  - **set-name**
    Specifies the set to search. The set must be included in the subschema.

  - **area-name**
    Specifies the area to search. The area must be included in the subschema.

**Native VSAM users:** FIRST, LAST, and *sequence* options are not valid for a native VSAM KSDS with spanned records.

### Examples

**Retrieval of Records in an Occurrence of the DEPT-EMPLOYEE Set**

The following figure illustrates the retrieval of records within an occurrence of the DEPT-EMPLOYEE set.

The FIND CALC statement establishes currency in the DEPT-EMPLOYEE set. Member EMPLOYEE records are then retrieved by a series of OBTAIN WITHIN SET statements. EMPLOYEE 106 is the last record in the set. The next OBTAIN statement returns an end-of-set condition, positioning run unit currency at the owner of the set, DEPARTMENT 2000.
Retrieving Records in Area Containing Multiple Record Types

The following figure illustrates special considerations when retrieving records within an area that contains multiple record types.
In this example, a sweep of the EMP-DEMO-REGION is performed, retrieving sequentially each EMPLOYEE record and all records in the associated EMPLOYEE-EXPERTISE set. The first command retrieves EMPLOYEE 119. Subsequent OBTAIN WITHIN SET statements retrieve the associated EXPERTISE records and establish currency on EXPERTISE 03. The FIND CURRENT statement reestablishes the proper position before retrieving EMPLOYEE 48. If FIND CURRENT EMPLOYEE is not specified, an attempt to retrieve the next EMPLOYEE record in the area would return EMPLOYEE 23.

**FIND/OBTAIN WITHIN SET/AREA (3)**

**FIND/OBTAIN WITHIN SET/AREA (4)**

**FIND/OBTAIN WITHIN SET/AREA (5)**

**Status Codes**

After completion of the FIND/OBTAIN WITHIN SET/AREA function, the ERROR-STATUS field in the IDMS communications block indicates the outcome:

<table>
<thead>
<tr>
<th>Status code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>The request has been serviced successfully.</td>
</tr>
<tr>
<td>0301</td>
<td>The area in which the named record participates has not been readied.</td>
</tr>
<tr>
<td>0303</td>
<td></td>
</tr>
<tr>
<td>Status code</td>
<td>Meaning</td>
</tr>
<tr>
<td>-------------</td>
<td>---------</td>
</tr>
<tr>
<td>0306</td>
<td>A successful FIND/OBTAIN CALC has not yet been executed. This code applies only to the DUPLICATE option.</td>
</tr>
<tr>
<td>0307</td>
<td>The end of the set or area has been reached, or the set is empty.</td>
</tr>
<tr>
<td>0308</td>
<td>The named record is not in the subschema. The program probably invoked the wrong subschema.</td>
</tr>
<tr>
<td>0310</td>
<td>The subschema specifies an access restriction that prohibits retrieval of the named record.</td>
</tr>
<tr>
<td>0318</td>
<td>The record has not been bound.</td>
</tr>
<tr>
<td>0326</td>
<td>The record or SPF index entry cannot be found, or no duplicates exist for the named record.</td>
</tr>
<tr>
<td>0331</td>
<td>The retrieval statement format conflicts with the location mode of the record.</td>
</tr>
<tr>
<td>0332</td>
<td>The value of the CALC data item in program variable storage does not equal the value of the CALC data item in current record. This code applies only to the DUPLICATE option only.</td>
</tr>
<tr>
<td>0364</td>
<td>The CALC control element has not been described correctly in the program or in the subschema.</td>
</tr>
<tr>
<td>0370</td>
<td>A database file will not open properly.</td>
</tr>
</tbody>
</table>

If the FIND/OBTAIN statement includes an explicit KEEP, major code 03 is returned when an error occurs during FIND processing. Major code 06 is returned when the error occurs during KEEP processing.

**FIND/OBTAIN WITHIN SET USING SORT KEY (COBOL)**

The FIND/OBTAIN WITHIN SET USING SORT KEY statement locates a member record in a sorted set. Sorted sets are ordered in ascending or descending sequence based on the value of a sort-control element in each member record. The search begins with the current of set or the owner of the current of set and always proceeds through the set in the next direction.

Before you issue this statement, initialize the sort control element in program variable storage. The selected record occurrence will have a key value that is equal to the value of the sort control element. If more than one record occurrence contains a sort key equal to the key value in variable storage, the first one is selected.

In a batch environment, sorted sets can be processed more efficiently by sorting the input transactions.

**Currency:** Following successful execution of a FIND/OBTAIN WITHIN SET USING SORT KEY statement, the accessed record becomes current of run unit, its record type, its area, and all sets in which it currently participates as member or owner. If a member record with the requested sort key value is not found, the current of set is nullified but the next of set and prior of set are maintained. The next
of set is the member record with the next higher sort key value (or next lower for descending sets) than the requested value. The prior of set is the member record with the next lower value (or higher for descending sets) than requested. Because these currencies are maintained, the program can walk the set to do a generic search on the sort key value.

Syntax

```
FIND record-name WITHIN set-name

OBTAIN KEEP EXCLUSIVE

CURRENT USING sort-key
```

Parameters

- **KEEP**
  Places a shared lock on the accessed record.

- **EXCLUSIVE**
  Places an exclusive lock on the accessed record.

- **record-name**
  Specifies the record type to locate.

- **WITHIN set-name**
  Specifies the set to search. The set must be included in the subschema. Unless you specify CURRENT, the search begins with the owner of the specified set.

- **CURRENT**
  Begins the search with the currencies that are already established for the specified set. If the key value for the record that is current of set is higher than the key value of the requested record (assuming ascending set order), a nonzero status condition results. In a descending set order, when the key value for the record that is current of set is lower than the key value of the requested record, a nonzero status condition results.

- **USING sort-key**
  Specifies the sort control element to use in searching the sorted set. Specify the symbolic name of a field that is defined in working storage and that contains the value of the sort control element.

**Note:**

Due to the architecture of the client interface for CA IDMS, 256 bytes are moved regardless of the actual length of the working storage sort key. Account for this additional storage to avoid potential program exceptions that can occur. While these exceptions are rare, they are more probable when the sort key is defined in a FILE or LINKAGE SECTION definition. To avoid this problem, define the sort key in the WORKING STORAGE SECTION of the program, padded to a full 256 bytes; and moved in and out of the FILE or LINKAGE SECTION fields.
The value coded for sort-key can specify only a single field name. If the sort key has of multiple elementary fields, the value that is coded should be a group-level name. The elementary fields that make up the group element must be in the same sequence as the corresponding fields in the schema definition for the database set. The data formats for the individual elementary fields must also match the formats of the corresponding fields within the database record.

A period or semicolon is required to terminate the statement unless an ON clause has been coded.

Here is an example of OBTAIN RECA WITHIN RECA-SET USING RECA-KEY. The sort key of the record would be defined as follows in the WORKING-STORAGE SECTION:

```plaintext
01 RECA-KEY.
 02 RECA-FIELD1 PIC X(10).
 02 RECA-FIELD2 PIC X(10).
```

The definition should be changed to:

```plaintext
01 RECA-KEY.
 02 RECA-FIELD1 PIC X(10).
 02 RECA-FIELD2 PIC X(10).
 02 FILLER PIC X(236).
```

Example

The following example illustrates the use of a FIND/OBTAIN WITHIN SET USING SORT KEY statement. Assume that the SKILL-NAMES-NDX set is ordered in ascending sequence based on the value stored in SKILL-NAMES-0455 in each SKILL record occurrence. The following statements retrieve a SKILL record with a skill name equal to PL/I:

```plaintext
MOVE 'PL/I' TO SKILL-NAMES-0455.
FIND SKILL WITHIN SKILL-NAMES-NDX USING SKILL-NAMES-0455.
```

Status Codes

After completion of the FIND/OBTAIN WITHIN SET USING SORT KEY function, the ERROR-STATUS field in the IDMS communications block indicates the outcome:

<table>
<thead>
<tr>
<th>Status code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>The request has been serviced successfully.</td>
</tr>
<tr>
<td>0301</td>
<td>The area in which the named record participates has not been readied.</td>
</tr>
<tr>
<td>0303</td>
<td>Invalid currency for a record to be retrieved on a FIND. This code occurs only when a run unit shares a transaction with other database sessions. The 03 minor status is returned when the run unit tries to find a record using a currency that has been invalidated because of changes that were made by another database session that is sharing the same transaction.</td>
</tr>
<tr>
<td>0306</td>
<td>Currency has not been established for the named set.</td>
</tr>
<tr>
<td>0308</td>
<td>The named record or set is not in the subschema or the named record is not a member of the named set. The program has probably invoked the wrong subschema.</td>
</tr>
<tr>
<td>0310</td>
<td>The subschema specifies an access restriction that prohibits retrieval of the named record.</td>
</tr>
</tbody>
</table>
### Status codes

<table>
<thead>
<tr>
<th>Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0326</td>
<td>The record cannot be found.</td>
</tr>
<tr>
<td>0331</td>
<td>The retrieval statement format conflicts with the location mode of the record.</td>
</tr>
<tr>
<td>0360</td>
<td>A record occurrence has been encountered whose record type is not a member or owner of the set as it is defined in the subschema.</td>
</tr>
<tr>
<td>0370</td>
<td>A database file will not open properly.</td>
</tr>
</tbody>
</table>

If the FIND/OBTAIN statement includes an explicit KEEP, major code 03 is returned when an error occurs during FIND processing. Major code 06 is returned when the error occurs during KEEP processing.

## FINISH (COBOL)

The FINISH statement commits database changes that were made through an individual run unit or through all database sessions that are associated with a task. A task-level finish also commits all changes that were made with scratch, queue, and print activity.

If the FINISH applies to an individual run unit and the run unit is sharing its transaction with another database session, the changes may not be committed when the FINISH statement is executed.

⚠️ **Note:** For more information about the impact of transaction sharing, see [Sharing Transactions Among Sessions](https://docops.ca.com/display/IDMSCU19/Sharing+Transactions+Among+Sessions).

Run units (and SQL sessions) that are impacted by the FINISH statement end, and their access to the database terminates.

The FINISH statement is used in navigational environments and in logical record facility environments. The FINISH TASK statement is also used in SQL programming environments.

**Currency:** Following the successful execution of a FINISH request, all currencies are set to null. The issuing program or task cannot perform database access through an impacted run unit without executing another BIND/READY sequence.

### Syntax

```plaintext
FINISH TASK .
```
Parameters

- **TASK**
  Commits the changes and terminates the run units. All changes from scratch, queue, and print activity are committed. All top-level run units that are associated with the current task are terminated. The impact on SQL sessions that are associated with the task depends on whether those sessions are suspended and whether their transactions are eligible to be shared.
  
  More information:
  For more information about the impact of a FINISH TASK statement on SQL sessions, see Effect of Teleprocessing Statements and Events (https://docops.ca.com/display/IDMS19/Effect+of+Teleprocessing+Statements+and+Events).
  For more information about run units and the impact of FINISH TASK, see Sharing Transactions Among Sessions (https://docops.ca.com/display/IDMSCU19/Sharing+Transactions+Among+Sessions).

Example

The following statement commits changes that were made by the run unit through which it is issued and terminates that run unit:

```
FINISH.
```

Status Codes

After completion of the FINISH function, the ERROR-STATUS field in the IDMS communications block indicates the outcome:

<table>
<thead>
<tr>
<th>Status code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>The request has been serviced successfully.</td>
</tr>
<tr>
<td>5031</td>
<td>The specified request is invalid. The program may contain a logic error.</td>
</tr>
<tr>
<td>5097</td>
<td>An error was encountered while processing a syncpoint request. Look at the log for details.</td>
</tr>
</tbody>
</table>

FREE STORAGE (COBOL)

The FREE STORAGE statement releases all or part of a variable storage area. The storage must have been acquired by a GET STORAGE request in the issuing task or by another task running on the same terminal as the issuing task. A partial release is valid only for user storage; shared storage must be freed in its entirety.

Syntax

```
FREE STORAGE STGID storage-id .
```
Parameters

- **STGID storage-id**
  Specifies the variable storage area to release. Specify the symbolic name of a user-defined field that contains the ID, or the ID itself enclosed in quotation marks.

- **FOR 01-level-storage-data-loc**
  Specifies the LINKAGE SECTION entry of the storage area.

- **FROM start-free-storage-loc**
  Specifies the starting point of the release. Specify the symbolic name of a user-defined field that contains the starting point of the storage area. The storage is released from the specified location to the end of the storage area.

Example

The following example releases the storage area that is identified by 09PA:

```cobol
FREE STORAGE STGID '09PA'.
```

Status Codes

After completion of the FREE STORAGE function, the ERROR-STATUS field in the IDMS-DC communications block indicates the outcome:

<table>
<thead>
<tr>
<th>Status code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>The request has been serviced successfully.</td>
</tr>
<tr>
<td>3213</td>
<td>The requested storage ID cannot be found.</td>
</tr>
<tr>
<td>3232</td>
<td>The derived length of the variable storage area is zero or negative.</td>
</tr>
<tr>
<td>3234</td>
<td>The request cannot be serviced because the variable storage area is not an 01-level entry in the LINKAGE SECTION.</td>
</tr>
</tbody>
</table>

GET (COBOL)

The GET statement transfers the contents of a specified record occurrence from the record buffer to program variable storage. Elements in the specified record are moved to their respective locations in variable storage according to the subschema view of the record. The transferred elements appear in storage at the location to which the record has been bound. (For further details, see BIND RECORD (COBOL) (see page 108).)
Currency: The GET statement operates only on the record that is current of run unit. Following successful execution of a GET statement, the accessed record is current of run unit, its record type, its area, and all sets in which it participates as member or owner.

Syntax

```
►►─── GET ▼ record-name ◄◄
```

Parameters

- **GET record-name**
  Specifies that the current of run unit must be an occurrence of the named record type.

Example

The following statement moves the record that is current of run unit (in this case, the OFFICE record) from the record buffer into program variable storage:

```
GET OFFICE.
```

Status Codes

After completion of the GET function, the ERROR-STATUS field in the IDMS communications block indicates the outcome:

<table>
<thead>
<tr>
<th>Status code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>The request has been serviced successfully.</td>
</tr>
<tr>
<td>0503</td>
<td>Invalid currency for a record to be retrieved on a GET. This code occurs only when a run unit is sharing a transaction with other database sessions. The 03 minor status is returned when the run unit tries to get a record using a currency that has been invalidated because of changes that were made by another database session that is sharing the same transaction.</td>
</tr>
<tr>
<td>0506</td>
<td>Currency has not been established.</td>
</tr>
<tr>
<td>0508</td>
<td>The named record is not in the subschema. The program has probably invoked the wrong subschema.</td>
</tr>
<tr>
<td>0510</td>
<td>The subschema specifies an access restriction that prohibits retrieval of the named record.</td>
</tr>
<tr>
<td>0513</td>
<td>A current record of run unit has not been established or has been nullified by a previous ERASE statement.</td>
</tr>
<tr>
<td>0518</td>
<td>The record has not been bound.</td>
</tr>
<tr>
<td>0520</td>
<td>The current record is not the same type as the named record.</td>
</tr>
<tr>
<td>0526</td>
<td>The requested record has been erased.</td>
</tr>
<tr>
<td>0555</td>
<td>An invalid length has been returned for a variable-length record.</td>
</tr>
</tbody>
</table>
GET QUEUE (COBOL)

The GET QUEUE statement retrieves a queue record and places it in a storage area that is associated with the issuing program. If the queue record is larger than the designated storage area, the record is truncated. The retrieved record is automatically deleted from the queue unless the GET QUEUE statement explicitly requests to keep the record in the queue.

Syntax

```
GET QUEUE ID queue-id NEXT | FIRST | LAST | PRIOR | SEQUENCE number | RECORD ID queue-record-id

DELETE KEEP | LOCK NOLOCK | WAIT NOWAIT

INTO return-queue-data-location TO end-queue-data-location MAX LENGTH queue-data-max-length

RETURN LENGTH INTO queue-data-actual-length
```

Parameters

- **ID queue-id**
  Specifies the queue that is associated with the record to retrieve. Specify the symbolic name of a user-defined field that contains the ID, or the ID itself enclosed in quotation marks. If the queue ID is not specified, a null ID of 16 blanks is assumed.

- **NEXT**
  Retrieves the next record in the queue. This value is the default. If currency has not been established, NEXT is equivalent to FIRST.

- **FIRST**
  Retrieves the first record in the queue.

- **LAST**
  Retrieves the last record in the queue.

- **PRIOR**
  Retrieves the prior record in the queue. If currency has not been established, PRIOR is equivalent to LAST.

- **SEQUENCE number**
  Retrieves the specified queue record. Specify the symbolic name of a user-defined field that contains the sequence number of the record, or the sequence number itself expressed as a numeric constant.
- **RECORD ID** `queue-record-id`
  Retrieves the specified record. Specify the symbolic name of the PIC S9(8) COMP (fullword) field containing the queue record ID that the PUT QUEUE function returned.

- **DELETE**
  Deletes the record from the queue. This value is the default. If DELETE is specified and the record has been truncated, the truncated data is lost.

- **KEEP**
  Keeps the record in the queue.

- **LOCK|NOLOCK**
  These parameters have been nonfunctional since CA IDMS Release 12.0, and are included only for release compatibility. Queue record locking is now performed as part of the standard database locking routines.

- **WAIT**
  Suspends task execution until the requested queue exists.

- **NOWAIT**
  Continues task execution when a queue is nonexistent. This value is the default.

- **INTO return-queue-data-location**
  Specifies the WORKING-STORAGE SECTION or LINKAGE SECTION entry of the data area that is reserved for the requested queue record. Specify the symbolic name of a user-defined field.

- **TO end-queue-data-location**
  Specifies the end of the WORKING-STORAGE SECTION or LINKAGE SECTION entry that is reserved for the requested queue record. Specify the symbolic name of a user-defined dummy byte field or a field with a data item that is not associated with the requested queue record.

- **MAX LENGTH** `queue-data-max-length`
  Explicitly defines the length of the data area that is reserved for the requested queue record. Specify the symbolic name of the user-defined field that contains the length of the queue records data, or the length itself expressed as a numeric constant.

- **RETURN LENGTH INTO** `queue-data-actual-length`
  Specifies where to return the actual length of the retrieved queue record. Specify the symbolic name of a user-defined 4-byte field. If the record has been truncated, the returned value is the actual length of the queue record before truncation.

**Example**

The following example retrieves the first record in the RES-Q queue, returns it to the PEND-RES field, and keeps the record in the queue:

```plaintext
GET QUEUE
  ID 'RES-Q'
  FIRST
  KEEP
  INTO PEND-RES MAX LENGTH 125.
```
### Status Codes

After completion of the GET QUEUE function, the ERROR-STATUS field of the IDMS-DC communications block indicates the outcome:

<table>
<thead>
<tr>
<th>Status code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>The request has been serviced successfully.</td>
</tr>
<tr>
<td>4404</td>
<td>The requested queue header record cannot be found.</td>
</tr>
<tr>
<td>4405</td>
<td>The requested queue record cannot be found.</td>
</tr>
<tr>
<td>4407</td>
<td>A database error occurred during queue processing. A DBKEY deadlock is a common cause. For a PUT QUEUE operation, this code can also mean that the queue upper limit has been reached.</td>
</tr>
<tr>
<td></td>
<td>If a database error has occurred, the CA-IDMS/DC/UCF log usually includes other messages indicating a problem encountered in RHDCRUAL, the internal Run Unit Manager. If a deadlock has occurred, messages DC001000 and DC001002 are also produced.</td>
</tr>
<tr>
<td>4419</td>
<td>The program storage area that was specified for return of the queue record is too small. The returned record has been truncated to fit the available space. In a DC-BATCH environment, the size of the queue record may exceed the MAX LENGTH value specified in the BIND TASK statement. This problem can also happen when the record size that is specified in the GET QUEUE statement is large enough for the queue record, but larger than the maximum that is specified in the BIND TASK statement. The record size is always truncated to the maximum length that is specified in the BIND TASK statement.</td>
</tr>
<tr>
<td>4431</td>
<td>The parameter list is invalid. In DC-BATCH, this code signifies that the specified record length has exceeded the maximum length based on the packet size.</td>
</tr>
<tr>
<td>4432</td>
<td>The derived length of the queue record data area is negative.</td>
</tr>
</tbody>
</table>

### GET SCRATCH (COBOL)

The GET SCRATCH statement obtains a scratch record and places it in a storage area that is associated with the issuing program. The storage area must be allocated to the requesting task already. No implicit GET STORAGE function is performed during the GET SCRATCH operation. If the scratch record is larger than the designated storage area, the data is truncated.

### Syntax

```
GET SCRATCH AREA ID scratch-area-id
    NEXT
    FIRST
    LAST
    PRIOR
    CURRENT
    RECORD ID scratch-record-id
    INTO return-scratch-data-loc TO end-scratch-data-loc
    MAX LENGTH scratch-data-max-length
    KEEP
    DELETE
```

---

15-Jan-2018 166/415
Parameters

- **AREA ID scratch-area-id**
  Specifies the scratch area that is associated with the record to retrieve. Specify the symbolic name of a user-defined field that contains the scratch area ID, or the ID itself enclosed in quotation marks. If you do not specify an area ID, an area ID of eight blanks is assumed.

- **NEXT**
  Retrieves the next record in the scratch area. This value is the default.

- **FIRST**
  Retrieves the first record in the scratch area.

- **LAST**
  Retrieves the last record in the scratch area.

- **PRIOR**
  Retrieves the prior record in the scratch area.

- **CURRENT**
  Retrieves the current record in the scratch area. The current record is the record that was most recently referenced by another scratch function.

- **RECORD ID scratch-record-id**
  Specifies the scratch record to retrieve. Specify the symbolic name of a user-defined PIC S9(8) COMP SYNC (fullword) field containing the 4-byte scratch record ID.

- **DELETE**
  Deletes the record from the scratch area. This value is the default. If DELETE is specified and the record has been truncated, the truncated data is lost. To maintain currency following a DELETE request, CA IDMS saves the next and prior currencies of the scratch area.

- **KEEP**
  Keeps the record in the scratch area.

- **INTO return-scratch-data-loc**
  Specifies the WORKING-STORAGE SECTION or LINKAGE SECTION entry of the data area in which to return the scratch record. Specify the symbolic name of a user-defined field.

- **TO end-scratch-data-loc**
  Specifies the end of the data area in which the scratch record is returned. Specify the symbolic name of a user-defined dummy byte field or a field containing a data item that is not associated with the scratch record.

- **MAX LENGTH scratch-data-max-length**
  Specifies the length in bytes of the data area that is associated with the requested scratch record. Specify the symbolic name of a WORKING-STORAGE SECTION or LINKAGE SECTION field that contains the length, or the length itself expressed as a numeric constant.
• RETURN LENGTH into scratch-data-actual-length
  Specifies the WORKING-STORAGE SECTION or LINKAGE SECTION entry in which to return the
  actual length of the requested scratch record. Specify the symbolic name of the entry. If the
  record has been truncated, scratch-data-actual-length contains the length of the full, untruncated
  scratch record.

Example

The following statement returns the contents of the current record in the scratch area to the variable-
storage area that is defined by WORK-PROC-AREA and END-WORK-PROC-AREA:

GET SCRATCH
  CURRENT
  INTO WORK-PROC-AREA TO END-WORK-PROC-AREA.

Status Codes

After completion of the GET SCRATCH function, the ERROR-STATUS field of the IDMS-DC
communications block indicates the outcome:

<table>
<thead>
<tr>
<th>Status code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>The request has been serviced successfully.</td>
</tr>
<tr>
<td>4303</td>
<td>The requested scratch area ID cannot be found.</td>
</tr>
<tr>
<td>4305</td>
<td>The requested scratch record ID cannot be found.</td>
</tr>
<tr>
<td>4307</td>
<td>An I/O error has occurred during processing.</td>
</tr>
</tbody>
</table>
| 4319        | The program storage area that was specified for return of the scratch record is too small. The
              returned record has been truncated to fit the available space. |
| 4331        | The parameter list is invalid. |
| 4332        | The derived length of the scratch record is negative. |

GET STORAGE (COBOL)

The GET STORAGE statement is used to acquire variable storage from a DC system storage pool or to
obtain the address of a previously acquired storage area. After the storage is acquired, it is available
for use by the following entities:

• The issuing task only (user storage)

• Subsequent tasks running on the same terminal (user kept storage)

• All tasks in the system (shared or shared kept storage)

The GET STORAGE parameters govern the storage availability.
Syntax

GET STORAGE FOR 01-level-storage-data-location
TO end-storage-data-location
LENGTH storage-data-length
POINTER storage-data-location-pointer
WAIT NOWAIT KEEP LONG SHORT USER SHARED
STGID storage-id VALUE IS LOW-VALUE HIGH-VALUE
LOCATION is ANY BELOW

Parameters

- **FOR 01-level-storage-data-location**
  Specifies the LINKAGE SECTION entry of the storage area to acquire. Specify the name of the 01-level entry that is used to acquire the storage area.

- **TO end-storage-data-location**
  Specifies the end of the storage area. Specify the symbolic name of a user-defined dummy byte field. This value is specified as a subordinate item within the 01-level record area following the last real data field.

  **Note:** CA IDMS does not support an OCCURS DEPENDING ON clause within 01-level-storage-data-location.

  This parameter is required when the precompiler execution option COBOL=1 is specified. This parameter is accepted but not required when COBOL=2 is specified. For more information about the COBOL= option, see Passing Parameters to the Precompiler (see page 280).

- **LENGTH storage-data-length**
  Specifies the length of the storage location (COBOL 85 only). Specify the symbolic name of a user-defined field that contains the length of the storage location.

- **POINTER storage-data-location-pointer**
  Specifies the pointer to receive the address of the storage location (COBOL 85 only). Specify the symbolic name of a user-defined field that contains a pointer to the storage location address.

- **WAIT**
  Specifies that the issuing task waits until sufficient storage is available in a storage pool. This value is the default.
• **NOWAIT**
  Specifies that the issuing task does not wait for storage to become available when an insufficient storage condition exists. If NOWAIT is specified, the program should check the ERROR-STATUS field in the IDMS-DC communications block to determine whether the GET STORAGE request has been completed. If the ERROR-STATUS value is 3202 (DC-NO-STORAGE), the program should perform alternative processing before reissuing the GET STORAGE request.

• **KEEP**
  Allows subsequent tasks that are executing on the same logical terminal to access the storage area. For a more detailed explanation of the KEEP parameter, see [Using Storage Pools](https://docops.ca.com/display/IDMSCU19/Using+Storage+Pools).

• **LONG**
  Allocates storage from the bottom of the storage pool. This value is the default. Specify LONG when allocating kept storage to be held across pseudo-converses.

• **SHORT**
  Allocates storage from the top of the storage pool. Specify SHORT when allocating small pieces of storage for a short duration.

  **Note:** An incorrect LONG/SHORT specification does not affect normal program execution; however, it can affect the overall performance of the DC system.

• **USER**
  Allows only the issuing task to access the storage area. If KEEP is specified, only subsequent tasks that are executing on the same terminal can access the storage area. This value is the default.

  **Note:** During system execution, a program that is defined at sysgen with the NOPROTECT option can access any storage area within the system, including an area that is associated exclusively with another task. Thus, the USER attribute may not protect the storage area that is being acquired. However, storage areas can be protected on a systemwide or program-by-program basis during system generation and by the modes that are specified when storage is allocated.

• **SHARED**
  Permits any task in the system to access and modify the acquired storage. Each task must establish addressability to the storage area by explicitly issuing a GET STORAGE request.

• **STGID storage-id**
  Specifies the storage area. The STGID parameter must be specified with GET STORAGE requests for previously allocated storage areas or for areas to be reallocated. Specify the symbolic name of a user-defined field that contains the storage ID, or the ID itself enclosed in quotation marks. The ID must be unique. Even though multiple variable storage areas (that is, one shared and the others user) can have the same ID, only one area can be owned by a given task at a time. To access the CA IDMS common work area, specify STGID ‘CWA’.
Note: If the STGID parameter specifies the address of an existing storage area, the USER /SHARED parameter must specify the same option as the GET STORAGE statement that originally allocated the storage area.

- **VALUE IS**
  Specifies how to initialize the storage area.
  - **LOW-VALUE**
    Initializes the storage area to all zeros.
  - **HIGH-VALUE**
    Initializes the storage area to the highest value in the computer collating sequence.
  - **initial-value**
    Specify the symbolic name of a user-defined field containing the initial value, or the value itself enclosed in quotation marks. All bytes of the acquired storage area are initialized to the same value.

- **LOCATION is**
  Specifies whether the storage is restricted to below the 16-megabyte line.
  - **ANY**
    Specifies that space above the 16-megabyte line is eligible for allocation. This value is the default.
  - **BELOW**
    Specifies that storage must be allocated below the 16-megabyte line.

**Example**

The following statement allocates the shared kept storage area, 09PA, and initializes it to all zeros:

```
GET STORAGE FOR EMPLMENU-KEPT-STORAGE TO
  EMPLMENU-KEPT-STORAGE-END
NOWAIT
KEEP
SHORT
SHARED
STGID '09PA'
VALUE IS LOW-VALUE.
```

**Status Codes**

After completion of the GET STORAGE function, the ERROR-STATUS field of the IDMS-DC communications block indicates the outcome:

<table>
<thead>
<tr>
<th>Status code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>The request has been serviced successfully.</td>
</tr>
<tr>
<td>3201</td>
<td>The requested storage cannot be allocated immediately; to wait would cause a deadlock.</td>
</tr>
</tbody>
</table>
Status code | Meaning
--- | ---
3202 | The requested storage cannot be allocated because insufficient space exists in the storage pool.
3210 | The request specified a storage ID that did not previously exist. The required space has been allocated.
3231 | The request specifies an invalid parameter list.
3232 | The requested length is zero or negative.
3234 | The request cannot be serviced because the variable storage area is not an 01-level LINKAGE SECTION variable.
3235 | The request cannot be serviced because the specified 01-level LINKAGE SECTION entry has been previously allocated or contains an OCCURS DEPENDING ON clause.

GET TIME (COBOL)

The GET TIME statement obtains the time of day and date from the operating system. The system time is returned to the issuing task in fixed binary, packed decimal, or edited format. The date is returned to the program in packed decimal format.

Syntax

```
GET TIME INTO return-time COMP COMP-3 EDIT
DATE INTO return-date
```

Parameters

- **INTO return-time**
  Specifies the field in which to return the time. Specify the symbolic name of a user-defined field. The following format options apply.

  - **COMP**
    Returns the time as a fixed binary value that represents the elapsed time since midnight in ten-thousandths of a second. The COMP option returns the most precise time. This value is the default.
    When COMP is specified, the field that is associated with `return-time` must be a PIC S9(8) COMP SYNC (fullword) field.

  - **COMP-3**
    Returns the time as a 6-byte packed decimal value in the format `Ohhmmssstttttt` (padded zero, hours, minutes, seconds, ten-thousandths of a second, sign).
    When COMP-3 is specified, the field that is associated with `return-time` must be defined as PIC S9(11) COMP-3.
- **EDIT**
  Returns the time as an edited character string in the format `hh:mm:ss:hh` (hours, minutes, seconds, hundredths of a second).
  When EDIT is specified, `return-time` must be defined as PIC X(11) DISPLAY.

- **DATE INTO return-date**
  Specifies the field in which to return the date. Specify the symbolic name of a user-defined COMP3 PIC S9(7) field. The date is returned in the Julian format `Oyydddc` (padded zero, current year relative to 1900, date, sign). For example, 0099365C represents December 31, 1999. 0100001C represents January 1, 2000.

**Example**

The following statement returns the current time and date to the CURRENT-TIME and CURRENT-DATE fields:

```cobol
GET TIME
  INTO CURRENT-TIME EDIT
DATE INTO CURRENT-DATE.
```

**Status Codes**

After completion of the GET TIME function, the only possible value in the ERROR-STATUS field of the IDMS-DC communications block is 0000.

**IF (COBOL)**

The IF statement tests for the presence of member record occurrences in a set and determines the membership status of a record occurrence in a specified set. After the set has been evaluated, the IF statement specifies further action based on the outcome of the evaluation. For example, an IF statement can determine whether a set occurrence is empty and, if it is empty, erase the owner record.

Depending on its format, the IF statement uses set or run-unit currency. The owner of the current record of the named set determines the object set occurrence of an IF statement. The current of run unit determines the object record occurrence.

Each IF statement contains a conditional phrase and an imperative statement. When an IF is issued, the precompiler generates a call to the DBMS to execute the conditional phrase. Then, the precompiler generates a COBOL IF statement that tests the results of the DBMS call to determine whether to execute the imperative statement.

**Important!** Exercise care when nesting DML IF within COBOL IF statements because the logic can be difficult to follow. You may need to code explicit scope terminators.
Note: If AUTOSTATUS is in use (see AUTOSTATUS Protocols (see page)), IDMS-STATUS is not performed automatically when an IF statement is issued.

Native VSAM users: The IF statement is invalid for sets having member records that are stored in native VSAM data sets.

Syntax

```plaintext
IF set-name IS NOT EMPTY imperative-statement .
IF NOT set-name MEMBER
```

Parameters

- **set-name**
  Specifies which set owner to examine for the presence of member record occurrences. The specified set must be included in the subschema.

- **EMPTY**
  Executes the imperative statement when the current occurrence of the named set is empty.

- **NOT EMPTY**
  Executes the imperative statement when the current occurrence of the named set is not empty.

- **MEMBER**
  Executes the imperative statement when the current record of the run unit is a member of any occurrence of the specified set.

- **NOT set-name MEMBER**
  Executes the imperative statement when the current record of the run unit is not a member of any occurrence of the specified set.

- **imperative-statement**
  Specifies the action to take when the specified condition is true.

Examples

The following examples illustrate the use of the IF statement.

- **Example 1**
  The following statement tests the COVERAGE-CLAIMS set for existing CLAIMS members. If no occurrences of the CLAIMS record are found (ERROR-STATUS is 0000), a message to that effect is moved to location CLAIMS-WS:

  ```plaintext
  IF COVERAGE-CLAIMS IS EMPTY MOVE 'NONE' TO CLAIMS-WS.
  ```
If the current occurrence of the COVERAGE-CLAIMS set contains one or more occurrences of the CLAIMS record (ERROR-STATUS is 1601), the MOVE statement is ignored and the next statement in the program is executed.

- **Example 2**
  The following statement verifies that the EMPLOYEE record that is current of run unit is not a member of the current occurrence of the OFFICE-EMPLOYEE set before executing code to connect the EMPLOYEE record to that set:

  ```cobol
  IF NOT OFFICE-EMPLOYEE MEMBER PERFORM LINK-SET.
  ```

  If the EMPLOYEE record is not a member of the OFFICE-EMPLOYEE set (ERROR-STATUS is 1601), the program performs the LINK-SET paragraph. If the EMPLOYEE record is already a member of the OFFICE-EMPLOYEE set (ERROR-STATUS is 0000), the PERFORM statement is ignored and the next statement in the program is executed.

### Status Codes

After completion of the IF function, the ERROR-STATUS field in the IDMS communications block indicates the outcome:

<table>
<thead>
<tr>
<th>Status code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>The set is empty or the record that is current of run unit is a member of the set.</td>
</tr>
<tr>
<td>1601</td>
<td>The set is not empty or the record that is current of run unit is not a member of the set.</td>
</tr>
<tr>
<td>1606</td>
<td>Currency has not been established for the named set.</td>
</tr>
<tr>
<td>1608</td>
<td>An invalid set name has been specified or the current record of run unit is not a member of the named set.</td>
</tr>
<tr>
<td>1613</td>
<td>A current record of run unit has not been established or has been nullified by a preceding ERASE statement.</td>
</tr>
</tbody>
</table>

### INQUIRE MAP (COBOL)

The INQUIRE MAP statement is used after a MAP IN request to accomplish one of the following actions:

- Move map-related information into variable storage (see page 176).
- Test for conditions relating to global map input operations (see page 178).
- Test specific map fields for the presence of the cursor (see page 179).
- Test for conditions relating to specific map fields (see page 180).

The following rules apply to INQUIRE MAP statements:

- If any of the test conditions are requested, INQUIRE MAP must specify a statement to execute when the condition is found to be true.
• An INQUIRE MAP statement can specify only one field-oriented inquiry. This inquiry can be specified alone or with a map-specific inquiry.

• A MAP IN request must be issued before INQUIRE MAP is used.

Move Map-Related Data

This version of the INQUIRE MAP statement moves one of the following map-related data items into variable storage:

• The attention ID (AID) key that was used

• The current cursor position (row and column)

• The entered length of a specific map input field

Syntax

```
INQUIRE MAP map-name

MOVE AID TO aid-indicator
CURSOR TO cursor-row cursor-column
IN LENGTH FOR field-name TO field-length
```

Parameters

• **INQUIRE MAP map-name**
  Specifies the map for which the inquiry is to be made. This map must be included in the MAP SECTION of the program.

• **MOVE**
  Moves screen-related information to program variable storage.

• **AID TO aid-indicator**
  Returns the attention ID to the specified location in variable storage. Specify the symbolic name of a 1-byte user-defined field that will be set to the 3270 AID character received in the last map input request. The following table lists the AID characters that are associated with each 3270-type control key.

  ![Note: The data dictionary includes a record that defines the AID character values as level-88 items to test for particular keyed input by including a COPY IDMS DC-AID-CONDITION-NAMES statement in the WORKING-STORAGE SECTION.]

• **CURSOR TO cursor-row cursor-column**
  Returns the cursor address from the last map input function to the specified location in program variable storage. Specify the symbolic names of user-defined PIC S9(4) COMP fields to which the row and column cursor address will be returned.
• **IN LENGTH FOR** field-name
  Returns the length in bytes of the data in the specified map field. Specify the name of the map field for which the length is being requested.

• **TO** field-length
  Specifies where to return the field length. Specify the symbolic name of a user-defined PIC S9(4) COMP field.

### Attention ID (AID) Key Values

<table>
<thead>
<tr>
<th>Key</th>
<th>AID Character</th>
<th>Key</th>
<th>AID Character</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENTER</td>
<td>'' (single quote)</td>
<td>PF14</td>
<td>'B'</td>
</tr>
<tr>
<td>CLEAR</td>
<td>'_' (underscore)</td>
<td>PF15</td>
<td>'C'</td>
</tr>
<tr>
<td>PF01</td>
<td>'1'</td>
<td>PF16</td>
<td>'D'</td>
</tr>
<tr>
<td>PF02</td>
<td>'2'</td>
<td>PF17</td>
<td>'E'</td>
</tr>
<tr>
<td>PF03</td>
<td>'3'</td>
<td>PF18</td>
<td>'F'</td>
</tr>
<tr>
<td>PF04</td>
<td>'4'</td>
<td>PF19</td>
<td>'G'</td>
</tr>
<tr>
<td>PF05</td>
<td>'5'</td>
<td>PF20</td>
<td>'H'</td>
</tr>
<tr>
<td>PF06</td>
<td>'6'</td>
<td>PF21</td>
<td>'I'</td>
</tr>
<tr>
<td>PF07</td>
<td>'7'</td>
<td>PF22</td>
<td>'J'</td>
</tr>
<tr>
<td>PF08</td>
<td>'8'</td>
<td>PF23</td>
<td>','</td>
</tr>
<tr>
<td>PF09</td>
<td>'9'</td>
<td>PF24</td>
<td>'&lt;'</td>
</tr>
<tr>
<td>PF10</td>
<td>':'</td>
<td>PA01</td>
<td>'%'</td>
</tr>
<tr>
<td>PF11</td>
<td>'#'</td>
<td>PA02</td>
<td>'&gt;'</td>
</tr>
<tr>
<td>PF12</td>
<td>'@'</td>
<td>PA03</td>
<td>','</td>
</tr>
<tr>
<td>PF13</td>
<td>'A'</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The following figure shows the code that is copied into the program as a result of the COPY IDMS DC-AID-CONDITION-NAMES specification:

```plaintext
*01 COPY IDMS DC-AID-CONDITION-NAMES.
  01 DC-AID-CONDITION-NAMES.
    03 DC-AID-IND-V PIC X.
      88 ENTER-HIT VALUE QUOTE.
      88 CLEAR-HIT VALUE ' '.
      88 PF01-HIT VALUE '1'.
      88 PF02-HIT VALUE '2'.
      88 PF03-HIT VALUE '3'.
      88 PF04-HIT VALUE '4'.
      88 PF05-HIT VALUE '5'.
      88 PF06-HIT VALUE '6'.
      88 PF07-HIT VALUE '7'.
      88 PF08-HIT VALUE '8'.
      88 PF09-HIT VALUE '9'.
      88 PF10-HIT VALUE ':'.
      88 PF11-HIT VALUE '#'.
      88 PF12-HIT VALUE '@'.
      88 PF13-HIT VALUE 'A'.
      88 PF14-HIT VALUE 'B'.
      88 PF15-HIT VALUE 'C'.
      88 PF16-HIT VALUE 'D'.
```
88 PF17-HIT VALUE 'E'.
88 PF18-HIT VALUE 'F'.
88 PF19-HIT VALUE 'G'.
88 PF20-HIT VALUE 'H'.
88 PF21-HIT VALUE 'I'.
88 PF22-HIT VALUE '·'.
88 PF23-HIT VALUE '.'.
88 PF24-HIT VALUE '<'.
88 PA01-HIT VALUE '%'.
88 PA02-HIT VALUE '>'.
88 PA03-HIT VALUE ',',
88 PEN-ATTN-SPACE-NUL VALUE '='.
88 PEN-ATTN VALUE QUOTE.

Example

The following example moves the 3270 AID character that was received in the last map input request to DC-AID-IND-V. If the AID character indicates that PF1 was pressed, the program performs a DC RETURN:

```plaintext
INQUIRE MAP EMPMAPLR
  MOVE AID TO DC-AID-IND-V.
  IF CLEAR-HIT
    DC RETURN.
```

Test for Global Map Input Conditions

This version of the INQUIRE MAP statement tests for one of the following global map input conditions:

- If the screen was not formatted before the input operation was performed
- If one or more input fields were truncated when transferred to variable storage data fields
- If one or more input fields were modified on the screen before being transferred
- If one or more fields that were modified on the screen are undefined in the map being used

Syntax

```
INQUIRE MAP map-name

IF INPUT UNFORMATTED
   THEN imperative-statement
IF INPUT TRUNCATED
   THEN imperative-statement
IF INPUT CHANGED
   THEN imperative-statement
IF INPUT EXTRANEOUS
   THEN imperative-statement
```

Parameters

- **INQUIRE MAP map-name**
  Specifies the map for which the inquiry is being made. The map must be included in the MAP SECTION of the program.

- **IF INPUT**
  Tests the outcome of the last map input request for conditions relating to the data input to the program.
- **UNFORMATTED**
  Tests whether the screen was formatted before the input operation.

- **TRUNCATED**
  Tests whether any map fields were truncated when they were transferred to variable-storage data fields.

- **CHANGED**
  Tests whether any map fields were mapped to variable-storage data fields when the map input operation was performed.

- **EXTRANEOUS**
  Tests whether the input data stream contained any data from a field that was not defined to the map. If this condition is true, the undefined data field is ignored.

- **THEN imperative-statement**
  Specifies the action to take when the test condition is true. Specify a COBOL statement, a DML statement, or a nested block of COBOL and/or DML statements.

**Example**

The following example tests to determine whether any fields in the EMPMAPLR map have been truncated. If fields were truncated, the DATA-TRUNC routine is performed:

```
INQUIRE MAP EMPMAPLR
  IF INPUT TRUNCATED
    THEN PERFORM DATA-TRUNC.
```

**Test for Cursor Position**

This version of the INQUIRE MAP statement tests a map field for the presence of the cursor.

**Syntax**

```
INQUIRE MAP map-name 
  IF CURSOR at DFLD field-name THEN imperative-statement .
```

**Parameters**

- **INQUIRE MAP map-name**
  Specifies the map for which the inquiry is being made. The map must be included in the MAP SECTION of the program.

- **IF CURSOR at DFLD field-name**
  Specifies the field to test. This syntax determines whether the cursor was in the named map field during the last map input operation.

- **THEN imperative-statement**
  Specifies the action to take when the test condition is true. Specify a COBOL statement, a DML statement, or a nested block of COBOL and/or DML statements.
Example

The following example tests for the presence of the cursor in the PASSED-DATA-01 data field. If the cursor is present, the CHECK-2 routine is performed:

INQUIRE MAP EMMAPLR
  IF CURSOR AT DFLD EMP-LAST-NAME-0415
  THEN PERFORM CHECK-2.

Test for Input Nonzero Status Conditions

This version of the INQUIRE MAP statement tests for the following input conditions relating to specific map fields:

- If map fields have been modified
- If map fields have been erased by operator action
- If map fields have been truncated
- If map fields are identical to map data that is currently in program variable storage
- If map fields are different from map data that is currently in program variable storage
- If the specified map fields are in error (the error flag has been set on for those fields)
- If the specified map fields are correct (the error flag has been set off)

Syntax

```
INQUIRE MAP map-name
  IF
    CURRENT
    ALL
    NONE
    ANY
    SOME
    ALL BUT CURRENT
    ALL
    NONE
    ANY
    SOME
    ALL BUT 
    DFLD field-name
    DFLD field-name
    DATA is 
      YES
      NO
      ERASE
      TRUNCATED
      IDENTICAL
      DIFFERENT 
    EDIT is 
      ERROR
      CORRECT 
    THEN imperative-statement
```
Parameters

- **INQUIRE MAP** `map-name`
  Specifies the map for which the inquiry is being made. The map must be included in the MAP SECTION of the program.

- **IF**
  Specifies the map fields to test.
  
  - **CURRENT**
    Applies the test only to the current field; that is, the map field that was referenced in the last MODIFY MAP or INQUIRE MAP statement that the program issued. If the last MODIFY MAP or INQUIRE MAP statement specified a field list, no currency exists.
  
  - **ALL**
    Specifies that the test is true when all map fields meet the specified condition.
  
  - **NONE**
    Specifies that the test is true when none of the map fields meet the specified condition.
  
  - **ANY**
    Specifies that the test is true when one or more map fields meet the specified condition.
  
  - **SOME**
    Specifies that the test is true if one or more but not all of the map fields meet the specified condition.
  
  - **ALL BUT CURRENT**
    Specifies that the test is true when all map fields except the current field meet the specified condition.

- **IF**
  Specifies the extent to which the condition applies to the map field.
  
  - **ALL**
    Specifies that the test is true when all of the named map fields meet the specified condition.
  
  - **NONE**
    Specifies that the test is true when none of the named map fields meet the specified condition.
  
  - **ANY**
    Specifies that the test is true when one or more of the named map fields meet the specified condition.
  
  - **SOME**
    Specifies that the test is true when one or more but not all of the named map fields meet the specified condition.
  
  - **ALL BUT**
    Specifies that the test is true when all map fields except for the named field meet the specified condition.
• **DFLD field-name**
  Specifies the individual map fields to which the test conditions apply. Specify a field within the named map. Separate multiple DFLD specifications by at least one blank.

• **DATA IS**
  Tests the input data in the specified map fields.

  • **YES**
    Determines whether the terminal operator entered data in the specified map fields.

  • **NO**
    Determines whether the terminal operator did not enter data in the specified map fields.

• **ERASE**
  Determines whether data has been erased from the specified map fields.

• **TRUNCATED**
  Determines whether data has been truncated in the specified map fields.

• **IDENTICAL**
  Tests whether input data is identical to map data that is currently in program variable storage. IDENTICAL is true in either of the following cases:

  • The modified data tag (MDT) for the field is off. On mapin, the MDT is usually off when the user did not type any characters in the field.

  • The MDT for the field is on, but each character that the user typed is identical (including capitalization) to the data in variable storage.

• **DIFFERENT**
  Tests whether input data is different from map data that is currently in program variable storage. DIFFERENT is true when the field MDT is on, and at least one input character differs from the data in variable storage.

• **EDIT IS**
  Tests for errors in the named map fields. When EDIT is specified, automatic editing must be enabled for the map and for each of the named map fields.

  • **ERROR**
    Determines whether the named map fields were found to be in error during automatic editing.

  • **CORRECT**
    Determines whether the named map fields were found to be correct during automatic editing.

• **THEN imperative-statement**
  Specifies the action to take when the test condition is true. Specify a single COBOL statement, a DML statement, or a nested block of COBOL and/or DML statements.
Examples

The following examples illustrate the use of the INQUIRE MAP statement.

- **Example 1: Test for Erroneous Data**
  The following example determines whether automatic editing has detected erroneous data in any field in the EMPMAPLR map. If errors were detected, the program modifies the map temporarily to display the erroneous fields with the bright and blinking attributes:

  INQUIRE MAP EMPMAPLR
  IF ANY EDIT IS ERROR
    THEN MODIFY MAP EMPMAPLR TEMPORARY
       FOR ALL ERROR FIELDS
       ATTRIBUTES BRIGHT BLINK.

- **Example 2: Test for Identical Data**
  Use an INQUIRE MAP statement to test whether the user has entered an employee ID number:

  INQUIRE MAP MAP01
  IF DFLD EMP-ID-0415 DATA IS IDENTICAL THEN
    PERFORM EMP-PROMPT-20
  ELSE
    PERFORM EMP-OBTAIN-20.

  If the IDENTICAL condition is true (the user does not specify a different ID number), the program displays the menu screen.
  If the IDENTICAL condition is false (the user specifies a different ID number), the program obtains the corresponding employee record from the database.

- **Example 3: Test for Changed Data**
  Use an INQUIRE MAP statement to test whether the user has entered a new department ID or department name. If the user has changed either value (DIFFERENT is true), the program branches to DEPTUP-30:

  INQUIRE MAP MAP02
  IF ANY DFLD DEPT-ID-0410
     DFLD DEPT-NAME-0410 DATA IS DIFFERENT
    THEN PERFORM DEPTUP-30.

Status Codes

After completion of the INQUIRE MAP function, the ERROR-STATUS field of the IDMS-DC communications block indicates the outcome:

<table>
<thead>
<tr>
<th>Status code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>The request has been serviced successfully.</td>
</tr>
<tr>
<td>4629</td>
<td>An invalid parameter has been passed from the program.</td>
</tr>
<tr>
<td>4641</td>
<td>The test condition has been found to be true. (This condition is tested for automatically by COBOL DML expansion statements.)</td>
</tr>
<tr>
<td>4644</td>
<td>The referenced map field is not in the specified map. A possible cause is a reference to an invalid map field subscript.</td>
</tr>
<tr>
<td>4656</td>
<td>The referenced map contains no data fields.</td>
</tr>
</tbody>
</table>
**KEEP CURRENT (COBOL)**

The KEEP CURRENT statement places an explicit shared or exclusive lock on a record that is current of run unit, record, set, or area. Locks that are placed on records through the KEEP CURRENT function are maintained for the duration of the database transaction, or until explicitly released by the COMMIT or FINISH statements.

**Syntax**

```
KEEP CURRENT  .
```

**Parameters**

- **EXCLUSIVE**
  Places an exclusive lock on the current record of run unit, record, set, or area. If you do not specify EXCLUSIVE, the record receives a shared lock by default.

- **record-name**
  Places the lock on the current record of the specified record type.

- **WITHIN set-name**
  Places the lock on the current record of the specified set.

- **WITHIN area-name**
  Places the lock on the current record of the specified area.

**Example**

The following example places a shared lock on the current EMPLOYEE record occurrence:

```
KEEP CURRENT EMPLOYEE.
```

**Status Codes**

After completion of the KEEP function, the ERROR-STATUS field in the IDMS communications block indicates the outcome:

<table>
<thead>
<tr>
<th>Status code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>The request has been serviced successfully.</td>
</tr>
<tr>
<td>0606</td>
<td>Currency has not been established for the named record, set, or area.</td>
</tr>
<tr>
<td>Status code</td>
<td>Meaning</td>
</tr>
<tr>
<td>-------------</td>
<td>---------</td>
</tr>
<tr>
<td>0608</td>
<td>The named record or set is not in the subschema, or the current record of run unit is not a member of the named set.</td>
</tr>
<tr>
<td>0610</td>
<td>The subschema of the program specifies an access restriction that prohibits execution of the KEEP function.</td>
</tr>
<tr>
<td>0623</td>
<td>The named area is not in the subschema.</td>
</tr>
<tr>
<td>0626</td>
<td>The record to be kept has been erased.</td>
</tr>
<tr>
<td>0629</td>
<td>A deadlock has occurred while attempting to set the lock.</td>
</tr>
</tbody>
</table>

**KEEP LONGTERM (COBOL)**

The KEEP LONGTERM statement establishes long-term record locks. This statement can also monitor access to records between tasks. Long-term database locks are used in pseudo-conversational transactions and can be shared or exclusive:

- Long-term *shared* locks allow other run units to access the locked record. However, the run units cannot update the record while the lock is maintained.

- Long-term *exclusive* locks prevent other run units from accessing the locked record. However, run units that execute on the logical terminal that is associated with the issuing task are not restricted from accessing the locked record. Therefore, subsequent tasks in a transaction can access the locked record and can complete the database processing that the transaction requires.

If a record has been locked with a KEEP LONGTERM or KEEP request, restrictions exist on the type of lock that other run units can place on that record. These restrictions are based on existing locks and whether the requesting run unit executes on the same logical terminal as the run unit that placed the original lock on the record. The following table illustrates these restrictions.

**Keep Long-Term Record Lock Options**

<table>
<thead>
<tr>
<th>Locks in effect</th>
<th>Locks that are allowed for other run units</th>
<th>Locks that are disallowed for other run units</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shared</td>
<td>Shared</td>
<td>Exclusive</td>
</tr>
<tr>
<td></td>
<td>Long term shared</td>
<td>Long term exclusive</td>
</tr>
<tr>
<td>Exclusive</td>
<td>None</td>
<td>Shared</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Exclusive</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Long term shared</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Long term exclusive</td>
</tr>
<tr>
<td>Long term shared</td>
<td>For all run units:</td>
<td>For run units on other terminals:</td>
</tr>
<tr>
<td></td>
<td>■ Shared</td>
<td>■ Exclusive</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Locks in effect

- Long term shared
- Long term exclusive

For run units on the same terminal:
- Exclusive
- Long term exclusive

Long term exclusive

For run units on the same terminal:
- Shared
- Exclusive
- Long term shared
- Long term exclusive

For run units on other terminals:
- Shared
- Exclusive
- Long term shared
- Long term exclusive

Tasks can monitor database activity that is associated with a specified record during a pseudo-
converse. The tasks can also place a long-term lock on the monitored record. A subsequent task can
make inquiries about that database activity for the record and can take the appropriate action.

CA IDMS maintains information about database activity by using five bit flags. Each flag is turned on
(binary 1) or off (binary 0). This information is returned to the program as a numeric value. The
following table shows the bit assignments, the corresponding numeric value that is returned to the
program, and a description of the associated database activity:

<table>
<thead>
<tr>
<th>Numeric Value</th>
<th>Bit Assignment</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>X'00000010</td>
<td>The record was physically deleted.</td>
</tr>
<tr>
<td>8</td>
<td>X'00000008</td>
<td>The record was logically deleted.</td>
</tr>
<tr>
<td>4</td>
<td>X'00000004</td>
<td>The record prefix was modified. That is, a set operation (for example, CONNECT or DISCONNECT) occurred involving the record.</td>
</tr>
<tr>
<td>2</td>
<td>X'00000002</td>
<td>The record data was modified.</td>
</tr>
<tr>
<td>1</td>
<td>X'00000001</td>
<td>The record was obtained.</td>
</tr>
</tbody>
</table>

To determine the actions that have occurred, compare the numeric value that was returned to the
program with an appropriate constant. For example:

- If the returned value is zero, no database activity occurred for the specified record.
- If the returned value is two, the record data was modified.
- If the returned value is two or greater, the record was altered in some way.
- If the returned value is eight or greater, the record was deleted.

The maximum possible value is 31, indicating that all the preceding actions occurred for the specified record.
You can monitor database activity across a pseudo-converse rather than set long-term locks. Monitoring does not restrict access to database records, sets, or areas by other run units. Monitoring enables a program to test a record for alterations that were made by other run units. The presence of long-term locks can prevent other run units from accessing locked records for an undesirable amount of time when the terminal operator fails to enter a response during a pseudo-converse. If long-term locks are used, you can release them at specified intervals.

⚠️ **Note:** For more information about the use of timeout intervals, see Configuring Timeouts (https://docops.ca.com/display/IDMS19/Configuring+Timeouts).

**Syntax**

```plaintext
KEEP LONGTERM ─┬─ ALL ─┬─ longterm-id ──┐
└─          ─┘
NOTIFY CURRENT ─┬─ record-name ─┬─ set-name ─┬─ area-name ──┐
│                        └─                  ─┘
lock-options TEST ─┬─ RETURN NOTIFICATION into location RELEASE
│                        └─                  ─┘
```

**Parameters**

- **ALL**
  Releases all long-term locks that are kept for the logical terminal that is associated with the current task. Use this parameter only with the RELEASE parameter.

- **longterm-id**
  Specifies the symbolic name of a user-defined field that contains the long-term ID, or the ID itself enclosed in quotation marks. This ID is used in subsequent references to the lock, when it is changed or released.

- **NOTIFY CURRENT**
  Monitors database activity that is associated with the current record. CA IDMS initializes a preallocated location in the program to contain information about database activity for the specified record.
- **record-name**
  Monitors database activity that is associated with the current occurrence of `record-name`.

- **set-name**
  Monitors database activity that is associated with the record current of `set-name`.

- **area-name**
  Monitors database activity that is associated with the record current of `area-name`.

- **TEST RETURN NOTIFICATION into location**
  Returns information about database activity that is associated with the `longterm-id` record to a previously allocated location in the program storage. Specify the symbolic name of a user-defined PIC S9(8) COMP (fullword) field that contains the WORKING-Storage or LINKAGE SECTION entry of the data area to receive the information. The TEST request must specify a long-term lock ID that matches a long-term lock ID from a previous KEEP LONGTERM NOTIFY CURRENT request.

- **RELEASE**
  Releases the long-term lock for the `longterm-id` record or for all record locks (ALL) that are owned by the logical terminal that is associated with the current task. RELEASE also releases the information that is associated with a previous KEEP LONGTERM NOTIFY request.

### Lock Options

- **SHARE**
  Applies a long-term shared lock to the specified record.

- **EXCLUSIVE**
  Applies a long-term exclusive lock to the specified record.

  - **CURRENT record-name**
    Applies the lock to the current occurrence of `record-name`.

  - **CURRENT set-name**
    Applies the lock to the record current of `set-name`.

  - **CURRENT area-name**
    Applies the lock to the record current of `area-name`.

- **UPGRADE**
  Upgrades a previous KEEP LONGTERM NOTIFY CURRENT request.

  - **SHARE**
    Places a shared long-term lock on the record.

  - **EXCLUSIVE**
    Places an exclusive long-term lock on the record.
RETURN NOTIFICATION into return-location
Returns information about database activity for the specified record. Specify the symbolic name of a user-defined PIC S9(8) COMP (fullword) field that contains the WORKING-STORAGE SECTION or LINKAGE SECTION entry of the data area to receive the information.

WAIT
Requests the issuing task to wait for an existing lock to be released. This value is the default. If the wait would cause a deadlock, the task is terminated abnormally.

NOWAIT
Requests the issuing task not to wait for an existing lock to be released.

NODEADLOCK
Requests the issuing task to wait for an existing lock to be released, unless to do so would cause a deadlock. If the wait would cause a deadlock, control is returned to the task.

Example

The following steps illustrate the use of the KEEP LONGTERM statement.

1. Begin monitoring database activities for the current occurrence of the EMPLOYEE record by coding the following statement:

```
KEEP LONGTERM KEEP-ID NOTIFY CURRENT EMPLOYEE.
```

2. Return the database activity statistics for the record identified by KEEP-ID into STAT-VALUE by coding the following statement:

```
KEEP LONGTERM KEEP-ID TEST RETURN NOTIFICATION INTO STAT-VALUE.
```

3. Depending on the value returned to STAT-VALUE, you may want to put a long-term shared lock on the EMPLOYEE record that is identified by KEEP-ID. Code the following statement:

```
KEEP LONGTERM KEEP-ID UPGRADE SHARE.
```

4. When processing completes, release all long-term locks by coding the following statement:

```
KEEP LONGTERM ALL RELEASE.
```

Status Codes

After completion of the KEEP LONGTERM function, the ERROR-STATUS field in the IDMS-DC communications block indicates the outcome:

<table>
<thead>
<tr>
<th>Status code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>The request has been serviced successfully.</td>
</tr>
<tr>
<td>5101</td>
<td>The NODEADLOCK option has been specified. However, to wait would cause a deadlock. Control has returned to the issuing task.</td>
</tr>
<tr>
<td>Status code</td>
<td>Meaning</td>
</tr>
<tr>
<td>-------------</td>
<td>---------</td>
</tr>
<tr>
<td>5102</td>
<td>Unable to obtain storage for the required KEEP LONGTERM control blocks.</td>
</tr>
<tr>
<td>5105</td>
<td>The requested record type cannot be found or currency has not been established.</td>
</tr>
<tr>
<td>5113</td>
<td>The required area control block was not found in the DMCL.</td>
</tr>
<tr>
<td>5121</td>
<td>The requested long-term ID cannot be found or the KEEP LONGTERM request was issued by a nonterminal task.</td>
</tr>
<tr>
<td>5123</td>
<td>The specified area cannot be found.</td>
</tr>
<tr>
<td>5131</td>
<td>The parameter list is invalid.</td>
</tr>
<tr>
<td>5147</td>
<td>The KEEP LONGTERM area has not been readied.</td>
</tr>
<tr>
<td>5148</td>
<td>The run unit that is associated with the KEEP LONGTERM request has not been bound.</td>
</tr>
<tr>
<td>5149</td>
<td>The NOWAIT option has been specified. However, a wait is required.</td>
</tr>
<tr>
<td>5151</td>
<td>A lock manager error occurred during the processing of the KEEP LONGTERM request.</td>
</tr>
<tr>
<td>5159</td>
<td>An error occurred when transferring the KEEP LONGTERM request to IDMSKEEP.</td>
</tr>
<tr>
<td>5160</td>
<td>The requested KEEP LONGTERM lock ID was already in use with a different page group.</td>
</tr>
<tr>
<td>5161</td>
<td>The requested KEEP LONGTERM lock ID was already in use with a different database key format.</td>
</tr>
</tbody>
</table>

**LOAD TABLE (COBOL)**

The LOAD TABLE statement loads a table (module or program) into the program pool and provides access to it through a COBOL LINKAGE SECTION entry.

**Syntax**

```
LOAD TABLE program INTO 01-level-program-location TO end-program-location POINTER table-location-pointer

DICTNODE nodename DICTNAME dictionary-name

LOADLIB library-name

WAIT NOWAIT .
```

**Parameters**

- **LOAD TABLE program**
  Specifies the symbolic name of a user-defined field that contains the table, or the name itself enclosed in quotation marks.
• INTO 01-level-program-location
  Specifies the LINKAGE SECTION entry of the 01-level record area that references the loaded table. Specify the symbolic name of a user-defined field that contains the name of the 01-level LINKAGE SECTION entry that was used to load the table.

  ! Note: You cannot specify an OCCURS DEPENDING ON clause within 01-level-program-location.

• TO end-program-location
  Specifies the end of the LINKAGE SECTION entry of the 01-level record area that references the loaded table. Specify the symbolic name of a user-defined dummy byte field or a field containing a data item that is not associated with the module being loaded. End-program-location is subordinate to the 01-level record.

  ! This parameter is optional under COBOL 85.

• POINTER table-location-pointer
  Specifies a pointer to the table address (COBOL 85 only). Specify the symbolic name of a user-defined field that is to contain the pointer to the table address.

• DICTNODE nodename
  Specifies the node that controls the dictionary where the table resides. Specify the symbolic name of a user-defined eight-character field in variable storage, or the node name itself enclosed in quotation marks.

• DICTNAME dictionary-name
  Specifies the dictionary where the table resides. Specify the symbolic name of a user-defined eight-character field in variable storage, or the dictionary name itself enclosed in quotation marks.

• LOADLIB library-name
  Specifies the load library containing the table. Specify the symbolic name of a user-defined eight-character field in variable storage, or the library name itself enclosed in quotation marks.

• WAIT
  Requests the issuing task to wait for sufficient storage when program pool storage is not immediately available to meet the requirements of the LOAD TABLE request. This value is the default.
  If you specify WAIT and an insufficient storage condition exists, the issuing task is placed in an inactive state. When the LOAD TABLE function is completed, control returns to the issuing task according to its previously established dispatching priority.

• NOWAIT
  Requests the issuing task not to wait for storage to become available. If you specify NOWAIT and an insufficient storage condition exists, a value of 3402 (DC-NO-STORAGE) is returned to the ERROR-STATUS field.
Examples

The following source code defines the 01-level LINKAGE SECTION entry for use with the LOAD TABLE request for a table that was built from an Assembler program:

```
LINKAGE SECTION.
01 STATE-TABLE.
  02 STATES OCCURS 50 TIMES.
    03 STATE-ABB PIC X(2).
    03 STATE-FULL PIC X(15).
  02 END-STATE-TABLE PIC X.
```

⚠️ **Note:** IDD edit and code tables contain special characters and variable-length fields. In general, these fields are not used in a COBOL program.

The following examples illustrate the use of the LOAD TABLE statement:

- **Example 1**
  The following statement loads the STATECON table into the 01-level LINKAGE SECTION entry STATE-TABLE:

    ```
    LOAD TABLE 'STATECON'
      INTO STATE-TABLE TO END-STATE-TABLE.
    ```

- **Example 2**
  The following example defines the 01-level LINKAGE SECTION entry for use with the LOAD TABLE request for an IDD CODE TABLE, defined as follows:

    ```
    ADD TABLE NAME IS DECODMTH
    TABLE DESCRIPTION IS 'MONTH CODE CONVERT'
    TYPE IS CODE
    SEARCH IS LINEAR
    ENCLOSE DATA IS ALPHANUMERICPIC 9(4) COMP.
    TABLE IS UNSORTED
    DUPLICATES ARE NOT ALLOWED
    VALUES ARE ( 01 JAN 02 FEB 03 MAR 04 APR
       05 MAY 06 JUN 07 JUL 08 AUG
       09 SEP 10 OCT 11 NOV 12 DEC).
    ```

  The following source code defines the 01-level LINKAGE SECTION entry for use with the LOAD TABLE request:

  ```
  LINKAGE SECTION.
  01 MONTH-TABLE.
    02 TABLE-HEADER.
      03 HDR-ENTRIES PIC 9(4) COMP.
    02 TABLE-DATA.
      OCCURS 12 TIMES.
      03 DTA-FILLER1 PIC X(2).
      03 DTA-MONTH-NUM PIC 9(2).
      03 DTA-FILLER2 PIC X.
      03 DTA-MONTH-TXT PIC X(3).
    02 END-MONTH-TABLE PIC X.
  ```

  The following statement loads the DECODMTH tables into the 01-level LINKAGE SECTION entry MONTH-TABLE:
PROCEDURE DIVISION USING MONTH-TABLE.

LOAD TABLE 'DECODMTH'
    INTO MONTH-TABLE TO END-MONTH-TABLE.

Notes: The following requirements apply to BS2000:
- Starting with COBOL85 compiler V2.2C and higher, each 01-level entry in the LINKAGE
  SECTION must be defined in the USING clause of the PROCEDURE DIVISION.
- The TABLE definition must be the last definition in the LINKAGE SECTION.

Status Codes

After completion of the LOAD TABLE function, the ERROR-STATUS field in the CA IDMS/DC
communications block indicates the outcome:

<table>
<thead>
<tr>
<th>Status code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>The request has been serviced successfully.</td>
</tr>
</tbody>
</table>
| 3401        | The requested module cannot be loaded immediately due to insufficient storage. To wait
             | would cause a deadlock. |
| 3402        | The requested module cannot be loaded because insufficient storage exists in the program
             | pool. |
| 3407        | The requested module cannot be loaded because an I/O error has occurred during
             | processing. |
| 3414        | The requested module cannot be loaded because it has been defined as nonconcurrent and is in use. |
| 3415        | The requested module has been overlaid temporarily in the program pool and cannot be
             | reloaded immediately. |
| 3435        | The request cannot be serviced because the specified 01-level LINKAGE SECTION entry has been previously allocated or contains an OCCURS DEPENDING ON clause. |
| 3436        | The requested program is not defined in the program definition table (PDT) or is marked out of service, or null PDEs are not specified or valid in this CA IDMS/DC system. |

MAP IN (COBOL)

The MAP IN statement requests a synchronous transfer of data from map fields on the screen to the corresponding variable-storage data fields. The MAP IN statement can also transfer data from an area in variable storage that contains a 3270-like data stream to map-related variable-storage data fields; this transfer is referred to as a native mode data transfer.
Syntax

MAP IN USING map-name

IO

NOIO DATASTREAM FROM mapped-data-location TO end-data-location

LENGTH data-length

INPUT DATA is YES/NO

DETAIL detail-options

HEADER

Expansion of detail-options

NEXT

FIRST

SEQUENCE NUMBER is seq-field-name

KEY is key

RETURNKEY is data-field-name

Parameters

- **MAP IN USING map-name**
  Specifies the map to use for the MAP IN request. The map must be included in the MAP SECTION of the program.

- **IO**
  Transfers data from map fields to variable-storage data fields that are associated with the specified map. This type of data transfer is the default.

- **NOIO DATASTREAM FROM mapped-data-location**
  Transfers data from an area in program variable storage to the variable-storage data fields that correspond to the specified map. No terminal I/O is associated with the request. Specify the symbolic name of a user-defined field that contains the WORKING-STORAGE SECTION or LINKAGE SECTION entry of the data stream to read. The length of the data stream is determined by one of the following specifications:

  - **TO end-data-location**
    Indicates the end of the WORKING-STORAGE SECTION or LINKAGE SECTION entry that contains the data stream. Specify the symbolic name of a user-defined dummy byte field or a field containing a data item that is not associated with the input data stream.

  - **LENGTH data-length**
    Defines the explicit length of the input data stream, in bytes. Specify the symbolic name of a user-defined field that contains the length of the data stream, or the length itself expressed as a numeric constant.

- **INPUT DATA is**
  Specifies whether to move the contents of all fields of the specified map to variable-storage data fields, or to leave them unchanged. This option applies only to I/O requests. This specification applies to all variable-storage data fields unless it is overridden by an INPUT DATA IS YES/NO clause in a previously issued MODIFY MAP request.

  - **YES**
    Moves the contents of all fields of the specified map to variable-storage data fields.
- **NO**
  Leaves the contents of all variable-storage data fields unchanged.

- **DETAIL**
  Pageable maps only. Specifies that the MAP IN operation is to retrieve data from a modified detail occurrence (MDT set on). The contents of all map fields in the detail occurrence are retrieved unless MODIFIED is specified for the MAP IN DETAIL statement; MODIFIED causes only modified fields to be retrieved.

  **Note:** For more information about pageable maps, see [Mapping Facility](https://docops.ca.com/display/IDMSCU19/Mapping+Facility).

- **NEXT**
  Retrieves the next sequential modified detail occurrence. This value is the default.
  An end-of-data condition (DC-NO-MORE-UPD-DETAILS) is returned in either of the following cases:

  - No detail occurrences have been modified.
  - All modified detail occurrences have been mapped in already.

- **FIRST**
  Retrieves the first available modified detail occurrence.

- **SEQUENCE-NUMBER** is `seq-field-name`
  Retrieves a detail occurrence by sequence number. Detail occurrences are built at run time by the application program and stored in the sequence in which they are created. Specify a PIC S9(8) COMP (fullword) field.
  A detail-not-found condition is returned in either of the following cases:

  - The specified occurrence is not a modified detail occurrence.
  - No detail occurrence with the specified value is found.

- **KEY IS** `key`
  Retrieves a modified detail occurrence that is based on the value that is associated with the detail occurrence. Specify the name of a PIC S9(8) COMP (fullword) field.

  **Note:** A value is associated with a detail occurrence by using the KEY IS parameter in the MAP OUT DETAIL command for that occurrence.

- **RETURNKEY IS** `data-field-name`
  Specifies the variable field in which CA IDMS stores the 4-byte value (if any) that is associated with the retrieved detail occurrence. If no value is associated with the detail occurrence, the `data-field-name` is set to zero. Specify the symbolic name of a PIC X(4) or PIC S9(8) COMP (fullword) field that contains the key value. `Data-field-name` does not have to be fullword-aligned.
HEADER
Retrieves the contents of data fields in the header and footer areas. The contents of all data fields in the header and footer areas are retrieved unless MODIFIED is specified for the MAP IN HEADER statement. MODIFIED causes only modified fields to be retrieved. This option applies to pageable maps only.

PAGE IS page-number
Specifies the variable field in which to store the current value of the $PAGE field on mapin. Specify a PIC S9(8) COMP (fullword) field. This option applies to pageable maps only.

MODIFIED
Retrieves only modified fields (MDT set on) within a modified detail occurrence. This option applies to pageable maps only.

Examples
The following examples illustrate the use of the MAP IN statement.

Example 1
The following statement reads the EMPMAPLR map. Data values are transferred from map fields on the EMPMAPLR map to the corresponding variable-storage data fields. Subsequent commands can evaluate the input values and can perform appropriate processing:

```
MAP IN USING EMPMAPLR
   INPUT DATA IS YES.
```

Example 2
The following statement illustrates a request to map in the next modified detail occurrence of the EMPMAPPG MAP:

```
MAP IN USING EMPMAPPG
   DETAIL
   NEXT MODIFIED.
```

Status Codes
After completion of the MAP IN function, the ERROR-STATUS field of the IDMS-DC communications block indicates the outcome:

<table>
<thead>
<tr>
<th>Status code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>The request has been serviced successfully.</td>
</tr>
<tr>
<td>4627</td>
<td>A permanent I/O error has occurred during processing.</td>
</tr>
<tr>
<td>4628</td>
<td>The dial-up line for the terminal has been disconnected.</td>
</tr>
<tr>
<td>4631</td>
<td>The map request block (MRB) contains an invalid field, indicating a possible error in the program parameters.</td>
</tr>
<tr>
<td>4632</td>
<td>The derived length of the specified map input data area is zero or negative.</td>
</tr>
<tr>
<td>4633</td>
<td>The map load module that is named in the MRB cannot be found.</td>
</tr>
<tr>
<td>Status code</td>
<td>Meaning</td>
</tr>
<tr>
<td>-------------</td>
<td>---------</td>
</tr>
<tr>
<td>4634</td>
<td>The specified LINKAGE SECTION entry is not at COBOL 01-level.</td>
</tr>
<tr>
<td>4638</td>
<td>The specified 01-level WORKING-STORAGE SECTION or LINKAGE SECTION entry has not been allocated.</td>
</tr>
<tr>
<td>4639</td>
<td>The terminal being used is out of service.</td>
</tr>
<tr>
<td>4640</td>
<td>The NOIO option has been specified but the requested data stream cannot be found.</td>
</tr>
<tr>
<td>4642</td>
<td>The requested map does not support the terminal device being used.</td>
</tr>
<tr>
<td>4652</td>
<td>The specified edit or code table cannot be found or is invalid for use with the named map.</td>
</tr>
<tr>
<td>4654</td>
<td>A data conversion error has occurred. Internal map data does not match the data description of the map.</td>
</tr>
<tr>
<td>4655</td>
<td>The user-written edit routine that was specified for the named map cannot be found.</td>
</tr>
<tr>
<td>4664</td>
<td>The requested node for a header or detail was not present or not updated.</td>
</tr>
<tr>
<td>4668</td>
<td>No more modified detail occurrences require mapin.</td>
</tr>
<tr>
<td>4672</td>
<td>The scratch record that contains the requested detail could not be accessed (internal error).</td>
</tr>
</tbody>
</table>

**MAP OUT (COBOL)**

The MAP OUT statement creates or modifies detail occurrences for a pageable map or requests a data transfer from variable-storage data fields to map fields on the terminal screen. MAP OUT can also transfer data to another area in program variable storage; this transfer is referred to as a native mode data transfer.

**Syntax**

```
MAP OUT USING map-name
  WAIT ← NOWAIT

  IO ←
    NOIO DATASTREAM INTO data-location TO end-data-location
    max LENGTH data-length

  RETURN LENGTH into data-actual-length

  OUTPUT DATA is YES NO ERASE ATTRIBUTE
    NEWPAGE ERASE LITERALS

  MESSAGE IS message-text TO end-message-data-location
    LENGTH data-length
```
Parameters

- **MAP OUT USING map-name**
  Specifies the map to use for the MAP OUT request. The map must be included in the MAP SECTION of the program.

- **WAIT**
  Specifies a synchronous data transfer. The issuing task is placed in an inactive state. When the MAP OUT operation is complete, the task resumes processing according to its established dispatching priority. This value is the default.

- **NOWAIT**
  Specifies an asynchronous data transfer. The task continues executing. If NOWAIT is specified, the program must issue a CHECK TERMINAL before performing any other I/O operation.

- **IO**
  Transfers data from variable-storage data fields that are associated with the named map to the terminal device that is associated with the issuing task. This value is the default.

- **NOIO DATASTREAM INTO data-location**
  Transfers data from variable-storage data fields that are associated with the named map to another area of program variable storage. No terminal I/O is associated with the request. Specify the symbolic name of a user-defined field containing the WORKING-STORAGE SECTION or LINKAGE SECTION entry to which the data is to be transferred.

- **TO end-data-location**
  Indicates the end of the WORKING-STORAGE SECTION or LINKAGE SECTION entry for the output data stream and is specified following the last data-item entry in data-location. Specify the symbolic name of a user-defined dummy byte field or a field containing a data item that is not associated with the output data stream.

- **max LENGTH data-length**
  Defines the maximum length of the output data stream. Specify the symbolic name of the user-defined field that contains the length of the data stream, or the length itself expressed as a numeric constant.

- **RETURN LENGTH INTO data-actual-length**
  Specifies the WORKING-STORAGE SECTION or LINKAGE SECTION entry to which CA IDMS returns the length in bytes of the output data stream. If the data stream has been truncated, this entry contains the length before truncation.

- **OUTPUT**
  Specifies screen display options for the data being output. This option applies only to IO requests.
DATA IS
Specifies whether to transmit the variable-storage data fields to the terminal. This specification applies to all variable-storage data fields unless it is overridden by an OUTPUT DATA IS YES/NO clause in a previously issued MODIFY MAP request.

- **YES**
  Transmits the contents of variable-storage data fields to the corresponding map fields.

- **NO**
  Does not transmit the contents of variable-storage data fields to the corresponding map fields. However, if the automatic error handling facility detects an error in any field, the applicable attribute bytes are transmitted.

- **ERASE**
  Does not transmit the contents of variable-storage data fields and fills the corresponding map fields with null values.

- **ATTRIBUTE**
  Transmits only the attribute bytes for variable-storage data fields. Data in the record buffer is not sent to the terminal.

NEWPAGE|ERASE
Activates the erase-write function. The screen is cleared and the literal and variable fields are transmitted to the map. If NEWPAGE or ERASE is not specified, any existing screen display is overwritten without first erasing it. (NEWPAGE and ERASE are synonymous.)
To erase individual map fields, use the OUTPUT DATA IS ERASE option of the MODIFY MAP statement. To erase all screen fields and to activate the erase-write function, the MAP OUT statement must specify OUTPUT DATA IS ERASE NEWPAGE.

LITERALS
Transmits literal fields and variable-storage data fields to the terminal. If LITERALS is not specified, literal fields are written to the map only when a MAP OUT request specifies the ERASE option.

MESSAGE IS message-text
Specifies the message to display in the map message area. Specify the symbolic name of a WORKING-STORAGE SECTION or LINKAGE SECTION entry that contains the message text. This option applies only to IO requests.

Notes:
- The MESSAGE parameter can only be used with MAP OUT DETAIL when the $MESSAGE field is associated with the detail occurrence at map generation.
- To reference a message that is stored in the data dictionary, use the ACCEPT TEXT INTO parameter of the WRITE LOG statement to copy the message into message-text.
- **TO end-message-data-location**
  Specifies the end of the WORKING-STORAGE SECTION or LINKAGE SECTION entry that contains the message text and is specified following the last data item in message-text. Specify the symbolic name of a user-defined dummy byte field or a field containing a data item that is not associated with the output data stream.

- **LENGTH data-length**
  Defines the length of the message text in bytes. Specify the symbolic name of a user-defined field that contains the length, or the length itself expressed as a numeric constant.

- **DETAIL**
  Creates or modifies a detail occurrence, and optionally associates a numeric key value with the occurrence. This option applies only to pageable maps.

- **NEW**
  Creates a detail occurrence of a pageable map. This value is the default. Occurrences are displayed in the order in which they are created by the application program.

- **CURRENT**
  Modifies the detail occurrence that was referenced by the most recent MAP IN DETAIL or MAP OUT DETAIL statement.

- **KEY IS key**
  Specifies a value to associate with the created or modified detail occurrence. Specify the name of a PIC S9(8) COMP (fullword) field containing the key of a database record that is associated with the detail occurrence.
  The 4-byte numeric value is not displayed on the terminal screen. When the KEY IS parameter is used with the MAP OUT DETAIL CURRENT command, the specified value replaces the value (if any) that was previously associated with the detail occurrence.

- **RESUME**
  Specifies the page of detail occurrences to map out to the terminal. This option applies only to pageable maps.
  - **PAGE is CURRENT**
    Redisplays the current page. This value is the default. If no page has been displayed, the first page of the pageable map is displayed.
  - **PAGE is NEXT**
    Displays the page that follows the current page. If no page follows the current page, the current page is redisplayed.
  - **PAGE is PRIOR**
    Displays the page that precedes the current page. If no page precedes the current page, the current page is redisplayed.
  - **PAGE is FIRST**
    Displays the first available page of detail occurrences.
  - **PAGE is LAST**
    Displays the page of detail occurrences with the highest available page number.
- **PAGE is page-number**
  Specifies the number of the page to display. A page number is stored in the variable field by a preceding MAP IN PAGE IS page-number statement that names the same numeric variable field.

### Examples

The following examples illustrate the use of the MAP OUT statement:

- **Example 1**
  The following statement writes all literal and data fields that are associated with the EMPMAPLR map to the terminal:

    ```cobol
    MAP OUT USING EMPMAPLR
    OUTPUT DATA IS YES
    NEWPAGE
    MESSAGE IS INITIAL-MESSAGE LENGTH 80.
    ```

- **Example 2**
  The following statement maps out the current detail. If the first page of the pageable map is not yet filled, no terminal I/O is associated with this request:

    ```cobol
    MAP OUT USING EMPMAPPG
    DETAIL
    KEY IS DETAIL-KEY.
    ```

### Status Codes

After completion of the MAP OUT function, the ERROR-STATUS field in the IDMS-DC communications block indicates the outcome:

<table>
<thead>
<tr>
<th>Status code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>The request has been serviced successfully.</td>
</tr>
<tr>
<td>4625</td>
<td>The output operation has been interrupted. The operator has pressed ATTENTION or BREAK.</td>
</tr>
<tr>
<td>4626</td>
<td>A logical error (for example, an invalid control character) has been encountered in the output data stream.</td>
</tr>
<tr>
<td>4627</td>
<td>A permanent I/O error has occurred during processing.</td>
</tr>
<tr>
<td>4628</td>
<td>The dial-up line for the terminal has been disconnected.</td>
</tr>
<tr>
<td>4631</td>
<td>The map request block (MRB) contains an invalid field, indicating a possible error in the program parameters.</td>
</tr>
<tr>
<td>4632</td>
<td>The derived length of the specified map output data area is zero or negative.</td>
</tr>
<tr>
<td>4633</td>
<td>The map load module that was named in the MRB cannot be found.</td>
</tr>
<tr>
<td>4634</td>
<td>The specified LINKAGE SECTION entry is not at COBOL 01-level.</td>
</tr>
<tr>
<td>4638</td>
<td>The WORKING-STORAGE SECTION or LINKAGE SECTION entry that was specified for return of the output data stream has not been allocated.</td>
</tr>
<tr>
<td>4639</td>
<td>The terminal being used is out of service.</td>
</tr>
</tbody>
</table>
### Status Code and Meaning

<table>
<thead>
<tr>
<th>Status Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>4640</td>
<td>The NOIO option has been specified but the requested data stream cannot be found.</td>
</tr>
<tr>
<td>4642</td>
<td>The requested map does not support the terminal device being used.</td>
</tr>
<tr>
<td>4652</td>
<td>The specified edit or code table cannot be found or is invalid for use with the named map.</td>
</tr>
<tr>
<td>4653</td>
<td>An error has occurred in a user-written edit routine.</td>
</tr>
<tr>
<td>4654</td>
<td>A data conversion error has occurred. The internal map data does not match the data description of the map.</td>
</tr>
<tr>
<td>4655</td>
<td>The user-written edit routine that was specified for the named map cannot be found.</td>
</tr>
<tr>
<td>4664</td>
<td>There is no current detail occurrence to be updated (MAP OUT DETAIL CURRENT only). No action is taken.</td>
</tr>
<tr>
<td>4668</td>
<td>The amount of storage that was defined for pageable maps at system generation time is insufficient. No action is taken. This statement and subsequent MAP OUT DETAIL statements are ignored.</td>
</tr>
<tr>
<td>4672</td>
<td>No detail occurrence, footer, or header fields exist to be mapped out by a MAPOUT RESUME command.</td>
</tr>
<tr>
<td>4676</td>
<td>The first screen page has been transmitted to the terminal.</td>
</tr>
<tr>
<td>4680</td>
<td>A pageable map page has been built but the page has not been displayed. This outcome can happen after you specify STARTPAGE NOAUTODISPLAY. Test for it after each MAP OUT DETAIL statement.</td>
</tr>
</tbody>
</table>

### MAP OUTIN (COBOL)

The MAP OUTIN statement requests an output data transfer (MAP OUT) followed by an input data transfer (MAP IN). MAP OUTIN combines the functions of the MAP OUT and MAP IN requests. However, MAP OUTIN cannot perform pageable map functions or native mode data transfers. By definition, the MAP OUTIN request is synchronous; it forces the program to be conversational.

### Syntax

```cobol
MAP OUTIN USING map-name
  OUTPUT DATA is
    YES NO ERASE LITERALS
  INPUT DATA is
    YES NO
  MESSAGE IS message-text TO end-message-data-location
    LENGTH data-length
```

---

15-Jan-2018 202/415
Parameters

- **MAP OUTIN map-name**
  Specifies the map to use for the MAP OUTIN request. The map must be included in the MAP SECTION of the program.

- **OUTPUT**
  Specifies screen display options for the data being output.

- **DATA is**
  Specifies whether to transmit variable-storage data fields to the terminal. This specification applies to all variable-storage data fields unless it is overridden by an OUTPUT DATA IS YES/NO clause in a previously issued MODIFY MAP request.
  - **YES**
    Transmits the contents of variable-storage data fields to the corresponding map fields.
  - **NO**
    Does not transmit the contents of variable-storage data fields to the corresponding map fields. However, if the automatic error handling facility detects an error in any field, the applicable attribute bytes are transmitted.
  - **ERASE**
    Does not transmit the contents of variable-storage data fields and fills the corresponding map fields with null values.
  - **ATTRIBUTE**
    Transmits only the attribute bytes for variable-storage data fields. Data in the record buffer is not sent to the terminal.

- **NEWPAGE | ERASE**
  Activates the erase-write function. The screen is cleared and literal and variable fields are transmitted to the map. If NEWPAGE or ERASE is not specified, any existing screen display is overwritten without first erasing it. (NEWPAGE and ERASE are synonymous.)
  To erase individual map fields, use the OUTPUT DATA IS ERASE option of the MODIFY MAP statement. To erase all screen fields and to activate the erase-write function, the MAP OUT statement must specify OUTPUT DATA IS ERASE NEWPAGE.

- **LITERALS**
  Transmits literal fields and variable-storage data fields to the terminal. If LITERALS is not specified, literal fields are written to the map only when a MAP OUT request specifies the ERASE option.

- **INPUT DATA is**
  Specifies whether to move the contents of map fields to variable-storage data fields or leave them unchanged. This specification applies to all variable-storage data fields unless it is overridden by an INPUT DATA IS YES/NO clause in a previously issued MODIFY MAP request.
  - **YES**
    Moves the contents of map fields to variable-storage data fields.
- **NO**
  Leaves the contents of map fields unchanged.

- **MESSAGE IS message-text**
  Specifies the message to display in the map message area. Specify the symbolic name of a WORKING-STORAGE SECTION or LINKAGE SECTION entry that contains the message text.

- **TO end-message-data-location**
  Specifies the end of the WORKING-STORAGE SECTION or LINKAGE SECTION entry that contains the message text and is specified following the last data item in message-text. Specify the symbolic name of a user-defined dummy byte field or a field containing a data item that is not associated with the output data stream.

- **LENGTH data-length**
  Defines the length of the message text in bytes. Specify the symbolic name of a user-defined field that contains the length, or the length itself expressed as a numeric constant.

Note: To reference a message that is stored in the data dictionary, use the ACCEPT TEXT INTO parameter of the WRITE LOG statement to copy the message into message-text.

### Example

The following statement erases the screen, transmits literal and variable map fields (null values), and performs a MAP IN when the operator presses an AID key:

```
MAP OUTIN USING EMPMAPLR
  OUTPUT DATA IS ERASE NEWPAGE
  INPUT DATA IS YES.
```

### Status Codes

After completion of the MAP OUTIN function, the ERROR-STATUS field in the IDMS-DC communications block indicates the outcome:

<table>
<thead>
<tr>
<th>Status code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>The request has been serviced successfully.</td>
</tr>
<tr>
<td>4625</td>
<td>The I/O operation has been interrupted. The terminal operator has pressed ATTENTION or BREAK.</td>
</tr>
<tr>
<td>4626</td>
<td>A logical error (for example, an invalid control character) has been encountered in the output data stream.</td>
</tr>
<tr>
<td>4627</td>
<td>A permanent I/O error has occurred during processing.</td>
</tr>
<tr>
<td>4628</td>
<td>The dial-up line for the terminal is disconnected.</td>
</tr>
<tr>
<td>4631</td>
<td></td>
</tr>
<tr>
<td>Status code</td>
<td>Meaning</td>
</tr>
<tr>
<td>-------------</td>
<td>---------</td>
</tr>
<tr>
<td></td>
<td>The map request block (MRB) contains an invalid field, indicating a possible error in the program parameters.</td>
</tr>
<tr>
<td>4633</td>
<td>The map load module that was named in the MRB cannot be found.</td>
</tr>
<tr>
<td>4639</td>
<td>The terminal being used is out of service.</td>
</tr>
<tr>
<td>4642</td>
<td>The requested map does not support the terminal device being used.</td>
</tr>
<tr>
<td>4652</td>
<td>The specified edit or code table cannot be found or is invalid for use with the named map.</td>
</tr>
<tr>
<td>4653</td>
<td>An error has occurred in a user-written edit routine.</td>
</tr>
<tr>
<td>4654</td>
<td>A data conversion error has occurred. The internal map data does not match the data description of the map.</td>
</tr>
<tr>
<td>4655</td>
<td>The user-written edit routine that was specified for the named map cannot be found.</td>
</tr>
</tbody>
</table>

**MODIFY (COBOL)**

The MODIFY statement replaces element values of the specified record occurrence with new element values that are defined in program variable storage.

Before the MODIFY statement is executed, the following conditions must be satisfied:

- All areas that are affected implicitly or explicitly must be readied in one of the update usage modes (see READY (COBOL) (see page 228)).

- The specified record must be established as current of run unit. If the record that is current of run unit is not an occurrence of the specified record, a nonzero status condition results.

- The values of all elements that are defined for the specified record in the program subschema view must be in variable storage. If an OBTAIN statement does not precede the MODIFY statement, you must initialize the appropriate values. Precede MODIFY with an OBTAIN statement to ensure that all elements in the modified record are present in variable storage.

The following items apply to the modification of CALC- and sort-control elements:

- If modification of a CALC- or sort-control element violates a duplicates-not-allowed option, the record is not modified. A nonzero status condition occurs.

- If a CALC-control element is modified, successful MODIFY execution lets the record be accessed by its new CALC-key value. The database key of the specified record is not changed.

- If a sort-control element is modified, the sorted set in which the specified record participates must be included in the subschema that the program invokes. A record occurrence that is a member of a set which is not defined in the subschema can be modified only when the undefined set is not sorted.
If any modified elements in the specified record are defined as sort-control elements for any set occurrence in which that record is a member, the set occurrence is examined. If necessary, the specified record is disconnected and reconnected in the set occurrence to maintain the set order that is specified in the schema.

The following items apply to the modification of records in native VSAM data sets:

- The record length in an entry-sequenced data set (ESDS) cannot be changed even when the records are variable length.
- The prime key for a key-sequenced data set (KSDS) cannot be modified.

**Currency:** The specified record must be established as current of run unit.

After the MODIFY statement executes successfully, the modified record becomes the current record of run unit, its record type, its area, and all sets in which it participates as member or owner.

### Syntax

```
MODIFY record-name .
```

### Parameters

- **MODIFY record-name**
  
  Specifies the record type to update. The record type must be included in the subschema. The occurrence of `record-name` residing in program variable storage is updated.

### Example

The following example modifies an occurrence of the EMPLOYEE record to change the employee address.

1. Retrieve the desired EMPLOYEE record and move its contents to variable storage:

   ```
   MOVE EMP-ID-IN TO EMP-ID-0415.
   OBTAIN CALC EMPLOYEE.
   ```

2. Update the EMP-ADDRESS-0415 field by moving the new address into the proper location in the EMPLOYEE record:

   ```
   MOVE NEW-ADDRESS TO EMP-ADDRESS-0415.
   ```

3. Issue a MODIFY statement to return all data items in the EMPLOYEE record to the database:

   ```
   MODIFY EMPLOYEE.
   ```
### Status Codes

After completion of the MODIFY function, the ERROR-STATUS field in the IDMS communications block indicates the outcome:

<table>
<thead>
<tr>
<th>Status code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>The request has been serviced successfully.</td>
</tr>
<tr>
<td>0803</td>
<td>Invalid currency for a record to be altered by a MODIFY. This outcome occurs only when a run unit is sharing a transaction with other database sessions. The 03 minor status is returned when the run unit tries to modify a record by using a currency that has been invalidated because of changes that were made by another database session that is sharing the same transaction.</td>
</tr>
<tr>
<td>0804</td>
<td>The OCCURS DEPENDING ON item is less than zero or greater than the maximum number of occurrences of the control element.</td>
</tr>
<tr>
<td>0805</td>
<td>Modification of the record would violate a duplicates-not-allowed option for a CALC record, a sorted set, or an index set.</td>
</tr>
<tr>
<td>0806</td>
<td>Currency has not been established for the named record.</td>
</tr>
<tr>
<td>0808</td>
<td>The specified record cannot be found. The record name has probably been misspelled.</td>
</tr>
<tr>
<td>0809</td>
<td>The area of the named record has not been readied in one of the update usage modes.</td>
</tr>
<tr>
<td>0810</td>
<td>The subschema specifies an access restriction that prohibits modification of the named record.</td>
</tr>
<tr>
<td>0811</td>
<td>The space is insufficient to hold the modified variable-length record occurrence.</td>
</tr>
<tr>
<td>0813</td>
<td>A current record of run unit has not been established or has been nullified by a previous ERASE statement.</td>
</tr>
<tr>
<td>0818</td>
<td>The record has not been bound.</td>
</tr>
<tr>
<td>0820</td>
<td>The current record of run unit is not the same type as the named record.</td>
</tr>
<tr>
<td>0821</td>
<td>An area other than the area of the named record has been readied with an incorrect usage mode.</td>
</tr>
<tr>
<td>0825</td>
<td>No current record of set type has been established.</td>
</tr>
<tr>
<td>0833</td>
<td>At least one sorted set in which the named record participates has not been included in the subschema.</td>
</tr>
<tr>
<td>0855</td>
<td>An invalid length has been defined for a variable length record.</td>
</tr>
<tr>
<td>0860</td>
<td>A record occurrence has a type that is inconsistent with the set named in the ERROR-SET field of the IDMS communications block. Probable causes include a broken chain and improper database description.</td>
</tr>
<tr>
<td>0883</td>
<td>A record length in a native VSAM ESDS has been changed or a prime key in a native VSAM KSDS has been modified.</td>
</tr>
</tbody>
</table>
MODIFY (LRF) (COBOL)

The MODIFY statement changes the field values in an existing logical-record occurrence. LRF uses the field values in the variable storage location that are reserved for the logical record to update the appropriate database records. You can optionally specify an alternative variable storage location from which to take the changed field values.

Syntax

```cobol
MODIFY logical-record-name
FROM alt-logical-record-location
WHERE boolean-expression
ON path-status imperative-statement
```

Parameters

- **MODIFY logical-record-name**
  Updates the data field values in the named logical record. Unless you include the FROM clause, the field values that are used to update the database are taken from the area in program variable storage that is reserved for the named logical record. The logical record must be defined in the subschema.

- **FROM alt-logical-record-location**
  Specifies an alternative variable storage location from which to take the field values. This record location must be defined in the WORKING-STORAGE SECTION or LINKAGE SECTION. The FROM clause must name the same location that was specified in the OBTAIN request.

- **WHERE boolean-expression**
  Specifies the selection criteria to apply to the logical record. For details on coding this clause, see Logical-Record Clauses (COBOL) (see page 272).

- **ON**
  Specifies the action to take, depending on the value that is returned to the LR-STATUS field in the LRC block. For details on coding this clause, see Logical-Record Clauses (COBOL) (see page 272).

  - **path-status**
    Specifies the value of the LR-STATUS field in the LRC block that triggers the specified action.

  - **imperative-statement**
    Specifies the action to take.
Example

The following example modifies an occurrence of the EMP-SKILL-LR logical record. This example upgrades the skill level for employee 120 from 02 (COMPETENT-0425) to 03 (PROFICIENT-0425).

1. Retrieve the desired logical-record occurrence:
   
   OBTAIN FIRST EMP-SKILL-LR WHERE EMP-ID-0415 EQ '0120'
   AND SKILL-ID-0455 EQ '3610'
   AND COMPETENT-0425.

2. Update the SKILL-LEVEL-0425 field:

   MOVE '03' TO SKILL-LEVEL-0425.

3. Issue the MODIFY statement for the updated EMP-SKILL-LR logical record:

   MODIFY EMP-SKILL-LR.

MODIFY EMP-SKILL-LR

The following figure illustrates the previous example by showing three occurrences of the EMP-SKILL-LR logical record:

<table>
<thead>
<tr>
<th>EMPLOYEE</th>
<th>EXPERTISE</th>
<th>SKILL</th>
</tr>
</thead>
<tbody>
<tr>
<td>120</td>
<td>03</td>
<td>3710</td>
</tr>
<tr>
<td>120</td>
<td>(02) 03</td>
<td>3610</td>
</tr>
</tbody>
</table>
| LRF retrieves the EMP-SKILL-LR logical record where the following conditions are true:

- EMP-ID-0415 = '0120'
- SKILL-ID-0455 = '0120'
- SKILL-LEVEL-0425 = '02' (COMPETENT-0425)

The bottom EXPERTISE occurrence represents the only data that is physically modified in the database.

MODIFY MAP (COBOL)

The MODIFY MAP statement modifies the options in the map request block (MRB) for a map. Modifications can be designated as permanent or temporary. You can request modifications that are field-specific, map-specific, or both. Field-specific revisions apply to the variable data fields in the map.
Note: The MODIFY MAP statement parameters that revise predefined map or map data field attributes have no defaults. If a parameter is not specified, it remains set to the value that was specified at map generation or in a previous MODIFY MAP PERMANENT statement.
Parameters

- **MODIFY MAP map-name**
  Specifies the map to modify. The map must be included in the MAP SECTION of the program.

- **PERMANENT**
  Applies the modifications to all mapping mode I/O requests until the program terminates or until a subsequent MODIFY MAP request overrides the requested revisions. This value is the default.

- **TEMPORARY**
  Applies the modifications only to the next mapping mode I/O request (that is, MAP IN, MAP OUT, or MAP OUTIN).

- **CURSOR AT**
  Specifies where to position the cursor during output operations.

  - **cursor-row**
    Specifies a row on the terminal screen. Specify the symbolic name of the user-defined field that contains the row value, or the value itself expressed as a numeric constant. Typically, fields with cursor row and column coordinates are level-77 data items that are defined as PIC S9(4) USAGE COMP (halfword).
- **cursor-column**
  Specifies a column on the terminal screen. Specify the symbolic name of a user-defined field that contains the column value, or the value itself expressed as a numeric constant. Typically, fields with cursor row and column coordinates are level-77 data items that are defined as PIC 9(4) USAGE COMP (halfword).

- **DFLD field-name**
  Specifies a map field. The cursor is moved to the first position in the field.

- **WCC**
  Specifies the write-control character (WCC) options for the output operation. If a MODIFY MAP request alters any WCC option, unspecified options are reset to NOMDT, NOKBD, and NOALARM.

- **RESEMTMDT**
  Resets (turns off) the modified data tags (MDTs) for the map fields automatically when the map is displayed.

- **NOMDT**
  Does not reset (turn off) the modified data tags (MDTs) for the map fields when the map is displayed. The associated data is retransmitted to variable-storage data fields during the next MAP IN request.

- **RESETKBD**
  Unlocks the keyboard automatically when the map is displayed.

- **NOKBD**
  Does not unlock the keyboard when the map is displayed.

- **ALARM**
  Sounds the terminal audible alarm (if installed) automatically when the map is displayed.

- **NOALARM**
  Does not sound the terminal audible alarm when the map is displayed.

- **STARTPRT**
  Prints the contents of the terminal buffer automatically when the data has been transmitted to the terminal. This option applies only to 3280-type printers.

- **NOPRT**
  Does not print the contents of the terminal buffer when the data has been transmitted to the terminal. This option applies only to 3280-type printers.

- **NLCR**
  Does not perform line formatting on the printer output. Printing begins on a new line only when the printer encounters new line (NL) and carriage control (CR) characters.

- **40CR**
  Prints the contents of the 3280-type printer buffer at 40 characters per line.

- **64CR**
  Prints the contents of the 3280-type printer buffer at 64 characters per line.
- **80CR**
  Prints the contents of the 3280-type printer buffer at 80 characters per line.

- **FOR**
  Specifies the map fields to modify or to exclude from modification.

### Expansion of field-specifications

- **ALL BUT (EXCEPT) CURRENT**
  Modifies all fields except the current field. The current field is the map field that was referenced in the last MODIFY MAP or INQUIRE MAP request. If that request referenced a list of fields rather than a single map field, no currency exists and all map fields are modified. BUT and EXCEPT are synonyms and can be used interchangeably.

- **ALL CORRECT FIELDS**
  Modifies all fields that are found to be correct during automatic editing or by a user-written edit module.
  To specify, ALL CORRECT FIELDS, automatic editing must be enabled for the map.

- **ALL ERROR FIELDS**
  Modifies all fields that are found to be in error during automatic editing or by a user-written edit module.
  If you specify ALL ERROR FIELDS, automatic editing must be enabled for the map.

- **ALL**
  Modifies all named map fields. This value is the default.

- **all BUT (EXCEPT)**
  Modifies all map fields except for the named fields.
  BUT and EXCEPT are synonyms and can be used interchangeably.

- **DFLD field-name**
  Specifies the map fields to modify or exclude from modification. Separate multiple DFLD specifications by at least one blank. Multiple DFLD specifications must come from the same map record. Field names that are not unique within the program must be qualified with the name of the associated record.

  Likewise, multiply occurring fields must be qualified with the appropriate subscripts. Use the following syntax:

  \[
  \text{map-data-field-name} \underbrace{\text{subscript}}_{\text{IN}} \underbrace{\text{OF}}_{\text{record-name}}
  \]
Modification Options

- **BACKSCAN**
  Backscans the specified fields and removes trailing blanks before performing a MAPOUT operation. Only the characters up to the last nonblank are sent to the terminal. Fields that remain on the screen contain whatever characters were present before the MAP OUT or MAP OUTIN request was issued. If the MAP OUT or MAP OUTIN request specifies the ERASE option, the contents of all terminal data fields are erased.

- **NOBACKSCAN**
  Does not backscan the specified fields and remove trailing blanks.

- **OUTPUT DATA IS**
  Specifies how to treat the output map fields.
  - **YES**
    Sets the fields to the value of the corresponding variable-storage data fields.
  - **NO**
    Leaves the fields unchanged.
  - **ERASE**
    Erases the fields.
  - **ATTRIBUTE**
    Transmits only the attribute byte of the fields.

- **INPUT DATA is YES**
  Moves map fields automatically to the corresponding variable-storage data fields during an input operation.

- **INPUT DATA is NO**
  Does not move map fields to the corresponding variable-storage data fields during an input operation.

- **RIGHT JUSTIFY**
  Right justifies the variable-storage fields on input.

- **LEFT JUSTIFY**
  Left justifies the variable-storage fields on input.

- **PAD**
  Specifies whether to pad variable-storage data fields on input and defines the padding value or character.
  If RIGHT JUSTIFY is specified, fields are padded on the left. If LEFT JUSTIFY is specified, fields are padded on the right.
  - **pad-character**
    Specifies the symbolic name of a user-defined PIC X DISPLAY field that contains the pad character, or the character itself enclosed in quotation marks.
- **LOW-VALUE**
  Pads the fields with zeros.

- **HIGH-VALUE**
  Pads the fields with the highest value in the computer collating sequence.

- **EDIT IS ERROR**
  Sets the error flag on for the specified map fields. Automatic editing must be enabled for the map. Setting the error flag enables programs to perform their own editing and validation in addition to that provided by the automatic editing feature. On a MAPOUT operation, when any field has an error flag, only attribute bytes are transmitted for all fields (both CORRECT and INCORRECT). No data is moved from program variable storage to the screen.

- **EDIT IS CORRECT**
  Sets the error flag off for the specified map fields. Automatic editing must be enabled for the map. Setting the error flag enables programs to perform their own editing and validation in addition to that provided by the automatic editing feature. On a MAPOUT operation, when any field has an error flag, only attribute bytes are transmitted for all fields (both CORRECT and INCORRECT). No data is moved from program variable storage to the screen.

- **REQUIRED**
  Requires the user to type data in the specified map fields. If no data is entered, an error occurs on MAPIN. Automatic editing must be enabled for the map and for the specified map fields.

- **OPTIONAL**
  Does not require the user to type data in the specified map fields.

- **error message is**
  Specifies whether to display an error message that is associated with the field.

  - **ACTIVE**
    Displays the error message that is associated with the field. This value is the default. Typically, ACTIVE is specified after specifying ERROR MESSAGE SUPPRESS for the map in a previous MODIFY MAP PERMANENT statement.

  - **SUPPRESS**
    Does not display the error message. When the map is redisplayed because of errors, the message that is defined for the map field is not displayed even when the field contains edit errors.

- **ATTRIBUTES**
  Indicates the 3270- and 3279-type terminal display attributes for the specified map fields. Only the named attributes are modified in the MRB. Separate multiple attributes by blanks.

  - **SKIP**
    Repositions the cursor automatically past the map fields to the next unprotected field. When you specify SKIP, the NUMERIC and PROTECTED attributes are assigned to the affected map fields automatically.
- **ALPHAMERIC**
  Allows the user to type any character on the 3270 keyboard into the map fields.

- **NUMERIC**
  Allows the user to type numeric characters only into the map fields. If the terminal does not have the numeric lock option, NUMERIC is ignored.

- **PROTECTED**
  Protects the specified map fields from data entry or modification by the user.

- **UNPROTECTED**
  Makes the specified map fields available for data entry or modification by the user.
  You cannot specify both UNPROTECTED and SKIP.

- **DISPLAY**
  Displays the specified map fields in normal intensity.

- **DARK**
  Does not display the specified map fields.
  You cannot specify both DARK and DETECT.

- **BRIGHT**
  Displays the specified map fields in bright intensity.
  Fields that have the BRIGHT attribute are detectable by a light pen.

- **DETECT**
  Makes the specified map fields detectable by a light pen.
  Fields that have the BRIGHT attribute are also detectable by a light pen.

- **MDT**
  Sets the modified data tag automatically for the map fields when they are displayed.

- **NOMDT**
  Does not set the modified data tag automatically for the map fields when they are displayed.

- **BLINK**
  Displays the specified map fields with blinking characters. This option applies only to 3279 terminals.
  If you specify BLINK, you cannot specify REVERSE-VIDEO or UNDERSCORE.

- **NOBLINK**
  Does not display the specified map fields with blinking characters. This option applies only to 3279 terminals.

- **REVERSE-VIDEO**
  Displays the specified map fields in reverse video (background and character colors reversed).
  This option applies only to 3279 terminals.
  If you specify REVERSE-VIDEO, you cannot specify BLINK or UNDERSCORE.

- **NORMAL-VIDEO**
  Displays the specified map fields in normal video. This option applies only to 3279 terminals.
UNDERSCORE
Displays the specified map fields with underlined characters. This option applies only to 3279 terminals.
If you specify UNDERSCORE, you cannot specify BLINK or REVERSE-VIDEO.

NOUNDERSCORE
Displays the specified map fields without underlined characters. This option applies only to 3279 terminals.

NOCOLOR/BLUE/RED/PINK/GREEN/TURQUOISE/YELLOW/WHITE
Specifies a display color for the specified map fields. This option applies only to 3279 terminals.

Examples

The following examples illustrate the use of the MODIFY MAP statement.

Example 1
The following statement positions the cursor at EMP-ID-0415 and prohibits the user from entering data in any field except EMP-ID-0415 and DEPT-ID-0410:

```
MODIFY MAP EMPMAPLR TEMPORARY
CURSOR AT DFLD EMP-ID-0415
FOR ALL BUT DFLD EMP-ID-0415
DFLD DEPT-ID-0410
ATTRIBUTES PROTECTED.
```

Example 2
The following statement sets the edit flag on for the TASK-CODE-01 field, which overrides automatic editing and error handling for the next MAP IN request:

```
MODIFY MAP EMPMAPLR TEMPORARY
FOR DFLD TASK-CODE-01
EDIT IS ERROR.
```

Example 3
The following statement suppresses the display of default error messages for fields EMP-ID and DEPT-ID on the current map:

```
MODIFY MAP EMPMAPLR TEMPORARY
FOR DFLD EMP-ID DFLD DEPT-ID
ERROR MESSAGE IS SUPPRESS.
```

Because this MODIFY MAP statement specifies TEMPORARY, the error messages are suppressed only for the next MAPOUT. If PERMANENT (default) was specified, the error messages would be suppressed until the program terminated or until a subsequent MODIFY MAP overrode the error message specifications.

Status Codes

After completion of the MODIFY MAP function, the ERROR-STATUS field in the IDMS-DC communications block indicates the outcome:
<table>
<thead>
<tr>
<th>Status code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>The request has been serviced successfully.</td>
</tr>
<tr>
<td>4629</td>
<td>The program passed an invalid parameter.</td>
</tr>
<tr>
<td>4644</td>
<td>The map field is not in the specified map. A reference to a map field</td>
</tr>
<tr>
<td></td>
<td>subscript may be invalid.</td>
</tr>
<tr>
<td>4656</td>
<td>The referenced map contains no data fields.</td>
</tr>
</tbody>
</table>

**OBTAIN (LRF) (COBOL)**

The OBTAIN statement retrieves a logical record and places it in the variable-storage location that is reserved for that logical record. The OBTAIN statement can retrieve a single logical record, or it can be issued in iterative logic to retrieve all logical records that meet the criteria in the WHERE clause. The OBTAIN statement can also place the logical record into an alternative variable storage location.

**Syntax**

```
OBTAIN logical-record-name
    FIRST
    NEXT
    INTO alt-logical-record-location
    WHERE boolean-expression
    ON path-status imperative-statement .
```

**Parameters**

- **FIRST**
  Retrieves the first occurrence of the logical record. Typically, OBTAIN FIRST is used to retrieve the first logical record in a series of occurrences, following the iterative retrieval of a different series of logical-record occurrences.

- **NEXT**
  Retrieves a subsequent occurrence of the named logical record, in the order that the DBA specified in the path. This value is the default. Typically, OBTAIN NEXT is used in iterative logic to retrieve a series of logical-record occurrences (possibly including the first). When LRF receives repeated OBTAIN NEXT commands, it replaces the field values in program variable storage with new values that are obtained through repeated access to the appropriate database records, thereby supplying the program with new occurrences of the desired logical record.
  If an OBTAIN FIRST statement is followed by an OBTAIN NEXT statement to retrieve a series of occurrences of the same logical record, the OBTAIN statements must direct LRF to the same path. Ensure that the selection criteria in the WHERE clause that accompanies the OBTAIN FIRST and OBTAIN NEXT statements describe the same attributes of the desired logical record.
If the program issues an OBTAIN NEXT statement without issuing an OBTAIN FIRST, or the last path status was LR-NOT-FOUND, LRF interprets the OBTAIN NEXT as OBTAIN FIRST. After LR-ERROR or a DBA-defined path status, LRF does not interpret OBTAIN NEXT as OBTAIN FIRST.

- **logical-record-name**
  Specifies a logical record that is defined in the subschema.

- **INTO **`alt-logical-record-location`
  Specifies an alternative location in variable storage in which to place the retrieved logical record. The record location must be defined in the WORKING-STORAGE SECTION or LINKAGE SECTION. Any subsequent MODIFY, STORE, or ERASE statements for a logical record that is placed in `alt-logical-record-location` must name that area as the one from which to obtain the data to update `logical-record-location` the logical record.

- **WHERE boolean-expression**
  Specifies the selection criteria to apply to the logical record. For details on coding this clause, see Logical-Record Clauses (COBOL) (see page 272).

- **ON**
  Specifies the action to take, depending on the value that is returned to the LR-STATUS field in the LRC block. For details on coding this clause, see Logical-Record Clauses (COBOL) (see page 272).

  - **path-status**
    Specifies the value of the LR-STATUS field in the LRC block that triggers the specified action.

  - **imperative-statement**
    Specifies the action to take.

**Example**

The following example retrieves a series of logical-record occurrences. The program issues the OBTAIN NEXT statement iteratively to retrieve the first and all subsequent occurrences of the EMP-JOB-LR logical record for all employees in the specified department:

```cobol
GET-AN-ORDER.
  MOVE DEPT-ID-IN TO DEPT-ID-0410.
  OBTAIN NEXT EMP-JOB-LR WHERE DEPT-ID-410 EQ DEPT-ID-0410 OF LR.
  IF LR-STATUS = LR-ERROR
    PERFORM ERROR-PROCESSING.
  IF LR-STATUS = LR-NOT-FOUND
    PERFORM END-PROCESSING.
  .
  GO TO GET-AN-ORDER.
```

**OBTAIN NEXT EMP-JOB-LR**

The following figure illustrates the information that each OBTAIN NEXT statement retrieves. The EMP-JOB-LR logical record consists of DEPARTMENT, OFFICE, EMPLOYEE, and JOB information:

<table>
<thead>
<tr>
<th>DEPARTMENT</th>
<th>EMPLOYEE</th>
<th>OFFICE</th>
<th>JOB</th>
</tr>
</thead>
<tbody>
<tr>
<td>ONE OCCURRENCE OF EMP-JOB-LR</td>
<td>5100</td>
<td>466</td>
<td>8 SNOWBLOWER</td>
</tr>
</tbody>
</table>
POST (COBOL)

The POST statement alters an event control block (ECB) by posting it to indicate completion of an event upon which another task is waiting, or by clearing it to an unposted status.

Syntax

```
POST EVENT ecb .
```  

Parameters

- **EVENT ecb**
  Specifies the ECB to post. Specify the symbolic name of a user-defined area that contains three PIC S9(8) COMP SYNC (fullword) fields. Program-allocated ECBS are cleared by moving zeros to ecb.

- **EVENT NAME ecb-id**
  Specifies the ECB ID to post or clear. Specify the symbolic name of a user-defined field that contains the ECB ID, or the ID itself enclosed in quotation marks.

- **CLEAR**
  Clears the specified ECB ID to an unposted status. Programs that are posting and waiting on ECBS are responsible for clearing the ECBS before issuing subsequent WAIT requests.

Example

The following example posts the event whose ECB identifier is in the FOUND-ECB field and clears the ECB to an unposted status:

```
POST EVENT NAME FOUND-ECB CLEAR .
```  

Status Codes

After completion of the POST function, the only possible value in the ERROR-STATUS field of the IDMS-DC communications block is 0000.
PUT QUEUE (COBOL)

The PUT QUEUE statement stores a queue record in the DDLDCRUN or the DDLDCQUE area of the data dictionary. An ID is assigned to the queue record and placed at the beginning or end of its associated queue.

Syntax

```
PUT QUEUE ID queue-id FIRST
          FROM queue-data-location TO end-queue-data-location
          LENGTH queue-data-length
          RETURN RECORD ID INTO return-queue-record-id
          RETENTION queue-retention-period
```

Parameters

- **ID queue-id**
  Directs the queue record to a previously defined queue. Specify the symbolic name of a user-defined field that contains the ID, or the ID itself enclosed in quotation marks.
  Default: 16 blanks

- **FIRST**
  Places the queue record at the beginning of the queue.

- **LAST**
  Places the queue record at the end of the queue. This value is the default.

- **FROM queue-data-location**
  Specifies the WORKING-STORAGE SECTION or LINKAGE SECTION entry that is associated with the data to be stored in the queue record. Specify the symbolic name of a user-defined field.

- **TO end-queue-data-location**
  Specifies the end of the WORKING-STORAGE SECTION or LINKAGE SECTION entry that contains the data to be stored in the queue. Specify the symbolic name of a user-defined dummy byte field or a field containing a data item that is not associated with the queue record.

- **LENGTH queue-data-length**
  Defines the length, in bytes, of the area that contains the data to be stored in the queue record. Specify the symbolic name of a user-defined field that contains the length, or the length itself expressed as a numeric constant.

- **RETURN RECORD ID INTO return-queue-record-id**
  Specifies the program location in which to return the system-assigned ID of the queue record. Specify the symbolic name of a user-defined PIC S9(8) COMP (fullword) field. The returned ID is used to reference the queue record in subsequent GET QUEUE and DELETE QUEUE statements.
• **RETENTION queue-retention-period**
  Specifies the time, in days, to retain the queue in the data dictionary. Specify the symbolic name of a user-defined fixed binary field that contains the retention period, or the retention period itself, expressed as a numeric constant in the range 0 through 255. A retention period of 255 indicates that the queue is never to be automatically deleted.
  At system startup, queues with expired retention periods are deleted automatically. The retention period begins when the first record is stored in the queue.
  The specified retention period takes precedence over retention periods that are associated with previously defined queues. The RETENTION parameter is ignored when the record being allocated is not the first record in the queue.

  **Note:** If RETENTION is omitted, the default retention period for dynamic queues is taken. For more information about the default retention period for dynamic queues, see Queue Statement Parameters (https://docops.ca.com/display/IDMS19/QUEUE+Statement+Parameters).

**Example**

The following example allocates a queue record in the beginning of the RES-Q queue, returns the record ID to the Q-REC-ID field, and retains the queue for 45 days:

```plaintext
PUT QUEUE
  ID 'RES-Q'
  FIRST
  FROM NEW-RES TO END-NEW-RES
  RETURN RECORD ID INTO Q-REC-ID
  RETENTION 45.
```

**Status Codes**

After completion of the PUT QUEUE function, the ERROR-STATUS field in the IDMS-DC communications block indicates the outcome:

<table>
<thead>
<tr>
<th>Status code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>The request has been serviced successfully.</td>
</tr>
<tr>
<td>0019</td>
<td>In a DC-BATCH environment, the record size exceeds the MAX LENGTH value in the BIND TASK statement.</td>
</tr>
<tr>
<td>4407</td>
<td>A database error occurred during queue processing. A common cause is a DBKEY deadlock. For a PUT QUEUE operation, this code can also mean that the queue upper limit has been reached.</td>
</tr>
<tr>
<td></td>
<td>If a database error has occurred, the CA-IDMS/DC/UCF log usually contains other messages that indicate a problem in RHDCRUAL, the internal Run Unit Manager. If a deadlock has occurred, messages DC001000 and DC001002 are also produced.</td>
</tr>
<tr>
<td>4431</td>
<td>The parameter list is invalid. Under DC-BATCH, this status indicates that the specified record length exceeds the maximum length based on the packet size.</td>
</tr>
<tr>
<td>4432</td>
<td>The derived length of the specified queue record is zero or negative.</td>
</tr>
</tbody>
</table>
PUT SCRATCH (COBOL)

The PUT SCRATCH statement stores or replaces a scratch record in the DDLDCSCR area of the data dictionary. For new records, PUT SCRATCH generates an index entry in a scratch area that is associated with the issuing task. If the scratch area does not exist, it is allocated dynamically in the storage pool.

Syntax

```
PUT SCRATCH  AREA ID scratch-area-id
              FROM scratch-data-location  TO end-scratch-data-location
              LENGTH scratch-data-length
              RECORD ID scratch-record-id
              REPLACE
              RETURN RECORD ID into return-scratch-record-id
```

Parameters

- **AREA ID scratch-area-id**
  Specifies the scratch area that is associated with the record being allocated. Specify the symbolic name of a user-defined field that contains the ID, or the ID itself enclosed in quotation marks. If you do not specify an AREA ID, an area ID of eight blanks is assumed.

- **FROM scratch-data-location**
  Specifies the data to store in the scratch record. Specify the symbolic name of a user-defined WORKING-STORAGE SECTION or LINKAGE SECTION entry that contains the data.

- **TO end-scratch-data-location**
  Specifies the end of the data area to store in the scratch record. Specify the symbolic name of a user-defined dummy byte field or a field containing a data item that is not associated with the scratch data being stored.

- **LENGTH scratch-data-length**
  Defines the length of the data area in bytes. Specify the symbolic name of a user-defined field that contains the length, or the length itself expressed as a numeric constant.

- **RECORD ID scratch-record-id**
  Specifies the ID of the scratch record being stored. Specify the symbolic name of a user-defined PIC S9(8) COMP (fullword) field that contains the ID, or the ID itself expressed as a numeric constant.

- **REPLACE**
  Specifies that the specified scratch record replaces an existing scratch record. If you specify REPLACE, and the specified scratch record does not exist, the record is stored, and the status code is set to 0000.
RETURN RECORD ID into return-scratch-record-id

Returns the automatically assigned ID of a scratch record to the program. Specify the symbolic name of a user-defined field into which to place the 4-byte scratch record ID.

Example

The following statement replaces the scratch record that is identified by SCR-REC-ID with data in the WORK-PROC-AREA field:

```
PUT SCRATCH
  FROM WORK-PROC-AREA LENGTH 125
  RECORD ID SCR-REC-ID REPLACE.
```

Status Codes

After completion of the PUT SCRATCH function, the ERROR-STATUS field in the IDMS-DC communications block indicates the outcome:

<table>
<thead>
<tr>
<th>Status code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>The request to add a scratch record has been serviced successfually.</td>
</tr>
<tr>
<td>4305</td>
<td>The requested scratch record ID cannot be found.</td>
</tr>
<tr>
<td>4307</td>
<td>An I/O error has occurred during processing.</td>
</tr>
<tr>
<td>4317</td>
<td>The request to replace a scratch record has been serviced successfually.</td>
</tr>
<tr>
<td>4322</td>
<td>The request to add a scratch record cannot be serviced because the specified scratch record already exists in the scratch area and REPLACE has not been specified.</td>
</tr>
<tr>
<td>4331</td>
<td>The parameter list is invalid.</td>
</tr>
<tr>
<td>4332</td>
<td>The derived length of the specified scratch record is zero or negative</td>
</tr>
</tbody>
</table>

READ LINE FROM TERMINAL (COBOL)

The READ LINE FROM TERMINAL statement requests a synchronous, line-by-line transfer of data from the terminal to the issuing program.

Syntax

```
READ LINE FROM TERMINAL [ ECHO ] [ NOBACKPAGE ]
  INTO input-data-location [ TO end-input-data-location ]
  [ MAX LENGTH input-data-max-length ]
  [ RETURN LENGTH into input-data-actual-length ]
```
Parameters

- **ECHO**
  Saves the line of data being input in the current page (as displayed on the screen). If you do not specify ECHO, the entered data is not retained and is not available for review by the user. This parameter applies only to 3270-type devices.

- **NOBACKPAGE**
  Does not save previously input pages in a scratch area. If you specify NOBACKPAGE, the user can view only the current page of data. NOBACKPAGE is valid only with the first input request in a line mode session. This parameter applies only to 3270-type devices.

- **INTO input-data-location**
  Indicates the WORKING-STORAGE SECTION or LINKAGE SECTION entry that is reserved for the input data. Specify the symbolic name of a user-defined field.

- **TO end-input-data-location**
  Indicates the end of the WORKING-STORAGE SECTION or LINKAGE SECTION that is reserved for the input data. Specify the symbolic name of a user-defined dummy byte field or a field containing a data item that is not associated with the reserved data area.

- **MAX LENGTH input-data-max-length**
  Defines the length of the input data stream in bytes. Specify the symbolic name of a user-defined field that contains the length of the data area, or the length itself expressed as a numeric constant.
  If the input data stream is larger than the data area that is reserved in the WORKING-STORAGE SECTION or LINKAGE SECTION, the data is truncated to fit the available space.

- **RETURN LENGTH into input-data-actual-length**
  Indicates where to return the actual length of the input data stream. Specify the symbolic name of a user-defined field. If the data stream has been truncated, the field contains the original length before truncation.

Examples

The following examples illustrate the use of the READ LINE FROM TERMINAL statement.

- **Example 1**
  The following statement reads the specified data from a 3270-type device into the specified location in the program and echoes the input data on the screen:

    ```plaintext
    READ LINE FROM TERMINAL
    ECHO
    INTO EMPL-DATA TO END-EMPL-DATA.
    ```

- **Example 2**
  The following statement reads the specified data into the program but does not save the pages that are associated with the line I/O session:

    ```plaintext
    READ LINE FROM TERMINAL
    NOBACKPAGE
    INTO EMPL-DATA MAX LENGTH 8
    RETURN LENGTH INTO REC-DATA-LENGTH.
    ```
Status Codes

After completion of the READ LINE FROM TERMINAL function, the ERROR-STATUS field in the IDMS-DC communications block indicates the outcome:

<table>
<thead>
<tr>
<th>Status code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>The request has been serviced successfully.</td>
</tr>
<tr>
<td>4707</td>
<td>A logical or permanent I/O error has been encountered in the input data stream.</td>
</tr>
<tr>
<td>4719</td>
<td>The input area that was specified for the return of data is too small. The returned data has been truncated to fit the available space.</td>
</tr>
<tr>
<td>4731</td>
<td>The line request block (LRB) contains an invalid field, indicating a possible error in the program parameters.</td>
</tr>
<tr>
<td>4732</td>
<td>The derived length of the specified line input area is zero or negative.</td>
</tr>
<tr>
<td>4738</td>
<td>The specified 01-level LINKAGE SECTION entry has not been allocated as required. A prior GET STORAGE request must be issued.</td>
</tr>
<tr>
<td>4743</td>
<td>The line I/O session has been canceled. The user has pressed CLEAR (3270s), ATTENTION (2741s), or BREAK (teletypes).</td>
</tr>
</tbody>
</table>

READ TERMINAL (COBOL)

The READ TERMINAL statement requests a synchronous or asynchronous basic mode data transfer from the terminal to program variable storage.

Syntax

```
READ TERMINAL
   WAIT [NOWAIT]
   BUFFER [MODIFIED FROM POSITION screen-position]
   GET STORAGE
   INTO input-data-location [TO end-input-data-location]
   MAX LENGTH input-data-max-length
   RETURN LENGTH into input-data-actual-length
```

Parameters

- **WAIT**
  Specifies a synchronous read operation. The issuing task automatically relinquishes control to CA IDMS and waits for completion of the read operation before continuing processing. This value is the default.
NOWAIT
Specifies an asynchronous read operation. The issuing task continues executing. If you specify NOWAIT, the program must issue a CHECK TERMINAL (COBOL) (see page 115) request before performing any other I/O operations.

MODIFIED
Reads all modified fields in the terminal buffer into variable storage without requiring the user to signal completion of data entry. This parameter applies only to 3270-type devices.

BUFFER
Executes a READ BUFFER command that reads the entire contents of the terminal buffer into variable storage without requiring the user to signal completion of data entry. This parameter applies only to 3270-type devices.

FROM POSITION screen-position
Defines the buffer address (screen position) at which the read will start. Specify the symbolic name of a user-defined PIC S9(4) COMP SYNC (halfword) field, or the address itself enclosed in quotation marks.

GET STORAGE
Acquires an input buffer for the data being read into the program. The required storage is allocated when the read operation is complete. This parameter applies only to synchronous requests.

INTO input-data-location
Specifies the 01-level WORKING-STORAGE SECTION or LINKAGE SECTION entry of the input data stream. Specify the symbolic name of a user-defined field.
If you also specify GET STORAGE, the data area that is reserved for the input data stream must be an unallocated 01-level LINKAGE SECTION entry.
If you do not specify GET STORAGE, the data area must be a WORKING STORAGE or a previously allocated LINKAGE SECTION entry.

TO end-input-data-location
Specifies the end of the data area that is reserved for the input data stream. Specify the symbolic name of a user-defined dummy byte field or a field containing a data item that is not associated with the reserved data area.

MAX LENGTH input-data-max-length
Defines the length, in bytes, of the data area that is reserved for the input data stream. Specify the symbolic name of a user-defined field that contains the length of the data area, or the length itself expressed as a numeric constant.
If the input data stream is larger than the specified WORKING-STORAGE SECTION or LINKAGE SECTION entry, the data is truncated to fit the available space.

RETURN LENGTH into input-data-actual-length
Specifies where to return the actual length of the input data stream. Specify the symbolic name of a user-defined field. If the data stream has been truncated, input-data-actual-length contains the original length before truncation.
Example

The following statement illustrates a basic mode request to read data from the terminal to the specified location in variable storage:

```
READ TERMINAL
WAIT
INTO TERM-LINE TO END-TERM-LINE.
```

Status Codes

After completion of the READ TERMINAL function, the ERROR-STATUS field in the IDMS-DC communications block indicates the outcome:

<table>
<thead>
<tr>
<th>Status code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>The request has been serviced successfully.</td>
</tr>
<tr>
<td>4519</td>
<td>The input area that was specified for the return of data to the issuing program is too small. The returned data has been truncated to fit the available space.</td>
</tr>
<tr>
<td>4527</td>
<td>A permanent I/O error has occurred during processing.</td>
</tr>
<tr>
<td>4528</td>
<td>The dial-up line for the terminal has been disconnected.</td>
</tr>
<tr>
<td>4531</td>
<td>The terminal request block (TRB) contains an invalid field, indicating a possible error in the program parameters.</td>
</tr>
<tr>
<td>4532</td>
<td>The derived length of the specified input data area is zero or negative.</td>
</tr>
<tr>
<td>4535</td>
<td>Storage for the input buffer cannot be acquired because the specified 01-level LINKAGE SECTION entry has been previously allocated. No I/O has been performed.</td>
</tr>
<tr>
<td>4537</td>
<td>Storage for the input buffer cannot be acquired because the specified entry is defined in the WORKING-STORAGE SECTION instead of the LINKAGE SECTION. No I/O has been performed.</td>
</tr>
<tr>
<td>4538</td>
<td>The specified 01-level LINKAGE SECTION entry has not been previously allocated and the GET STORAGE option has not been specified. No I/O has been performed.</td>
</tr>
<tr>
<td>4539</td>
<td>The terminal device that is associated with the issuing task is out of service.</td>
</tr>
</tbody>
</table>

READY (COBOL)

The READY statement prepares a database area for access by DML functions and specifies the usage mode of the area.

The DBA can specify default usage modes in the subschema. Run units that use such a subschema need not issue any READY statements. The areas are readied in the predefined usage modes automatically. However, if a run unit issues a READY statement for one area, it must issue READY statements for all areas that it accesses unless the FORCE option was specified for the default usage mode. Areas using the default usage mode combined with the FORCE option are readied automatically even when the run unit has already issued READY for other areas.
The specified usage mode can be qualified with a PROTECTED option to prevent concurrent updates or an EXCLUSIVE option to prevent concurrent use of areas by other run units that are executing under the central version. Each area can be readied in its own usage mode. Usage modes can be changed by executing a `FINISH (COBOL)` statement, and then starting a new run unit by issuing a `BIND RUN-UNIT` statement, the appropriate `BIND RECORD` statements, and a `READY` statement specifying the new usage mode.

When the run unit readies database areas, all areas can be readied with a single `READY` statement or each area can be readied individually. All areas that are affected explicitly or implicitly by the DML statements that the run unit issues must be readied. Other areas that are included in the subschema need not be readied.

The `READY` statement can appear anywhere within an application program. However, to avoid runtime deadlock, ready all areas before issuing any other DML statements.

The `READY` statement is used in both the navigational and the non-navigational environments.

### Syntax

```
READY  [area-name]

USAGE-MODE IS [PROTECTED] [EXCLUSIVE]
```

### Parameters

- **READY area-name**
  Specifies an area that is included in the subschema. By default, if you do not specify an area, `READY` opens all areas in the subschema.

- **USAGE-MODE IS**
  Specifies the usage mode in which to open the area.

  - **PROTECTED**
    Prevents concurrent updates of the area by run units that are executing under the same central version. After a run unit has readied an area with the PROTECTED option, no other run unit can ready that area in any UPDATE usage mode until the first run unit releases it with a `FINISH` statement. A run unit cannot ready an area with the PROTECTED option when another run unit has readied the area in UPDATE usage mode or with the EXCLUSIVE option. By default, if you do not specify PROTECTED or EXCLUSIVE, the areas are opened in shared mode.

  - **EXCLUSIVE**
    Prevents concurrent use of the area by any other run unit that is executing under the central version. After a run unit has readied an area with the EXCLUSIVE option, no other run unit can ready that area in any usage mode until the first run unit releases it. By default, if you do not specify PROTECTED or EXCLUSIVE, the areas are opened in shared mode.
- **RETRIEVAL**
  Opens the area for retrieval only and allows other concurrently executing run units to open the same area in any nonexclusive usage mode. This value is the default.

- **UPDATE**
  Opens the area for both retrieval and update and allows other concurrently executing run units to open the same area in any usage mode other than exclusive or protected.

---

**Notes:**

- If a READY statement would result in a usage mode conflict for an area, while running under the central version, the run unit issuing the READY is placed in a wait state on the first functional database call.
- Modification statements involving areas that are opened in one of the update usage modes are not valid when they affect sets that include records in an area that is opened in one of the retrieval usage modes.

---

### Example

The following statement readies all subschema areas in a usage mode of PROTECTED UPDATE:

```
READY USAGE-MODE IS PROTECTED UPDATE.
```

### Status Codes

After completion of the READY function, the ERROR-STATUS field in the IDMS communications block indicates the outcome:

<table>
<thead>
<tr>
<th>Status code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>The request has been serviced successfully.</td>
</tr>
<tr>
<td>0910</td>
<td>The subschema specifies an access restriction that prohibits readying the area in the specified usage mode.</td>
</tr>
<tr>
<td>0923</td>
<td>The named area is not in the subschema.</td>
</tr>
<tr>
<td>0928</td>
<td>The run unit has attempted to ready an area that has been readied previously.</td>
</tr>
<tr>
<td>0966</td>
<td>The specified area is not available for update. If the 0966 status code is ignored, subsequent attempts to access the area return a 01 or 09 minor code. Probable causes for the return of the status code are as follows: If running in local mode, the area is locked against update. If running under the central version, the area is not available to the program in the desired access mode.</td>
</tr>
<tr>
<td>Status code</td>
<td>Meaning</td>
</tr>
<tr>
<td>-------------</td>
<td>---------</td>
</tr>
<tr>
<td>0970</td>
<td>The database or journal file will not ready properly. A JCL error is the probable cause.</td>
</tr>
<tr>
<td>0971</td>
<td>The AREA being readied could not be found in the current DBNAME.</td>
</tr>
<tr>
<td>0978</td>
<td>A wait for an area would cause a deadlock. We recommend that all areas be readied before the first functional call is issued or that all programs ready areas in the same order.</td>
</tr>
</tbody>
</table>

**RETURN (COBOL)**

The RETURN statement retrieves the database key for an indexed record without retrieving the record itself, thus establishing currency in the indexed set. The symbolic key is moved into the data fields within the record in program variable storage. The contents of all nonkey fields for the record are unpredictable after the execution of the RETURN verb. Optionally, the program can move the symbolic key into some other specified variable storage location.

The following items establish current of index:

- Successful execution of the RETURN statement, which sets current of index at the index entry from which the database key was retrieved.
- A status code of 1707 (end of index), which sets currency on the index owner. The DBMS returns the database key of the owner.
- A status code of 1726 (index entry not found), which sets current of index as follows:
  - Between the two entries that are higher and lower than the specified value
  - After the highest entry, if the specified value is higher than all index entries
  - Before the lowest entry, if the specified value is lower than all index entries

The RETURN statement is used in both the navigational and the non-navigational environments.

**Syntax**

```plaintext
RETURN dbkey FROM index-set-name CURRENCY
    FIRST currency
    LAST currency
    NEXT currency
    PRIOR currency
    USING index-key-value

KEY into symbolic-key .
```
Parameters

- **RETURN db-key**
  Specifies which database key to retrieve. Specify the symbolic name of a user-defined PIC S9(8) COMP SYNC (fullword) field.

- **FROM index-set-name**
  Specifies the indexed set from which the database key is to be returned.

- **CURRENCY**
  Retrieves the database key for the current index entry.

- **FIRST currency**
  Retrieves the database key for the first index entry.

- **LAST currency**
  Retrieves the database key for the last index entry.

- **NEXT currency**
  Retrieves the database key for the index entry following current of index. If the current of index is the last entry, status code 1707 (end of index) is returned.

- **PRIOR currency**
  Retrieves the database key for the index entry preceding current of index. If the current of index is the first entry, status code 1707 (end of index) is returned.

- **USING index-key-value**
  Retrieves the database key for the first index entry with the specified symbolic key.

- **KEY into symbolic-key**
  Saves the symbolic key (CALC, sort, or index) of the specified record. Specify a user-defined alphanumeric field into which to return the key. Symbolic-key must be large enough to contain the largest contiguous or noncontiguous symbolic key. If KEY into is not specified, the symbolic key is moved into the corresponding fields in the user record storage. The precompiler views an incorrectly formatted RETURN statement as a COBOL RETURN function and does not flag the error. The incorrect RETURN DML statement is passed to the COBOL compiler without expansion into a CALL statement, causing compile-time errors.

Example

The following RETURN statement retrieves the database key for the first index entry in the EMP-LNAME-NDX set and moves the symbolic key into the NDX-LNAME-SYM-KEY field:

```
RETURN INT-INDEX-KEY FROM EMP-LNAME-NDX
  FIRST CURRENCY
  KEY INTO NDX-LNAME-SYM-KEY.
```
Status Codes

After completion of the RETURN function, the ERROR-STATUS field in the IDMS communications block indicates the outcome:

<table>
<thead>
<tr>
<th>Status code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>The request has been serviced successfully.</td>
</tr>
<tr>
<td>1701</td>
<td>The area in which the object record or its index owner record participates has not been readied.</td>
</tr>
<tr>
<td>1707</td>
<td>The end of the indexed set has been reached or the indexed set is empty.</td>
</tr>
<tr>
<td>1725</td>
<td>Currency has not been established for the specified indexed set.</td>
</tr>
<tr>
<td>1726</td>
<td>The record cannot be found.</td>
</tr>
</tbody>
</table>

ROLLBACK (COBOL)

The ROLLBACK statement rolls back uncommitted changes that were made to the database through an individual run unit or through all database sessions that are associated with a task. A task-level rollback also backs out all uncommitted changes that were made with scratch, queue, and print activity.

The execution environment determines whether the changes are automatically backed:

- If the changes were made under the control of a central version that is journaling to a disk file, they are backed out automatically. The central version continues to process other applications during recovery.
- The changes are not backed out automatically under the following circumstances:
  - If the changes were made under the control of a central version that is journaling to a tape file
  - If the changes were made in local mode

In these cases, the ROLLBACK statement causes the affected areas to remain locked against subsequent access by other database sessions. Recover the areas manually. If changes cannot be backed out and CONTINUE was specified on the rollback request, a nonzero error status is returned to the application. Also, if the request was for an individual run unit, that run unit is terminated.

Note: For more information about manual recovery, see Manual Recovery (https://docops.ca.com/display/IDMS19/Manual+Recovery).
If CONTINUE is not specified, the run units (and SQL sessions) that are impacted by the ROLLBACK statement end, and their access to the database terminates. If CONTINUE is specified, the impacted database sessions remain active after the operation is complete.

The ROLLBACK statement is used in both the navigational and logical record facility environments. The ROLLBACK TASK statement is also used in an SQL programming environment.

**Currency:** Following a ROLLBACK statement, all currencies are set to null. If CONTINUE is not specified, the issuing program or task cannot perform database access through an impacted run unit without executing another BIND/READY sequence.

### Syntax

```
ROLLBACK [TASK] [CONTINUE].
```

### Parameters

- **TASK**
  Rolls back the uncommitted changes and terminates the run units. All scratch, queue, and print activity is committed. All top-level run units that are associated with the current task are terminated. The impact on SQL sessions that are associated with the task depends on whether those sessions are suspended and whether their transactions are eligible to be shared.
  For more information about the impact of a ROLLBACK TASK statement on SQL sessions, see Effect of Teleprocessing Statement and Events (https://docops.ca.com/display/IDMS19/Effect+of+Teleprocessing+Statements+and+Events).
  For more information about run units and the impact of ROLLBACK TASK, see Sharing Transactions Among Sessions (https://docops.ca.com/display/IDMSCU19/Sharing+Transactions+Among+Sessions).

- **CONTINUE**
  Causes the affected run units and SQL sessions to remain active after their changes are backed out. Database access can be resumed without reissuing BIND and READY statements. This option applies only to the central version.

**Note:** Do not use the CONTINUE option in local mode when database changes have been made.

### Example

The following statement reverses the effects of the run unit through which it is issued and terminates the run unit:

```
ROLLBACK.
```
Status Codes

After completion of the ROLLBACK function, the ERROR-STATUS field in the IDMS communications block indicates the outcome:

<table>
<thead>
<tr>
<th>Status code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>The request has been serviced successfully.</td>
</tr>
<tr>
<td>1958</td>
<td>CONTINUE was specified and database changes could not be backed out. The run unit has been terminated.</td>
</tr>
<tr>
<td>5031</td>
<td>The specified request is invalid. The program may contain a logic error.</td>
</tr>
<tr>
<td>5058</td>
<td>TASK CONTINUE was specified and database changes could not be backed out.</td>
</tr>
<tr>
<td>5097</td>
<td>An error was encountered while processing a syncpoint request. See the log for details.</td>
</tr>
</tbody>
</table>

SEND MESSAGE (COBOL)

The SEND MESSAGE statement sends a message to another terminal or user, or to a group of terminals or users that is defined as a destination, during system generation. The SEND MESSAGE function does not employ the data dictionary message area. Instead, each message is placed in a queue, and sent to the appropriate terminal only when it is possible to do so without disrupting executing tasks. Typically, queued messages are sent to a terminal the next time the ENTER NEXT TASK CODE message is displayed.

Syntax

```
SEND MESSAGE [ONLY | ALWAYS]
TO DEST ID destination-id
    USER ID user-id
    LTERM ID lterm-id
FROM message-location TO end-message-location .
LENGTH message-length
```

Parameters

- ONLY
  Sends the message immediately when the destination, user, or terminal is available, and does not queue the message for subsequent transmission when the destination, user, or terminal is not available. This value is the default.

Note: If ONLY is specified with the DEST ID option and some, but not all, of a group of users or terminals in the destination are available, the message is sent to the available ones. The sender is not aware of any unsuccessful transmissions.
ALWAYS
Sends the message immediately when the destination, user, or terminal is available, and queues
the message for later transmission when the destination, user, or terminal is not available.

TO DEST ID destination-id
Specifies the destination ID of the message recipient. Specify the symbolic name of a user-defined
field that contains the destination ID, or the ID itself enclosed in quotation marks. The destination
must have been defined during system generation.

TO USER ID user-id
Specifies the user that receives the message. User-id is the symbolic name of a 32-byte user-
defined field that contains the user ID. The specified user can be signed on to any terminal.

TO LTERM ID lterm-id
Specifies the logical terminal that receives the message. Specify the symbolic name of a user-
defined field that contains the terminal ID, or the ID itself enclosed in quotation marks.

FROM message-location
Specifies the WORKING-STORAGE SECTION or LINKAGE SECTION entry that contains the message
text to send. Specify the symbolic name of a user-defined field.

TO end-message-location
Specifies the end of the WORKING-STORAGE SECTION or LINKAGE SECTION entry that contains
the message text. Specify the symbolic name of a user-defined dummy byte field or a field
containing a data item that is not associated with the message text.

LENGTH message-length
Defines the length of the message text in bytes. Specify the symbolic name of a user-defined field
that contains the length, or the length itself expressed as a numeric constant.

Examples

The following examples illustrate the use of the SEND MESSAGE statement.

Example 1
The following statement sends the message in the TERM-MESS field to the logical terminal
KENNEDYA:

SEND MESSAGE ALWAYS
   TO LTERM ID 'KENNEDYA'
   FROM TERM-MESS TO END-TERM-MESS.

Example 2
The following statement sends the message in the TERM-MESS field to the user field:

MOVE 'KYJOE2' to USER32.
SEND MESSAGE
   TO USER ID USER32
   FROM TERM-MESS TO END-TERM-MESS.

Example 3
The following statement sends the message in the TERM-MESS field to the destination ALL:
SEND MESSAGE
TO DEST ID 'ALL'
FROM TERM-MESS TO END-TERM-MESS.

Status Codes

After completion of the SEND MESSAGE function, the ERROR-STATUS field in the IDMS-DC communications block indicates the outcome:

<table>
<thead>
<tr>
<th>Status code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>The request has been serviced successfully.</td>
</tr>
<tr>
<td>4907</td>
<td>An I/O error has occurred during processing.</td>
</tr>
<tr>
<td>4921</td>
<td>The specified message recipient has not been defined.</td>
</tr>
<tr>
<td>4931</td>
<td>The parameter list is invalid.</td>
</tr>
<tr>
<td>4932</td>
<td>The derived length of the specified message data area is zero or negative.</td>
</tr>
<tr>
<td>4934</td>
<td>The specified variable storage area that contains the message text is in the LINKAGE SECTION but is not an 01-level entry.</td>
</tr>
<tr>
<td>4938</td>
<td>The specified WORKING-STORAGE SECTION or LINKAGE SECTION entry has not been allocated. A GET STORAGE request must be issued.</td>
</tr>
</tbody>
</table>

SET ABEND EXIT (COBOL)

The SET ABEND EXIT (STAE) statement establishes or cancels linkage to an abend routine to which CA IDMS passes control when the issuing task terminates abnormally. Any program within a task can establish an abend exit. However, only one abend exit is in effect at any given time for each task level. If more than one abend exit has been established, CA IDMS recognizes the exit that is associated with the last STAE request.

When a task terminates abnormally (after a processing error or an ABEND request), abend exits for the program that was executing at the time of the abend and for all higher-level programs are executed before the task is terminated. The program can prevent automatic execution of abend exits by coding the EXITS IGNORED clause in an ABEND request or by coding a DC RETURN request in the abend routine.

Syntax

```plaintext
SET ABEND EXIT on PROGRAM program OFF .
```
Parameters

- **on PROGRAM**
  Specifies where to transfer control when the issuing task terminates abnormally. Specify the symbolic name of a user-defined field that contains the program name, or the name itself enclosed in quotation marks.

  ![Note: CA IDMS does not verify whether the specified program name is valid when the STAE request is issued. If the program is not found or is unloadable when CA IDMS attempts to execute it, the request is ignored.]

- **OFF**
  Cancels any previous STAE request for the issuing task level.

Examples

The following examples illustrate the use of the SET ABEND EXIT statement.

- **Example 1**
  The following statement establishes an abend exit that executes the program ABENDRTN when the issuing task terminates abnormally:
  
  ```
  SET ABEND EXIT ON PROGRAM 'ABENDRTN'.
  ```

- **Example 2**
  The following statement cancels all abend exits that were previously established at the task level of the issuing program:
  
  ```
  SET ABEND EXIT OFF.
  ```

Status Codes

After completion of the SET ABEND EXIT function, the only possible value in the ERROR-STATUS field of the IDMS-DC communications block is 0000.

SET TIMER (COBOL)

The SET TIMER statement defines an event that occurs after a specified time interval or that cancels the effect of a previous SET TIMER request. Using the SET TIMER function, a program can perform the following tasks:

- Delay task processing for a specified timeframe.
- Post an ECB at the end of a specified timeframe.
Initiate a task at the end of a specified timeframe.

**Syntax**

```
SET TIMER ──WAIT─┬─POST─┬─START─┬─CANCEL─┐
├─INTERVAL ──time-interval seconds─┐
├─EVENT ──post-ecb─┐
└─TIMER ID ──ice-address─┐
```

**Parameters**

- **WAIT**
  Places the issuing task in a wait state and rediscovers the issuing task after the specified time interval elapses. Because WAIT relinquishes control until the time interval has elapsed, a subsequent SET TIMER request cannot be used to cancel this WAIT request.

- **POST**
  Posts a user-specified ECB after the specified time interval elapses. The issuing task continues to run. If POST is specified, the EVENT parameter must also be specified.

- **START**
  Initiates a user-specified task after the specified time interval elapses. If START is specified, the TASK CODE parameter must also be specified.

- **CANCEL**
  Cancels the effect of a previous SET TIMER request.

- **INTERVAL time-interval seconds**
  Specifies the time, in seconds, from the issuance of a SET TIMER request at which the requested event occurs. Specify the symbolic name of a user-defined field that contains the time interval, or the interval itself expressed as a numeric constant. This option applies only to WAIT, POST, and START requests.

- **EVENT post-ecb**
  Specifies the ECB to post. Specify the symbolic name of a user-defined area that contains three PIC S9(8) COMP SYNC (fullword) fields. This option applies only to POST requests.
Note: The POST instruction only posts an ECB that is within storage owned by the TASK initiating the POST instruction. If the same task does not own the storage, it is not executed.

- **TASK CODE** `task-code`
  Specifies the task to initiate. Specify the symbolic name of the user-defined field that contains the task code, or the task code itself enclosed in quotation marks. The task code must have been defined during system generation or at runtime with a DCMT VARY DYNAMIC TASK command. This option applies only to START requests.

- **PRIORITY** `priority`
  Specifies a dispatching priority for the task. Specify the symbolic name of a user-defined field that contains the priority, or the priority itself expressed as a numeric constant in the range 0 through 240. The priority for the new task defaults to the priority that is defined for that task code.

- **TIMER ID** `ice-address`
  Specifies the address of the interval control element (ICE) that is associated with the timed event. Specify the symbolic name of a user-defined PIC 9(8) COMP SYNC (fullword) field. If POST or START has been specified, `ice-address` references a field to which the ICE address is returned. If CANCEL has been specified, `ice-address` references the field containing the ICE address that was returned following a SET TIMER POST or SET TIMER START request. This option applies only to POST, START, and CANCEL requests.

  Note: If the program is to issue subsequent SET TIMER CANCEL requests, specify TIMER ID with SET TIMER POST and SET TIMER START requests.

- **DATA FROM** `task-data-location`
  Specifies the user data to pass to the new task. Specify the symbolic name of a user-defined field that identifies the beginning of an area containing the data items to pass. This option applies only to START requests.

- **TO** `end-task-data-location`
  Specifies the end of the data area being passed to the new task. Specify the symbolic name of a user-defined dummy byte field or a field containing a data item that is not associated with the data area being passed.

- **LENGTH** `task-data-length`
  Specifies the length of the data area in bytes. Specify the symbolic name of a user-defined WORKING-STORAGE SECTION or LINKAGE SECTION field that contains the length of the data area, or the length itself expressed as a numeric constant.

  Note: The data that is passed to the new task consists of two bytes containing the length of the original data followed by the original data itself. This data can be accessed by a LINKAGE SECTION entry corresponding to the data and a USING clause in the PROCEDURE DIVISION header.
Examples

The following examples illustrate the use of the SET TIMER statement.

- **Example 1**
The following statement places the issuing task in a wait state and redisc dispatches it after 9 seconds have elapsed:

```
SET TIMER WAIT
   INTERVAL 9 SECONDS.
```

- **Example 2**
The following statement posts the event PODB after 5 seconds have elapsed:

```
SET TIMER POST
   INTERVAL 5 SECONDS
   EVENT PODB
   TIMER ID TMR-ID.
```

- **Example 3**
The following statement starts the SPSG task after 5 seconds have elapsed and passes the specified data to that task:

```
SET TIMER START
   INTERVAL 5 SECONDS
   TASK CODE 'SPSG'
   TIMER ID TMR-ID
   DATA FROM PASSGR LENGTH REC-LENGTH.
```

- **Example 4**
The following statement cancels the timed event that TMR-ID references:

```
SET TIMER CANCEL
   TIMER ID TMR-ID.
```

Status Codes

After completion of the SET TIMER function, the ERROR-STATUS field in the IDMS-DC communications block indicates the outcome:

<table>
<thead>
<tr>
<th>Status code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>The request has been serviced successfully.</td>
</tr>
<tr>
<td>3512</td>
<td>The specified task code is not known to the DC/UCF system.</td>
</tr>
<tr>
<td>3516</td>
<td>The interval control element (ICE) that was specified for a SET TIMER CANCEL request cannot be found.</td>
</tr>
<tr>
<td>3531</td>
<td>The parameter list is invalid.</td>
</tr>
<tr>
<td>3532</td>
<td>The derived length of the data area is negative.</td>
</tr>
</tbody>
</table>
SNAP (COBOL)

The SNAP statement requests a memory snap of the following areas:

- **Task areas** – These areas include the associated resources, the task control element (TCE), and the dispatch control element (DCE) for the issuing task. The displayed information is formatted with headers.

- **System areas** – These areas include all tasks and CA IDMS internal control blocks. Task areas are not itemized separately. The displayed information is formatted with headers.

- **Specified locations in memory** – These areas include one or more memory areas that are requested by location and length. The displayed information is not formatted with headers.

The SNAP request writes the requested areas to the DC system log file. This log file is defined during system generation as a sequential data set or a data dictionary area.

### Syntax

```
SNAP TITLE is title [ALL SYSTEM TASK] FROM begin-snap-location TO end-snap-location LENGTH snap-length
```

### Parameters

- **TITLE is title**
  Specifies the title to print at the beginning of each snap page. Specify the symbolic name of a user-defined field that contains the title. The title must contain 134 characters. The first character is reserved for use by CA IDMS, and the second character must be a valid ASA carriage control character (blank, 0, +, or -).

- **ALL**
  Writes a snap of both task and system areas. The areas that are associated with the issuing task are formatted separately from the system areas. (Task areas are also included with the system areas but are not itemized by task.)

- **SYSTEM**
  Writes a snap of system areas.

- **TASK**
  Writes a snap of task areas.
- **FROM begin-snap-location**
  Writes a snap of the specified memory location. Specify the symbolic name of a user-defined field indicating the starting location of the area to be snapped.

- **TO end-snap-location**
  Specifies the end of the area to snap. Specify the symbolic name of a user-defined dummy byte field or a field containing a data item that is not associated with the requested area.

- **LENGTH snap-length**
  Defines the length in bytes of the area to include in the snap. Specify the symbolic name of a user-defined field that contains the length of the data area, or the length itself expressed as a numeric constant.

  **Note:** If `snap-length` is greater than 100, some COBOL compilers can produce errors. In this case, use a symbolic name that contains the length, or use the FROM/TO verb form.

**Example**

The following example requests a memory snap of the specified memory location:

```
SNAP TITLE IS SNAP-TITLE
   FROM WS-START TO WS-END.
```

**Status Codes**

After completion of the SNAP function, the ERROR-STATUS field in the IDMS-DC communications block indicates the outcome:

<table>
<thead>
<tr>
<th>Status code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>The request has been serviced successfully.</td>
</tr>
<tr>
<td>4032</td>
<td>The derived length of the specified snap storage area is zero or negative.</td>
</tr>
</tbody>
</table>

**STARTPAGE (COBOL)**

The STARTPAGE statement initiates a paging session. You can follow this statement with any number of DML commands, including MAP IN and MAP OUT. The map paging session is terminated by an ENDPAGE command (or by another STARTPAGE that occurs before an ENDPAGE).

  **Note:** The statements within a STARTPAGE/ENDPAGE pair can process only one pageable map.
### Syntax

```
STARTPAGE session map-name
   │      │              │      │
   │      │    WAIT    │      │ NOWAIT │
   │      │      ┌───┐      │      │
   │   ┌───┘  NOWAIT  ┌───┘
   │   │      │      │      │
   │   │      │      │      │
   │   │      │      │      │
   └───┘      │      │      └───┘
     BACKPAGE NOBACKPAGE UPDATE BROWSE AUTODISPLAY NOAUTODISPLAY
```

### Parameters

- **map-name**
  Specifies the pageable map to use.

- **WAIT**
  Specifies that runtime mapping automatically handles paging transactions that do not update data. If the user presses a control key that requests an update or nonpaging operation, control passes to the program.

- **NOWAIT**
  Specifies that runtime mapping automatically handles all paging and update transactions. This value is the default. Control passes to the program only when neither an update nor paging request is made when the operator presses a control key.

- **RETURN**
  Specifies that runtime mapping does not handle any terminal transactions in the paging session. Control passes to the program whenever the operator presses a control key.

**Note:** Runtime mapping updates program variable storage only when a MAP IN command is issued. In cases where the operator can update data, we recommend specifying WAIT or RETURN for the session so that data can be retrieved as it is updated.

- **BACKPAGE**
  Lets the operator display previous pages of detail occurrences. This value is the default.

- **NOBACKPAGE**
  Does not let the operator display any page of detail occurrences with a page number lower than the current page number. Modifications that are made on a given page of the map must be requested by MAP IN statements in the application program before a MAP OUT RESUME command is issued. The previous page of detail occurrences is deleted from the session scratch record when a new map page is displayed.

**Note:** NOBACKPAGE cannot be assigned when UPDATE and NOWAIT are specified for the session.
UPDATE
Lets the user modify variable map fields, subject to the restrictions that were specified for the map at map definition time or by statements in the program. This value is the default.

BROWSE
Lets the user modify only the page and response fields (if any) of the map. The MDTs for variable fields on the map can be set on only according to specifications that were made in the map definition or by statements in the program.

AUTODISPLAY
Enables automatic display of the first page of the pageable map. This value is the default.

NOAUTODISPLAY
Disables automatic display of the first page of the pageable map. Manually display the page by using a MAP OUT RESUME statement.

Examples

The following examples illustrate the use of the STARTPAGE statement.

Initiating a Paging Session
The following statement initiates a paging session in which the operator can page forward and backward within the pageable map but can make no modifications:

STARTPAGE SESSION EMPMAPPG NOWAIT BACKPAGE BROWSE.

Overriding Automatic Display
The following statement overrides automatic display for the first page of pageable map EMPMAPPG:

STARTPAGE SESSION EMPMAPPG NOAUTODISPLAY.

Status Codes

After completion of the STARTPAGE function, the ERROR-STATUS field in the IDMS-DC communications block indicates the outcome:

<table>
<thead>
<tr>
<th>Status code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>The request has been serviced successfully.</td>
</tr>
<tr>
<td>4604</td>
<td>A paging session was already in progress when this STARTPAGE command was received. An implied ENDPAGE was processed before this STARTPAGE executed successfully.</td>
</tr>
</tbody>
</table>

STORE (COBOL)

The STORE statement performs the following functions:
- Acquires space and a database key for a new record occurrence in the database.

- Transfers the value of the appropriate elements from program variable storage to the specified record occurrence in the database.

- Connects the new record occurrence to all sets for which it is defined as an automatic member.

Before the STORE statement is executed, the following conditions must be satisfied:

- All areas that are affected implicitly or explicitly must be readied in one of the update usage modes (see READY (COBOL) (see page 228)).

- The program must initialize all control elements (CALC and sorted set control fields).

- If the record being stored has a location mode of DIRECT, the DIRECT-DBKEY contents (positions 197-200 of the IDMS communications block (see page 35)) must be initialized with a suggested database key value or a null database key value of -1.

- If the record is to be stored in a native VSAM relative-record data set (RRDS), the DIRECT-DBKEY contents must be initialized with the relative record number that represents the storage location within the data set.

- All sets in which the record is an automatic member and the owner record for each set must be included in the subschema. Sets for which the record is a manual member need not be defined in the subschema because the STORE statement does not access those sets. (An automatic member connects to the selected set occurrence automatically when the record is stored. A manual member does not connect to the selected set occurrence automatically.)

- If the record has a location mode of VIA, currency must be established for that VIA set, regardless of whether the record is an automatic or manual member of that set. Current of the VIA set provides the suggested page for the record being stored.

- Currency must be established for all set occurrences in which the stored record will participate as an automatic member. Depending on set order, the STORE statement uses currency as follows:
  - If the record is a member of a set that is ordered FIRST or LAST, the record that is current of set establishes the set occurrence to which the new record is connected.
  - If the record is a member of a set that is ordered NEXT or PRIOR, the record that is current of set establishes the set occurrence into which the new record is connected and determines its position within the set.
  - If the record is a member of a sorted set, the record that is current of set establishes the set occurrence into which the new record is connected. The DBMS compares the sort key of the new record with the sort key of the current record of set to determine whether the new record can be inserted into the set by movement in the next direction. If it can, the current of set remains positioned at the record that is current of set and the new record is inserted. If it cannot, the DBMS finds the owner of the current of set (not necessarily the current occurrence of the owner record type) and moves as far forward in the next direction as is necessary to determine the logical insertion point for the new record.

A record is stored in the database based on the location mode that was specified in the schema definition of the record. The location modes are as follows:
For CALC, the record is placed on or near a page that is calculated from a control element (the CALC key) in the record.

For VIA, the record is placed in one of the following locations:

- As close as possible to the current of set (if current of set and member record occurrences share a common page range)

- In the same relative position in the member record’s page range as the current of set is in its associated page range (if current of set and member record occurrences do not share a common page range)

For DIRECT, the record is placed on or near a user-specified page as determined by the DIRECT-DBKEY value in the IDMS communications block. If DIRECT-DBKEY contains a valid database key for the record, the DBMS assigns a database key on the same page when space is available to the new record occurrence. Otherwise, the DBMS assigns the next available database key, subject to the page-range limits of the record being stored. If DIRECT-DBKEY contains a value of -1, the first available database key in the page range in which the record is to be stored is assigned to the record. In any case, the database key of the stored record occurrence is returned to DBKEY (positions 13-16 in the IDMS communications block). The contents of DIRECT-DBKEY remain unchanged.

**Currency:** Following successful execution of a STORE statement, the stored record becomes current of run unit, its record type, its area, and all sets in which it participates as owner or automatic member.

**Syntax**

```
STORE record-name .
```

**Parameters**

- **STORE record-name**
  
  Specifies the record type to include in the subschema. The current occurrence of the record type is moved from variable storage to the database, is connected to an occurrence of each set for which it is an automatic member, and is established as the owner of a set occurrence for each set in which it is defined as an owner.

  The ordering rules for each set govern the insertion point of the specified record in the set.

**Example**

The following figure illustrates the steps to add a new EMPLOYEE record to the database. Because EMPLOYEE is an automatic member of the DEPT-EMPLOYEE and OFFICE-EMPLOYEE sets, currency must be established in each set before the STORE is issued. The first two DML statements establish OFFICE 1 and DEPARTMENT 3100 as current of the OFFICE-EMPLOYEE and DEPT-EMPLOYEE sets, respectively. When EMPLOYEE 27 is stored, it connects to each set automatically.
After completion of the STORE function, the ERROR-STATUS field in the IDMS communications block indicates the outcome:

<table>
<thead>
<tr>
<th>Status code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>The request has been serviced successfully.</td>
</tr>
<tr>
<td>1201</td>
<td>The area in which the named record is to be stored has not been readied.</td>
</tr>
<tr>
<td>1202</td>
<td>The suggested DIRECT-DBKEY value is not within the page range for the named record.</td>
</tr>
<tr>
<td>1203</td>
<td></td>
</tr>
<tr>
<td>Status code</td>
<td>Meaning</td>
</tr>
<tr>
<td>-------------</td>
<td>---------</td>
</tr>
<tr>
<td>1204</td>
<td>The OCCURS DEPENDING ON item is less than zero or greater than the maximum number of occurrences of the control element.</td>
</tr>
<tr>
<td>1205</td>
<td>Storage of the record would violate a duplicates-not-allowed option for a CALC record, a sorted set, or an index set.</td>
</tr>
<tr>
<td>1208</td>
<td>The named record is not in the subschema. The program has probably invoked the wrong subschema.</td>
</tr>
<tr>
<td>1209</td>
<td>The record area has not been readied in one of the update usage modes.</td>
</tr>
<tr>
<td>1210</td>
<td>The subschema specifies an access restriction that prohibits storage of the named record.</td>
</tr>
<tr>
<td>1211</td>
<td>The record cannot be stored in the area because of insufficient space.</td>
</tr>
<tr>
<td>1212</td>
<td>The record cannot be stored because no database key is available. This error is a system internal error.</td>
</tr>
<tr>
<td>1218</td>
<td>The record has not been bound.</td>
</tr>
<tr>
<td>1221</td>
<td>An area other than the area of the named record occurrence has been readied with an incorrect usage mode.</td>
</tr>
<tr>
<td>1225</td>
<td>A set occurrence has not been established for each set in which the named record is to be stored.</td>
</tr>
<tr>
<td>1233</td>
<td>At least one set in which the record participates as an automatic member has not been included in the subschema.</td>
</tr>
<tr>
<td>1253</td>
<td>The subschema definition of an indexed set does not match the physical structure of the indexed set in the database.</td>
</tr>
<tr>
<td>1254</td>
<td>The prefix length of an SR51 record is less than zero or the data length is less than or equal to zero.</td>
</tr>
<tr>
<td>1255</td>
<td>An invalid length has been defined for a variable length record.</td>
</tr>
<tr>
<td>1260</td>
<td>A record occurrence that was encountered while connecting automatic sets is inconsistent with the set that was named in the ERROR-SET field of the IDMS communications block. Probable causes include a broken chain and improper database description.</td>
</tr>
<tr>
<td>1261</td>
<td>The record cannot be stored because of broken chains in the database.</td>
</tr>
</tbody>
</table>

**STORE (LRF) (COBOL)**

The STORE statement updates the database with field values for a logical-record occurrence. STORE does not necessarily result in storing new occurrences of any database records that participate in the logical record. The path that is selected to service a STORE logical-record request performs whatever database access operations the DBA has specified to service the request. For example, if an existing department gets a new employee, only the new employee information is stored in the database. The department information is not stored in the database because it already exists.
LRF uses field values from the variable storage location that is reserved for the logical record to make the appropriate updates to the database. You can optionally name an alternative storage location from which to obtain the new field values and perform the requested store operation.

**Syntax**

```
STORE logical-record-name
FROM alt-logical-record-location
WHERE boolean-expression
ON path-status imperative-statement
```

**Parameters**

- **STORE logical-record-name**
  Specifies a logical record that is defined in the subschema.

- **FROM alt-logical-record-location**
  Specifies an alternative variable storage location that contains the field values to use to make appropriate database updates. Specify a record location that is defined in the WORKING-STORAGE SECTION or LINKAGE SECTION. When you are storing a logical record that was previously retrieved into an alternative variable storage location, use the FROM clause to name the same area that was specified in the OBTAIN request.

- **WHERE boolean-expression**
  Specifies the selection criteria to apply to the object logical record. For details about coding this clause, see Logical-Record Clauses (COBOL) (see page 272).

- **ON**
  Specifies the action to take depending on the value that is returned to the LR-STATUS field in the LRC block. For details about coding this clause, see Logical-Record Clauses (COBOL) (see page 272).

  - **path-status**
    Specifies which LR-STATUS value in the LRC block triggers the specified action.

  - **imperative-statement**
    Specifies the action to take.

**Example**

The following example stores a new EMP-INSURANCE-LR for a given employee:

```
MOVE EMP-ID-IN TO  EMP-ID-0415.
MOVE  INS-PLAN-IN  TO INS-PLAN-CODE-0435.
MOVE S-DATE-IN TO  SELECTION-DATE-0400
MOVE T-DATE-IN TO  TERMINATION-DATE-0400
MOVE TYPE-IN  TO TYPE-0400
MOVE PLAN-IN TO  INS-PLAN-CODE-0400.
STORE EMP-INSURANCE-LR.
```
The following figure illustrates the new occurrence of the EMP-INSURANCE-LR, which consists of EMPLOYEE 149, INS-PLAN 001, and COVERAGE 'D'. The bottom COVERAGE occurrence represents the only data that is physically added to the database:

```
<table>
<thead>
<tr>
<th>EMPLOYEE</th>
<th>INS-PLAN</th>
<th>COVERAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>149</td>
<td>002</td>
<td>M</td>
</tr>
<tr>
<td>149</td>
<td>002</td>
<td>F</td>
</tr>
<tr>
<td>149</td>
<td>001</td>
<td>(D)</td>
</tr>
</tbody>
</table>
```

TRANSFER CONTROL (COBOL)

The TRANSFER CONTROL statement is used to perform the following actions:

- Establish linkage with a specified program and pass control and an optional parameter list to that program. The program that issues the TRANSFER CONTROL RETURN request expects return of control at the instruction immediately following the TRANSFER CONTROL statement when the linked program terminates or issues a DC RETURN request.

- Transfer control and an optional parameter list to a specified program. The program that issues the TRANSFER CONTROL NORETURN request does not expect return of control.

Syntax

```
TRANSFER CONTROL to program
```

Parameters

- **TRANSFER CONTROL to program**
  Specifies the symbolic name of a user-defined field that contains the program name, or the name itself enclosed in quotation marks.

- **RETURN (LINK)**
  Establishes linkage with the specified program, expecting return of control. RETURN and LINK are synonyms and can be used interchangeably.

- **NORETURN (XCTL)**
  Transfers control to the specified program, not expecting return of control. This value is the default. NORETURN and XCTL are synonyms and can be used interchangeably.
• **USING parameter**
  Passes one or more parameters (data items) to the program that is receiving control. Specify the symbolic name of a user-defined field that contains the names of the data items to pass. Separate multiple parameter specifications with a blank.
  If USING is specified with RETURN, the data items being passed are defined in the WORKING-STORAGE SECTION or the LINKAGE SECTION of the calling program, and in the LINKAGE SECTION of the linked program.
  If USING is specified with NORETURN, the data items being passed are defined in the LINKAGE SECTION of both the calling program and the program receiving control.
  In either case, the program that is receiving control must have a corresponding USING clause and parameter list as part of its PROCEDURE DIVISION header.

**Examples**

The following examples illustrate the use of the TRANSFER CONTROL statement.

• **Example 1**
  The following statement transfers control to the program in the PROGRAM-NAME field. The issuing program expects return of control:

  ```
  TRANSFER CONTROL TO PROGRAM-NAME
  LINK.
  ```

• **Example 2**
  The following statement transfers control to PROGRAMD and passes three data items (FIELD-1, FIELD-2, and FIELD-3) to the program. The issuing program does not expect return of control:

  ```
  TRANSFER CONTROL TO 'PROGRAMD'
  NORETURN
  USING FIELD-1 FIELD-2 FIELD-3.
  ```

**Status Codes**

After completion of the TRANSFER CONTROL function, the ERROR-STATUS field in the IDMS-DC communications block indicates the outcome:

<table>
<thead>
<tr>
<th>Status code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>The request has been serviced successfully.</td>
</tr>
<tr>
<td>3020</td>
<td>The request cannot be serviced because an I/O, program-not-found, or potential deadlock error has occurred.</td>
</tr>
</tbody>
</table>
WAIT (COBOL)

The WAIT statement relinquishes control to CA IDMS, pending completion of one or more events, or
to a higher priority ready-to-run task. If control is relinquished to wait for the completion of one or
more events, an event control block (ECB) must be defined for each event. If an ECB is already posted
when the WAIT is issued, the task is redispached immediately and control does not pass to another
task.

Syntax

```
WAIT LONG EVENT ecb EVENT NAME ecb-id .
```

Parameters

- **LONG**
  Specifies that the wait is expected to be long term. This value is the default. Specify LONG for all
  waits that are expected to last a second or more (for example, terminal input).

- **SHORT**
  Specifies that the wait is expected to be short term. Specify SHORT for all waits that are expected
to last less than a second (for example, a disk I/O).

- **EVENT ecb**
  Specifies one or more ECBs upon which the task will wait. Specify the symbolic name of a user-
defined area that contains three PIC S9(8) COMP SYNC (fullword) fields. Separate multiple EVENT
  parameters with at least one blank.

- **EVENT NAME ecb-id**
  Specifies the ECB ID upon which the task will wait. Specify the symbolic name of a user-defined
  field that contains the ECB ID, or the ID itself enclosed in quotation marks. You cannot specify
  multiple EVENT NAME values.

- **REDISPATCH**
  Specifies that the issuing task wishes to relinquish control to any higher priority ready-to-run task
  before being redispached.

Example

The following example requests a short-term wait on the event PODB:

```
WAIT SHORT EVENT NAME 'PODB'.
```
Status Codes

After completion of the WAIT function, the ERROR-STATUS field in the IDMS-DC communications block indicates the outcome:

<table>
<thead>
<tr>
<th>Status code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>The request has been serviced successfully.</td>
</tr>
<tr>
<td>3101</td>
<td>Waiting on the specified ECB would cause a deadlock.</td>
</tr>
</tbody>
</table>

WRITE JOURNAL (COBOL)

The WRITE JOURNAL statement writes a task-defined record to the journal file. These records are available to user-defined exit routines during a task- or system-initiated rollback.

Syntax

```
WRITE JOURNAL
    WAIT | NOWAIT
    SPAN | NOSPAN
    FROM record-location
    TO   end-record-location
    LENGTH record-length
```

Parameters

- **WAIT**
  Specifies that the issuing task waits for completion of the physical I/O that is associated with the WRITE JOURNAL function before resuming execution. This option causes CA IDMS to write a partially filled buffer to the journal file.

- **NOWAIT**
  Specifies that the issuing task does not wait for completion of the WRITE JOURNAL function. The journal record remains in a storage buffer until a future request necessitates writing the buffer to the journal file. This value is the default.

- **SPAN**
  Writes the record across several journal file blocks, if necessary. This value is the default.

- **NOSPAN**
  Writes the record to a single journal file block. If the record is longer than the journal block, it is split. When a record is shorter than the block, based on available space in the current journal block, it is placed in the block, split across multiple blocks (SPAN), or written to a new block after the current block is written (NOSPAN). The following considerations apply when using an exit routine to retrieve journal file records during recovery:
If a WRITE JOURNAL statement with the SPAN option is issued before a failure, records can be written across several journal blocks. To retrieve these records, the exit routine is invoked once for each segment of each record to retrieve.

If a WRITE JOURNAL statement with the NOSPAN option is issued before a failure and the records that were written to the journal file are shorter than journal blocks, the exit routine is concerned with the complete records.

**Note:** In general, SPAN provides better space utilization in the journal file because it increases the average fullness of each block.

- **FROM record-location**
  Specifies the WORKING-STORAGE SECTION or LINKAGE SECTION entry of the record to write to the journal file. Specify the symbolic name of a user-defined field.

- **TO end-record-location**
  Specifies the end of the record area to write to the journal file. Specify the symbolic name of a user-defined dummy byte field or a field containing a data item that is not associated with the record being written to the journal file.

- **LENGTH record-length**
  Specifies the length in bytes of the record to write to the journal file. Specify the symbolic name of the user-defined field that contains the length, or the length itself expressed as a numeric constant.

**Example**

The following statement writes the JOURNAL-DATA record to the journal file, spanning it across several blocks if necessary:

```
WRITE JOURNAL SPAN
  FROM JOURNAL-DATA TO END-JOURNAL-DATA.
```

**Status Codes**

After completion of the WRITE JOURNAL function, the ERROR-STATUS field in the IDMS-DC communications block indicates the outcome:

<table>
<thead>
<tr>
<th>Status code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>The request has been serviced successfully.</td>
</tr>
<tr>
<td>5002</td>
<td>Storage is not available for the required control blocks.</td>
</tr>
<tr>
<td>5032</td>
<td>The derived length of the specified journal record is zero or negative.</td>
</tr>
<tr>
<td>5097</td>
<td>An invalid status has been received from DBIO/DBMS. See the DC system log for details.</td>
</tr>
</tbody>
</table>
WRITE LINE TO TERMINAL (COBOL)

The WRITE LINE TO TERMINAL statement transfers data from program variable storage to a terminal. This statement also establishes, modifies, and deletes page header lines.

WRITE LINE TO TERMINAL statements can request data transfers that are synchronous or asynchronous:

- For a synchronous request, control passes to CA IDMS, which places the issuing task in an inactive state. For non-3270 devices, control does not return to the issuing program until the WRITE LINE TO TERMINAL request is complete. For 3270-type devices, all lines of output are saved in a buffer. The buffer is not transmitted to the terminal until it is full. The transfer of a line to the buffer causes a processing delay. However, control returns to the program immediately following the request. If the line of data fills the buffer, the entire page of data must be transmitted to the terminal. In this case, control does not return to the issuing program until the user responds by pressing ENTER. Thus, the program is made conversational.

- For an asynchronous request, control returns immediately to the issuing program. Thereafter, each time the program issues a line mode I/O request, CA IDMS automatically determines whether the last asynchronous request has completed and a new data transfer can be initiated. With asynchronous requests, programs can buffer all required pages of output without suspending task execution during the actual transmission of data. However, the task can optionally terminate itself, thereby freeing resources and allowing the user to review the buffered output.

I/O requests are processed in the sequence that they are received from the task. If a program issues a synchronous WRITE LINE TO TERMINAL request after issuing one or more asynchronous requests, all I/O requests are completed before returning control to the issuing program.

The WRITE LINE TO TERMINAL request that CA IDMS automatically issues to empty partially filled buffers upon task completion is synchronous. Therefore, the user can view all screens and can catch up with processing then. If an application lets the user interrupt or terminate processing within a task, a synchronous WRITE LINE TO TERMINAL request must be issued to suspend processing while awaiting an operator response.

Syntax

```
WRITE_LINE_TO_TERMINAL
  [WAIT | NOWAIT]
  [NEWPAGE | ERASE | NOBACKPAGE]
  [FROM output-data-location end-output-data-location]
  [TO end-output-data-location]
  [LENGTH output-data-length]
  [HEADER header-line-number]
```

15-Jan-2018
Parameters

- **WAIT**
  Specifies a synchronous write operation. The issuing task automatically relinquishes control and must wait for completion of the output operation before processing can continue. This value is the default.

- **NOWAIT**
  Specifies an asynchronous write operation. The issuing task continues executing.

- **NEWPAGE (ERASE)**
  Writes the output data line beginning on a new page. For 3270-type devices, the NEWPAGE option forces CA IDMS to write out the contents of the current buffer, even when the buffer is not full.
  NEWPAGE and ERASE are synonyms and can be used interchangeably.

- **NOBACKPAGE**
  Does not keep pages output in a scratch area. The user can view only the current page of output.
  NOBACKPAGE is valid only with the first I/O request in a line mode session, and only with 3270-type devices.

- **FROM output-data-location**
  Specifies the WORKING-STORAGE SECTION or LINKAGE SECTION entry of the data to transfer to the terminal device or the page header line being created, modified, or deleted. Specify the symbolic name of a user-defined field.

- **TO end-output-data-location**
  Specifies the end of the WORKING-STORAGE SECTION or LINKAGE SECTION entry that contains the output data stream. Specify the symbolic name of a user-defined dummy byte field or a field containing a data item that is not associated with the output data.

- **LENGTH output-data-length**
  Specifies the length of the output data area in bytes. Specify the symbolic name of a user-defined field that contains the length of the data area, or the length itself expressed as a numeric constant.

  **Note:** If the WRITE LINE TO TERMINAL statement is being used to delete a page header line, *output-data-length* must be zero.

- **HEADER header-line-number**
  Specifies the page header line being created, modified, or deleted. Specify the symbolic name of a user-defined field that contains the header line number, or the header line number itself expressed as a numeric constant.
Examples

The following examples illustrate the use of the WRITE LINE TO TERMINAL statement.

- **Example 1**
  The following statement defines the value of a data area as a header to display at the top of each new page that is written to the terminal:

  ```
  WRITE LINE TO TERMINAL
  FROM EMPL-HEAD TO END-EMPL-HEAD
  HEADER 1.
  ```

- **Example 2**
  The following statement writes the value in the specified data area to a new page on the terminal:

  ```
  WRITE LINE TO TERMINAL
  NOWAIT
  FROM EMPL-RPT LENGTH 60.
  ```

Status Codes

After completion of the WRITE LINE TO TERMINAL function, the ERROR-STATUS field in the IDMS-DC communications block indicates the outcome:

<table>
<thead>
<tr>
<th>Status code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>The request has been serviced successfully.</td>
</tr>
<tr>
<td>4707</td>
<td>A logical or permanent I/O error has occurred during processing.</td>
</tr>
<tr>
<td>4731</td>
<td>The line request block (LRB) contains an invalid field, indicating a possible error in the program parameters.</td>
</tr>
<tr>
<td>4732</td>
<td>The derived length of the specified line output area is zero or negative.</td>
</tr>
<tr>
<td>4738</td>
<td>The specified 01-level LINKAGE SECTION entry has not been allocated as required. A GET STORAGE request must be issued.</td>
</tr>
<tr>
<td>4743</td>
<td>The line I/O session has been canceled. The user has pressed CLEAR (3270s), ATTENTION (2741s), or BREAK (teletypes).</td>
</tr>
</tbody>
</table>

WRITE LOG (COBOL)

The WRITE LOG statement retrieves a predefined message from the message area of the data dictionary and optionally writes the message to a specified location in program variable storage. Retrieved messages are sent to the destination that is specified in the message definition. Typical destinations are the operator console and the DC system log file. If the operator console has been defined as the message destination, the WRITE LOG statement can request a reply. When a reply is requested, control is not returned to the issuing task until the reply is received.
The message ID that is specified in the WRITE LOG statement is a seven-digit number. The first six (most significant) digits make up the actual message ID that is used to retrieve the message from the data dictionary. The seventh digit is a severity code. This severity code is predefined in the dictionary and is retrieved with the message text to indicate the action to take after the message is written to the log:

<table>
<thead>
<tr>
<th>Severity Level</th>
<th>CA IDMS Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Return control to the issuing program and continue processing.</td>
</tr>
<tr>
<td>1</td>
<td>Snap all task resources and return control to the issuing program.</td>
</tr>
<tr>
<td>2</td>
<td>Snap all system areas and return control to the issuing program.</td>
</tr>
<tr>
<td>3</td>
<td>Snap all task resources and abend the task with a task abend code of D002.</td>
</tr>
<tr>
<td>4</td>
<td>Snap all system areas and abend the task with a task abend code of D002.</td>
</tr>
<tr>
<td>5</td>
<td>Terminate the task with a task abend code of D002.</td>
</tr>
<tr>
<td>6</td>
<td>Undefined</td>
</tr>
<tr>
<td>7</td>
<td>Undefined</td>
</tr>
<tr>
<td>8</td>
<td>Snap all system areas and abend the system with a system abend code of 3996.</td>
</tr>
<tr>
<td>9</td>
<td>Terminate the system with a system abend code of 3996.</td>
</tr>
</tbody>
</table>

If a WRITE LOG statement specifies a message ID that is not in the data dictionary, CA IDMS uses a prototype message and performs the action that is associated with the severity code in the WRITE LOG request.

Messages that are stored in the data dictionary can contain symbolic parameters. Symbolic parameters are identified by an ampersand (&). and are followed by a two-digit numeric identifier. Symbolic parameters can appear in any order within the message. The WRITE LOG statement can specify replacement values for one or more symbolic parameters. However, the position of replacement values within the WRITE LOG request must correspond exactly with the two-digit numeric identifier in the message text. For example, the first value that is specified corresponds to &01., the second to &02., and so forth.

**Syntax**

```
WRITE LOG MESSAGE ID message-id
PARMS FROM parameter-location TO end-parameter-location LENGTH parameter-length
REPLY INTO reply-location TO end-reply-location MAX LENGTH reply-max-length
MESSAGE PREFIX is 'DC'
```
Parameters

- **WRITE LOG MESSAGE ID** `message-id`
  Specifies the message ID. The first six digits specify the message ID. The seventh digit specifies the message severity code. Specify the symbolic name of a user-defined PIC S9(8) COMP (fullword) field that contains the message ID, or the ID itself expressed as a numeric constant. Message IDs 000001 through 900000 are reserved for use by CA IDMS. The WRITE LOG statement can specify any number in the range 900001 through 999999.

  **Important!** The message length must be seven digits. The system always interprets the last digit as the severity level. If you request message 987659 and you do not code a severity level of zero (that is, 9876590) you are actually requesting that message 098765 be written to the log and that the system terminate with a 3996 abend code.

When messages are added to the data dictionary for use with the WRITE LOG statement, they are assigned an eight-character identification number. The first two characters are DC. A request for message 987654 retrieves DC987654.

- **PARMS FROM** `parameter-location`
  Supplies replacement values for one or more symbolic parameters that are stored with the message text. Specify the symbolic name of a user-defined field that contains the WORKING-STORAGE SECTION or LINKAGE SECTION entry of the replacement parameter. This field must begin with a 1-byte field into which the system places the length of the adjacent field. The value in the length does not include the length byte.

- **TO** `end-parameter-location`
  Specifies the end of the WORKING-STORAGE SECTION or LINKAGE SECTION entry that contains the replacement parameter. Specify the symbolic name of a user-defined dummy byte field or a field containing a data item that is not associated with the replacement parameter.

- **LENGTH** `parameter-length`
  Specifies the length of the replacement parameter in bytes. Specify the symbolic name of a user-defined field that contains the length, or the length itself expressed as a numeric constant.

- **REPLY INTO** `reply-location`
  Specifies the WORKING-STORAGE SECTION or LINKAGE SECTION entry of the area that is reserved for a reply to the message that the WRITE LOG request issues. Specify the symbolic name of a user-defined field. This field must begin with a 1-byte field into which the system places the length of the adjacent field. The value in the length does not include the length byte.
- **TO end-reply-location**
  Specifies the end of the WORKING-STORAGE SECTION or LINKAGE SECTION entry that is reserved for the reply. Specify the symbolic name of a user-defined dummy byte field or a field containing a data item that is not associated with the reply.

- **MAX LENGTH reply-max-length**
  Specifies the maximum length, in bytes, of the area that is reserved for the reply. Specify the symbolic name of a user-defined field that contains the length, or the length itself expressed as a numeric constant.

- **MESSAGE PREFIX IS 'DC'**
  Specifies the two characters that precede the numeric position of a message. The default is 'DC'.

- **TEXT INTO text-return-location**
  Writes the contents of the named message and any replacement parameters to the issuing program. Specify the symbolic name of a user-defined 1 through 132 character alphanumeric field that contains the WORKING-STORAGE SECTION or LINKAGE SECTION entry to which the message text is to be returned. This field must begin with a 1-byte field into which the system places the length of the adjacent field. The value in the length does not include the length byte.

- **TO end-text-return-location**
  Specifies the end of the WORKING-STORAGE SECTION or LINKAGE SECTION entry that is reserved for the text. Specify the symbolic name of a user-defined dummy byte field or a field containing a data item that is not associated with the returned text.

- **MAX LENGTH text-max-length**
  Specifies the maximum length, in bytes, of the WORKING-STORAGE SECTION or LINKAGE SECTION entry that is reserved for the returned message text. Specify the symbolic name of a user-defined field that contains the text length, or the length itself expressed as a numeric constant.

- **MESSAGE PREFIX is**
  Specifies the format of the message prefix. To override the default DC prefix, specify any one or two characters for message-prefix. To suppress a prefix, specify blanks.

  - **YES**
    Indicates that the message text is preceded by:
    IDMS ppnnnnnnn Vssssss REPLYnn

    - **pp** is the prefix that is specified in the MESSAGE PREFIX parameter.
    - **nnnnnnn** is the message number.
    - **Vssssss** is the system number.
    - **REPLYnn** is the system-supplied reply number for the message (present only if the REPLY parameter is used).
    
    This value is the default.

  - **NO**
    Indicates that the message text is preceded by:
    ppnnnnnnn
- *pp* is the prefix that is specified in the MESSAGE PREFIX parameter.

- *nnnnnnn* is the message number.

- **TEXT is ONLY**
  Indicates that the message text is output with no prefix.

**Example**

The following figure illustrates a WRITE LOG statement that supplies three replacement parameters. Task A issues a WRITE LOG request for message 900121, specifying values to replace symbolic parameters &01., &02., and &03. which are stored with the message text. The message is sent to its destination, which is the logical terminal that is associated with the issuing task.

**MESSAGE SOURCE AS INPUT TO IDD**

```
ADD MESSAGE NAME IS DC900121
MESSAGE SEVERITY IS 0
DESTINATION IS TERMINAL
MESSAGE IS 'FLIGHT &01 FROM &02 TO &03 FULLY BOOKED'.
```

**WRITE LOG REQUEST**

```
WRITE LOG MESSAGE ID 9001210
PARMS FROM FLT-NO TO END-FLT-NO
    FROM DPT-CITY TO END-DEPT-CITY
    FROM ARV-CITY TO END-ARV-CITY.

WHERE:  FLT-NO = AAA201
         DPT-CITY = LA
         ARV-CITY = NY
```

WRITE LOG
Status Codes

After completion of the WRITE LOG function, the ERROR-STATUS field of the IDMS DC communications block indicates the outcome:

<table>
<thead>
<tr>
<th>Status code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>The request has been serviced successfully.</td>
</tr>
<tr>
<td>3623</td>
<td>No storage or resource control element (RCE) can be allocated for the specified reply area.</td>
</tr>
<tr>
<td>3624</td>
<td>The maximum number of outstanding replies has been exceeded. A maximum of 98 messages can await a reply at a given time.</td>
</tr>
<tr>
<td>3625</td>
<td>The maximum number of replacement parameters has been exceeded. A maximum of 8 replacement parameters can be used when the &quot;Message Prefix&quot; option is not used. If the &quot;Message Prefix&quot; option is used, the maximum number of replacement parameters is limited to 7.</td>
</tr>
<tr>
<td>3631</td>
<td>The parameter list is invalid.</td>
</tr>
</tbody>
</table>

WRITE PRINTER (COBOL)

The WRITE PRINTER statement transmits data from a task to a terminal that is defined to the system as a printer device during system generation. Any type of terminal can be designated as a printer; however, the terminal is usually a hardcopy device.

CA IDMS does not transmit data directly from program variable storage to the terminal. Instead, data is passed to a queue that CA IDMS maintains, and from the queue to the printer. The data stream that WRITE PRINTER passes to the queue contains only data. CA IDMS adds line and device control characters when it writes the data to the printer.

⚠️ **Note:** Native mode data streams (streams that contain device-control information and user data) can also be transmitted with a WRITE PRINTER request. This capability is useful in formatting reports for 3280-type printers.

Each line of data that is transmitted in a WRITE PRINTER request is considered a record. Each record is associated with a report in the print queue. A report consists of one or more records. A task can have up to 256 active print reports. A program can issue multiple WRITE PRINTER requests, each specifying a different report. Because the records that are associated with each report are maintained individually, records from one report are not interspersed with records from other reports when printed.

The WRITE PRINTER request can direct reports to print classes and to destinations:
During system generation, one or more print classes are associated with each terminal that is designated as a printer. For each report, the first record that WRITE PRINTER transmits to the print queue establishes the print class for that report. The report is printed on the first available printer that is assigned to the same print class.

Destinations are groups of terminals, printers, or users. If destinations have been defined during system generation, the WRITE PRINTER request can direct a report to a destination. Reports that are sent to printer destinations are printed on the first available printer for the destination, regardless of print class.

A report is printed only when that report is completed, either explicitly as part of a WRITE PRINTER request, or implicitly when the issuing task terminates.

![Note: All reports for a task end upon normal task termination, a FINISH TASK request, or a COMMIT TASK request. Queued reports are made eligible for printing.]

An abnormal task termination (abend) or a ROLLBACK TASK request cause any queued reports belonging to the task to be deleted.

**Syntax**

```
WRITE PRINTER [NEWPAGE | ERASE] [ENDRPT]
```

```
FROM message-location TO end-message-location
```

```
SCREEN CONTENTS
```

```
COPIES copy-count REPORT ID report-id
```

```
CLASS printer-class
DESTINATION printer-destination ALL
```

```
HOLD KEEP
```

**Parameters**

- **NEWPAGE (ERASE)**
  Prints the data stream beginning on a new page.
  NEWPAGE and ERASE are synonyms and can be used interchangeably.

- **ENDRPT**
  Indicates that the data stream constitutes the last record in the specified report. When ENDRPT is specified, the report can be printed before the issuing task has terminated. However, the program must issue a COMMIT TASK request to signal to print the ended report. A subsequent WRITE PRINTER request with the same report ID starts a separate report.
NATIVE
Specifies that the data stream contains device control characters. If NATIVE is not specified, the characters are automatically inserted.

FROM message-location
Specifies the WORKING-STORAGE SECTION or LINKAGE SECTION entry of the data to transmit to the print queue. Specify the symbolic name of a user-defined field.

TO end-message-location
Specifies the end of the WORKING-STORAGE SECTION or LINKAGE SECTION entry that contains the data to transmit to the print queue. Specify the symbolic name of a user-defined dummy byte field or a field containing a data item that is not associated with the output data.

LENGTH message-length
Defines the length of the data stream in bytes. Specify the symbolic name of a user-defined field that contains the length of the data, or the length itself expressed as a numeric constant.

SCREEN CONTENTS
Transmits the contents of the currently displayed screen to the print queue. This option applies only to 3270-type devices.

COPIES copy-count
Specifies the number of report copies to print. Specify the symbolic name of a user-defined field that contains the copy count, or the count itself expressed as a numeric constant. The count must be an integer from 1 through 255. The default is 1.

REPORT ID report-id
Specifies the identifier of the report to print. Specify the symbolic name of a user-defined field that contains the report ID, or the ID itself expressed as a numeric constant. Report-id must be an integer from 1 through 255. The default is 1.

CLASS printer-class
Assigns a print class to the report. Specify the symbolic name of a user-defined field that contains the print class, or the class itself expressed as a numeric constant. Valid print classes are 1 through 64; the default is 1.

DESTINATION printer-destination
Sends the report to the specified destination. Specify the symbolic name of a user-defined field that contains the destination, or the destination itself enclosed in quotation marks. The specified destination must have been defined during system generation. Specify the CLASS or DESTINATION only for the first line of each report. If no class or destination is specified, the default print class that was assigned to the physical terminal of the issuing task during system generation is used.

ALL
Prints the report on all printers belonging to the specified destination. The report is printed, one printer at a time, and saved until it has been printed on each printer that is associated with the destination.

HOLD
Holds a queued report without printing it. The report is held until a DCMT VARY REPORT report-name RELEASE command is issued at run time.
**KEEP**

Keeps the report in the print queue after it has been printed. The report can be released for printing with a DCMT VARY REPORT `report-name` RELEASE command. This option lets you print the report. A KEPT report can be deleted from the print queue manually (using a DCMT VARY REPORT `report-name` DELETE command at run time) or automatically (when the queue retention period has been exceeded).

**Examples**

The following examples illustrate the use of the WRITE PRINTER statement.

- **Example 1**
  The following statement associates the data in the specified location with report 32 in the print queue, and prints it beginning on a new page. Report 32 prints on the first terminal that is assigned to print class 3 when the program notifies CA IDMS that the report is complete or when the task terminates.

  ```plaintext
  WRITE PRINTER
  NEWPAGE
  FROM PASSGR-RPT TO END-PASSGR-RPT
  REPORT ID 32
  CLASS 3.
  ```

- **Example 2**
  The following statement prints three copies of the current screen contents on a printer that is associated with destination A, and keeps the report contents in the print queue after it has printed:

  ```plaintext
  WRITE PRINTER
  SCREEN CONTENTS
  COPIES 3
  DESTINATION 'A'
  KEEP.
  ```

**Status Codes**

After completion of the WRITE PRINTER function, the ERROR-STATUS field in the IDMS-DC communications block indicates the outcome:

<table>
<thead>
<tr>
<th>Status code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>The request has been serviced successfully.</td>
</tr>
<tr>
<td>4807</td>
<td>An I/O error has occurred while placing the record in the print queue.</td>
</tr>
<tr>
<td>4818</td>
<td>The current DC system definition contains no logical terminal-printer associations.</td>
</tr>
<tr>
<td>4821</td>
<td>The specified printer destination is undefined or is not a printer.</td>
</tr>
<tr>
<td>4831</td>
<td>The parameter list is invalid.</td>
</tr>
<tr>
<td>4832</td>
<td>The derived length of the specified printer output data area is zero or negative.</td>
</tr>
<tr>
<td>4838</td>
<td>The specified 01-level LINKAGE SECTION entry has not been allocated. A GET STORAGE request for the specified variable must be issued before the WRITE PRINTER statement.</td>
</tr>
<tr>
<td>Status Code</td>
<td>Meaning</td>
</tr>
<tr>
<td>-------------</td>
<td>---------</td>
</tr>
<tr>
<td>4845</td>
<td>A WRITE PRINTER SCREEN CONTENTS request cannot be serviced because the terminal that is associated with the issuing task is not a 3270-type device or is a remote 3270 device running under TCAM.</td>
</tr>
<tr>
<td>4846</td>
<td>A terminal I/O error has occurred.</td>
</tr>
</tbody>
</table>

**WRITE TERMINAL (COBOL)**

The WRITE TERMINAL statement requests a synchronous or asynchronous data transfer from program variable storage to the terminal buffer.

**Syntax**

```
WRITE TERMINAL
  WAIT ◄─► NOWAIT
```

**Parameters**

- **WAIT**
  Specifies a synchronous write operation. The issuing task automatically relinquishes control to CA IDMS and waits for completion of the write operation before continuing processing. This value is the default.

- **NOWAIT**
  Specifies an asynchronous write operation. The issuing task continues executing.

  **Note:** When NOWAIT is specified, the program must issue a CHECK TERMINAL (COBOL) request before performing any other I/O operation.

- **NEWPAGE (ERASE)**
  Activates the page-eject (SYSINOUT devices) or erase-write (3270-type devices) mechanism to erase the contents of a screen. If NEWPAGE is not specified, the WRITE TERMINAL request writes over the data that is displayed instead of erasing the data. NEWPAGE and ERASE are synonyms and can be used interchangeably.
- **EAU (ERASE ALL UNPROTECTED)**
  Activates the erase-all-unprotected mechanism. This option applies only to 3270-type devices. After a WRITE TERMINAL EAU function, only protected fields remain on the terminal. When EAU is specified, the FROM clause is not needed. EAU and ERASE ALL UNPROTECTED are synonyms and can be used interchangeably.

- **FREE STORAGE**
  Releases the output buffer that is associated with the data being written to the terminal. The storage area being freed must have been acquired by a `GET STORAGE (COBOL)` (see page 168) statement or the GET STORAGE option of a previously issued READ TERMINAL or WRITE THEN READ TERMINAL request. If FREE STORAGE is not specified, the storage that is associated with the output buffer is not freed until the issuing task terminates.

- **FROM `output-data-location`**
  Specifies the 01-level WORKING-STORAGE SECTION or LINKAGE SECTION entry of the output data stream. Specify the symbolic name of a user-defined field. If FREE STORAGE is specified, `output-data-location` must be an 01-level LINKAGE SECTION entry.

- **TO `end-output-data-location`**
  Specifies the end of the output data stream. Specify the symbolic name of a user-defined dummy byte field or a field containing a data item that is not associated with the output data stream. Place this syntax after the last data-item entry in `output-data-location`.

- **LENGTH `output-data-length`**
  Defines the length of the output data stream in bytes. Specify the symbolic name of a user-defined field that contains the length of the data area, or the length itself expressed as a numeric constant.

## Example

The following statement illustrates an asynchronous basic mode request to write data to the terminal from the specified location in program variable storage:

```plaintext
WRITE TERMINAL
NOWAIT
FROM TERM-LINE LENGTH 72.
```

## Status Codes

After completion of the WRITE TERMINAL function, the ERROR-STATUS field in the IDMS-DC communications block indicates the outcome:

<table>
<thead>
<tr>
<th>Status code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>The request has been serviced successfully.</td>
</tr>
<tr>
<td>4525</td>
<td>The output operation has been interrupted. The user has pressed ATTENTION or BREAK.</td>
</tr>
<tr>
<td>4526</td>
<td>A logical error (for example, an invalid control character) has been encountered in the output data stream.</td>
</tr>
<tr>
<td>Status code</td>
<td>Meaning</td>
</tr>
<tr>
<td>-------------</td>
<td>-------------------------------------------------------------------------</td>
</tr>
<tr>
<td>4527</td>
<td>A permanent I/O error has occurred during processing.</td>
</tr>
<tr>
<td>4528</td>
<td>The dial-up line for the terminal has been disconnected.</td>
</tr>
<tr>
<td>4531</td>
<td>The terminal request block (TRB) contains an invalid field, indicating a possible error in the program parameters.</td>
</tr>
<tr>
<td>4532</td>
<td>The derived length of the specified output data area is zero or negative.</td>
</tr>
<tr>
<td>4537</td>
<td>Storage for the output buffer cannot be freed. The specified data area is defined in the WORKING-STORAGE SECTION instead of the LINKAGE SECTION.</td>
</tr>
<tr>
<td>4539</td>
<td>The terminal that is associated with the issuing task is out of service.</td>
</tr>
</tbody>
</table>

**WRITE THEN READ TERMINAL (COBOL)**

The WRITE THEN READ TERMINAL statement transfers data from program variable storage to the terminal buffer. When the user has completed entering data, the statement also transfers that data back to program variable storage.

**Syntax**

```
WRITE THEN READ TERMINAL
   WAIT [NOWAIT]
   NEWPAGE | ERASE | EAU | ERASE ALL UNPROTECTED
   FROM output-data-location TO end-output-data-location
   LENGTH output-data-length
   MODIFIED BUFFER FROM POSITION screen-position
   GET STORAGE
   INTO input-data-location TO end-input-data-location MAX LENGTH input-data-max-length
   RETURN LENGTH into input-data-actual-length
```

**Parameters**

- **WAIT**
  Specifies a synchronous I/O operation. The issuing task automatically relinquishes control to CA IDMS and waits for completion of the I/O operation before continuing processing. This value is the default.
- **NOWAIT**
  Specifies an asynchronous I/O operation. The issuing task continues executing.

  **Note:** When NOWAIT is specified, the program must issue a `CHECK TERMINAL (COBOL)` (see page 115) request before performing any other I/O operation.

- **NEWPAGE (ERASE)**
  Activates the page-eject (SYSINOUT devices) or erase-write (3270-type devices) mechanism to erase the contents of a screen. If NEWPAGE is not specified, the WRITE TERMINAL request writes over the displayed data instead of erasing it. NEWPAGE and ERASE are synonyms and can be used interchangeably.

- **EAU (ERASE ALL UNPROTECTED)**
  Activates the erase-all-unprotected mechanism. This option applies only to 3270-type devices. After a WRITE TERMINAL EAU function, only protected fields remain on the terminal. When EAU is specified, the FROM clause is not needed. EAU and ERASE ALL UNPROTECTED are synonyms and can be used interchangeably.

- **FREE STORAGE**
  Releases the output buffer that is associated with the data being written to the terminal. The storage area being freed must have been acquired by a `GET STORAGE (COBOL)` (see page 168) statement or the GET STORAGE option of a previously issued READ TERMINAL or WRITE THEN READ TERMINAL request. If FREE STORAGE is not specified, the storage that is associated with the output buffer is not freed until the issuing task terminates.

- **FROM `output-data-location`**
  Specifies the 01-level WORKING-STORAGE SECTION or LINKAGE SECTION entry of the output data stream. Specify the symbolic name of a user-defined field. If FREE STORAGE has been specified, `output-data-location` must be an 01-level LINKAGE SECTION entry.

- **TO `end-output-data-location`**
  Specifies the end of the output data stream. Specify the symbolic name of a user-defined dummy byte field or a field containing a data item that is not associated with the output data stream.

- **LENGTH `output-data-length`**
  Defines the length of the output data stream in bytes. Specify the symbolic name of a user-defined field that contains the length of the data stream, or the length itself expressed as a numeric constant.

- **MODIFIED**
  Reads all modified fields in the terminal buffer into program variable storage. This option applies only to 3270-type devices.

- **BUFFER**
  Executes a READ BUFFER command that reads the entire contents of the terminal buffer into the program variable storage. This option applies only to 3270-type devices.
FROM POSITION screen-position
Specifies the buffer address (screen position) at which to start the read. Specify the symbolic name of a user-defined PIC S9(8) COMP SYNC (fullword) field, or the address itself enclosed in quotation marks.

GET STORAGE
Acquires an input buffer for the data being read into the program. CA IDMS allocates the required storage when the read operation is complete. This option applies only to synchronous requests.

INTO input-data-location
Specifies the 01-level WORKING-STORAGE SECTION or LINKAGE SECTION entry of the data area that is reserved for the input data stream. Specify the symbolic name of a user-defined field. If GET STORAGE is specified, the data area must be an unallocated 01-level LINKAGE SECTION entry. If GET STORAGE is not specified, the data area must be a WORKING-STORAGE SECTION or a previously allocated LINKAGE SECTION entry.

TO end-input-data-location
Specifies the end of the data area that is reserved for the input data stream. Specify the symbolic name of a user-defined dummy byte field or a field containing a data item that is not associated with the data area.

MAX LENGTH input-data-max-length
Defines the length, in bytes, of the data area that is reserved for the input data stream. Specify the symbolic name of a user-defined field that contains the length of the data stream, or the length itself expressed as a numeric constant. If the input data stream is larger than the data area that is reserved in the WORKING-STORAGE SECTION or LINKAGE SECTION, the data stream is truncated to fit the available space.

RETURN LENGTH into input-data-actual-length
Specifies where to return the actual length of the input data stream. Specify the symbolic name of a user-defined field. If the data stream has been truncated, input-data-actual-length contains the original length before truncation.

Example
The following statement illustrates a basic mode request to write data from the program (OUTPUT-LINE) to the terminal, read the data from the terminal to the specified location (INPUT-LINE) in the program, and return the actual length of the input data stream (LINE-LENGTH) to variable storage:

WRITE THEN READ TERMINAL
WAIT
FROM OUTPUT-LINE TO END-INPUT-LINE
INTO INPUT-LINE MAX LENGTH 80
RETURN LENGTH INTO LINE-LENGTH.

Status Codes
After completion of the WRITE THEN READ TERMINAL function, the ERROR-STATUS field in the IDMS-DC communications block indicates the outcome:
### Logical-Record Clauses (COBOL)

Logical-record clauses are used with the DML statements that access logical records (OBTAIN, MODIFY, STORE, and ERASE). The logical-record clauses are as follows:

- **WHERE** specifies criteria to select or limit the selection of logical-record occurrences.
- **ON** tests for a specific path status returned to indicate the result of a logical-record DML statement.

## WHERE

The WHERE clause has two major functions:

- **To direct the program to a path** that was predefined in the subschema by the DBA and transparent to the application program. This function lets you access the database without issuing specific instructions for navigating the database. LRF automatically picks the most appropriate path to service the request efficiently.
To specify the selection criteria to apply to a logical record. This function lets the program specify attributes of the desired logical record, which reduces the need for the program to inspect multiple logical records to isolate the logical record of interest.

The WHERE clause is issued in the form of a Boolean expression consisting of comparisons and keywords that are connected by Boolean operators (AND, OR, and NOT). The format of the WHERE clause follows COBOL syntax rules (that is, operands or operators are separated by a blank).

**Syntax**

```
WHERE NOT dba-designated-keyword
    comparison
    AND OR NOT dba-designated-keyword
```

**Parameters**

- **dba-designated-keyword**
  Specifies a DBA-designated keyword to apply to the logical record. This value is a DBA-specified keyword that is applicable to the logical record. The keyword cannot exceed 32 characters. The keyword represents an operation to perform at the path level and serves only to route the logical-record request to the appropriate, predetermined path.
  A path must exist to service a request that issues `dba-designated-keyword`. If no such path exists, the precompiler flags this condition by issuing an error message.

- **comparison**
  Specifies a comparison operation to perform, using the indicated operands and operators. This value also helps direct the logical-record request to a path.
  Connect individual comparisons and keywords by the Boolean operators AND, OR, and NOT. Use parentheses to clarify a multiple-comparison Boolean expression or to override the precedence of operators.

**Expansion of comparison**

```
'logical-record-field-name'
idd-defined-variable-field-name
logical-record-field-name
arithmetic-expression
CONTAINS MATCHES EQ = NE GT > LT < GE LE
```

OF group-id
OF LR
Parameters

- **logical-record-field-name**
  Specifies a data field that participates in the named logical record.

- **CONTAINS|MATCHES|EQ|NE|GT|LT|GE|LE**
  Specifies the comparison operator:
  - CONTAINS
    Is true when the value of the right operand occurs in the value of the left operand. Both operands in the CONTAINS operator must be alphanumeric values.
  - MATCHES
    Is true when each character in the left operand matches a corresponding character in the right operand (the mask). When MATCHES is specified, LRF compares the left operand with the mask, one character at a time, moving from left to right. The result of the match is true or false. The result is true when the end of the mask is reached before encountering a character in the left operand that does not match a corresponding character in the mask. The result is false when LRF encounters a character in the left operand that does not match a mask character. Three special characters can be used in the mask to perform pattern matching: @, which matches any alphabetic character; #, which matches any numeric character; and *, which matches any alphabetic or numeric character. The left operand and the mask must be alphanumeric values and elementary elements.
  - EQ
    Is true when the value of the left operand is equal to the value of the right operand.
  - NE
    Is true when the value of the left operand is not equal to the value of the right operand.
  - GT
    Is true when the value of the left operand is greater than the value of the right operand.
  - LT
    Is true when the value of the left operand is less than the value of the right operand.
  - GE
    Is true when the value of the left operand is greater than or equal to the value of the right operand.
  - LE
    Is true when the value of the left operand is less than or equal to the value of the right operand.
  - 'literal'
    Specifies any alphanumeric or numeric literal. Alphanumeric literals must be enclosed in quotation marks.
  - **idd-defined-variable-field-name**
    Specifies the name of a program variable storage field that is predefined in the data dictionary.
• **OF group-id**
  Uniquely identifies the named variable field. Specify the name of the group element that contains the field. This qualifier is required when *idd-defined-variable-field-name* is not unique within program variable storage.
  You can specify a maximum of 15 different **OF group-id** qualifiers to identify up to 15 levels of group elements.

• **logical-record-field-name**
  Specifies a data field that participates in the named logical record.

• **OF group-id**
  Uniquely identifies the named logical-record field. Specify the name of the group element or database record that contains the field. This qualifier is required when *logical-record-field-name* is not unique within all subschema records, including those that are not part of the logical record, and including all non-CA IDMS database records that are copied into the program.
  You can specify a maximum of 15 different **OF group-id** qualifiers to identify up to 15 levels of group elements.

• **OF LR**
  Uses the value of the named field at the time the request is issued throughout processing of the request. If the field value changes during request processing, LRF continues to use the original value.
  If you do not specify **OF LR**, the field value changes during request processing, and the field is required for further processing, the new value is used.

• **arithmetic-expression**
  Specifies an arithmetic expression that is designated as a unary minus (-), unary plus (+), simple arithmetic operation, or compound arithmetic operation. The following arithmetic operators are permitted in an arithmetic expression: add (+), subtract (-), multiply (*), and divide (/). Operands can be literals, variable-storage fields, and logical-record fields.

  If the WHERE clause compares a CALC-key field to a literal, the literal format must correspond exactly to the CALC-key definition. Enclose the literal in quotation marks when the CALC key has a usage of DISPLAY. Use leading zeros when the literal consists of fewer characters than the field picture. For example, if the *calc-key-field* CALC key is defined as PIC 9(3) USAGE DISPLAY, code the WHERE clause as follows:

  ```
  WHERE calc-key-field EQ '054'
  ```

  The WHERE clause can contain as many comparisons and keywords as required to specify the criteria to apply to the logical record. The SIZE value on the COPY IDMS SUBSCHEMA-LR-CTRL statement can be increased to accommodate large and complex WHERE clause specifications. The composition of the WHERE clause does not affect processing efficiency (other than the logical order of the operators) because LRF automatically uses the most efficient path to process the logical-record request.

  The operators in a WHERE clause are evaluated in the following order:

  1. Comparisons that are enclosed in parentheses

  2. Arithmetic, comparison, and Boolean operators by order of precedence, from highest to lowest:

   a. Unary plus or minus in an arithmetic expression
2. From left to right within operators of equal precedence

Examples

The following examples illustrate the use of the WHERE clause.

- **Example 1**
  The following logical-record request uses a DBA-designated keyword (PROGRAMMER-ANALYSTS) to direct LRF to a DBA-defined access path:

  ```
  OBTAIN NEXT EMP-JOB-LR
  WHERE PROGRAMMER-ANALYSTS.
  ```

- **Example 2**
  The following logical-record request uses Boolean selection criteria to specify the desired occurrence of EMP-JOB-LR:

  ```
  OBTAIN EMP-JOB-LR
  WHERE OFFICE-CODE-0450 EQ '001'.
  ```

ON

The ON clause tests for a specific path status returned to indicate the result of the statement. If LRF returns the specified path status, the imperative statement in the ON clause is executed. If the specified path status is not returned, the imperative statement is ignored and IDMS-STATUS is performed.

If the DML statement with the ON clause is the object of a PERFORM, use the THROUGH option of the PERFORM statement to avoid scope problems.

A logical-record DML statement can include an ON clause only when the AUTOSTATUS protocol is in effect for the program. AUTOSTATUS automatically invokes an error-checking routine after every DML statement except IF. For more information, see Error Detection (see page ).

The ON clause tests for a standard or DBA-defined path status, which is in the form of a 1- through 16-character unquoted string. Path statuses are issued during execution of the path that was selected to service the request. The following path statuses are standard:
### LR-FOUND
Indicates that the logical-record request executed successfully. This status can be returned as the result of any of the four LRF DML statements. When LR-FOUND is returned, the ERROR-STATUS field of the IDMS communications block contains 0000.

### LR-NOT-FOUND
Indicates that the logical record cannot be found because no such record exists or because all such occurrences have already been retrieved. This status can be returned as the result of any of the four LRF DML statements, if the path to which LRF is directed includes retrieval logic. When LR-NOT-FOUND is returned, the ERROR-STATUS field of the IDMS communications block contains 0000.

**Note:** A successful STORE can return LR-NOT-FOUND when its WHERE clause references a logical-record field and the STORE path performs no OBTAINs.

### LR-ERROR
Indicates that a logical-record request was issued incorrectly or an error occurred in the processing of the path that was selected to service the request. When LR-ERROR is returned, the status code that is returned to the program in the ERROR-STATUS field of the IDMS communications block differs according to the type of error:

- When the error occurs in the **logical-record request**, the ERROR-STATUS field contains a status code that is issued by LRF (major code of 20).

- When an error occurs in the **logical-record path processing**, the ERROR-STATUS field contains a status code that is issued by the DBMS (major code from 00 to 19). For more information about status codes, see Communications Blocks and Error Detection (COBOL) (see page 35).

When you access ASF-defined data tables, always look for the following path statuses:

### INVALID-DATA
Indicates that the data violates the definition-time selection criteria (for example, WHERE STATE EQ 'MA' and the program tries to replace the state with 'NY'). When INVALID-DATA is returned, the ERROR-STATUS field in the IDMS communications block is set to 0000.

### DEFN-MISSING
Indicates that the record definition cannot be found. When DEFN-MISSING is returned, the ERROR-STATUS field in the IDMS communications block is set to 0000.

### OOAK-MISSING
Indicates that a one-of-a-kind record cannot be found. When OOAK-MISSING is returned, the ERROR-STATUS field in the IDMS communications block is set to 0000.

### SYNC-ERROR
Indicates that the timestamp in the catalog and the table definition do not match. When SYNC-ERROR is returned, the ERROR-STATUS field in the IDMS communications block is set to 0000. This status applies to ASF tables only.

The return of any of these statuses indicates a fatal error. For more information, consult your DBA.
Syntax

```
ON path-status imperative-statement
```

Parameters

- **ON**
  Tests for a path status returned as the result of the logical-record request that the program issued.

- **path-status**
  Specifies a 1- through 16-character alphanumeric value.

- **imperative-statement**
  Specifies the program action to take when the indicated path status results from the logical-record request.

Examples

The following statements use the path status LR-NOT-FOUND in two different ways. If LR-NOT-FOUND occurs following the initial statement, an LR-MISSING message is issued. If LR-NOT-FOUND occurs in subsequent statements, an END-OF-LR message is issued.

```
OBTAIN-FIRST-LR.
OBTAIN FIRST EMP-JOB-LR
  WHERE OFFICE-CODE-450 EQ OFFICE-CODE-IN
  ON LR-NOT-FOUND
    GO TO LR-MISSING.
  .
  .

OBTAIN-REST-LR.
OBTAIN NEXT EMP-JOB-LR
  WHERE OFFICE-CODE-0450 EQ OFFICE-CODE-IN
  ON LR-NOT-FOUND
    GO TO END-OF-LR.
  .
  .

GO TO OBTAIN-REST-LR.
```

Status Codes

The following codes are returned to the ERROR-STATUS field in the IDMS or IDMS-DC communications block when an LR-ERROR path status is returned to the LR-STATUS field in the LRC block:

<table>
<thead>
<tr>
<th>Status code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>2001</td>
<td>The requested logical record was not found in the subschema. (The path DML statement, EVALUATE, returns 0000 when true, and 2001 when false.)</td>
</tr>
<tr>
<td>2008</td>
<td>The named record is not in the subschema, or the specified request is not permitted for the named record.</td>
</tr>
<tr>
<td>2010</td>
<td>The subschema prohibits access to logical records.</td>
</tr>
<tr>
<td>Status code</td>
<td>Meaning</td>
</tr>
<tr>
<td>-------------</td>
<td>---------</td>
</tr>
<tr>
<td>2018</td>
<td>A path command has attempted to access a database record that has not been bound.</td>
</tr>
<tr>
<td>2040</td>
<td>The WHERE clause in an OBTAIN NEXT command directed LRF to a different processing path than the WHERE clause in the preceding OBTAIN command for the same logical record.</td>
</tr>
<tr>
<td>2041</td>
<td>The WHERE clause cannot be matched to a path in the subschema.</td>
</tr>
<tr>
<td>2042</td>
<td>The logical-record path for the request specifies the return of the LR-ERROR status.</td>
</tr>
<tr>
<td>2043</td>
<td>Bad or inconsistent data was encountered in the logical-record buffer during evaluation of the WHERE clause.</td>
</tr>
<tr>
<td>2044</td>
<td>The WHERE clause does not include data that the logical-record path requires.</td>
</tr>
<tr>
<td>2045</td>
<td>A subscript value in a WHERE clause is less than zero or greater than its maximum allowed value.</td>
</tr>
<tr>
<td>2046</td>
<td>A program check has revealed an arithmetic exception (for example, overflow, underflow, significance, divide) during the evaluation of a WHERE clause.</td>
</tr>
<tr>
<td>2063</td>
<td>The WHERE clause contains a keyword that exceeds the 16-character maximum.</td>
</tr>
<tr>
<td>2064</td>
<td>The path command has attempted to access a CALC data item that was not defined properly in the subschema.</td>
</tr>
<tr>
<td>2072</td>
<td>The WHERE clause is too long to be evaluated in the available work area.</td>
</tr>
</tbody>
</table>
DML Precompile, COBOL Compile, and Link-Edit JCL

This article contains the JCL used to prepare COBOL source code that contains DML statements for execution. Link-edit considerations are also discussed. Samples of z/OS, z/VSE, and z/VM JCL are included.

- Compiling a COBOL Program (see page 280)
- z/OS JCL (see page 283)
- z/VSE JCL (see page 286)
- CMS Commands (see page 294)
- Link-Edit Considerations (see page 296)
- Passing Parameters to the Precompiler (see page 296)

Compiling a COBOL Program

To compile a COBOL program under the DML precompiler:

1. Execute the program IDMSDMLC
2. Execute the COBOL compiler
3. Link edit

Input to IDMSDMLC consists of source code written in COBOL/DML, protocol/control information, and data dictionary record descriptions. Output from IDMSDMLC is as follows:

- A source COBOL program
- A DML source listing and diagnostics

Input to the COBOL compiler consists of the source program produced by IDMSDMLC. Output is as follows:

- An object program
- COBOL listings

Input to the linkage editor consists of the object program produced by the COBOL compiler. Output is as follows:

- A load module (phase)
- A link-edit map

The following figure illustrates the steps involved in compiling a COBOL program.
Compiling Programs

z/OS JCL

Sample JCL for z/OS operating systems is shown below, followed by a description of statements that need tailoring for site-specific conditions.

```plaintext
Compiling Programs

z/OS JCL

Sample JCL for z/OS operating systems is shown below, followed by a description of statements that need tailoring for site-specific conditions.

```/**
** PRECOMPILE COBOL PROGRAM **
**
**/precomp EXEC PGM=IDMSDMLC,REGION=4096K,
// PARM='precompiler-options'
//STEPLIB DD DSN=idms.dba.loadlib,DISP=SHR
// DD DSN=idms.custom.loadlib,DISP=SHR
// DD DSN=idms.cagjload,DISP=SHR
//sysctl DD DSN=idms.sysctl,DISP=SHR
//dcmsg DD DSN=idms.sysmsg.ddldcmmsg,DISP=SHR
//SYS001 DD UNIT=sysda,SPACE=(TRK,(10,10)),
// DCB=(RECFM=VB,LRECL=133,BLKSIZE=1334,DSORG=PS)
//SYS002 DD UNIT=sysda,SPACE=(TRK,(10,10)),
// DCB=(RECFM=VB,LRECL=133,BLKSIZE=1334,DSORG=PS)
//SYS003 DD UNIT=sysda,SPACE=(TRK,(10,10)),
// DCB=(RECFM=VB,LRECL=133,BLKSIZE=1334,DSORG=PS)
//SYSCH DD DSN=&&SOURCE.,DISP=(NEW,PASS),
// UNIT=sysda,SPACE=(TRK,(10,5),RLSE),
// DCB=(RECFM=FB,LRECL=80,BLKSIZE=3120)
//SYSLST DD SYSPUT=A
//SYSIDMS DD *
//DMCL=dmcl-name
//sysidms-input-parms
/*
//SYSIPT DD *
//COBOL DML source statements
*/
/**
** COMPILER COBOL PROGRAM **
**
**/cbclcmp EXEC PGM=igycrctl,REGION=4096K,
// PARM='compiler-options'
//STEPLIB DD DSN=cobol.loadlib,DISP=SHR
//SYSUT1 DD UNIT=sysda,SPACE=(TRK,(10,5))
//SYSUT2 DD UNIT=sysda,SPACE=(TRK,(10,5))
//SYSUT3 DD UNIT=sysda,SPACE=(TRK,(10,5))
//SYSUT4 DD UNIT=sysda,SPACE=(TRK,(10,5))
//SYSUT5 DD UNIT=sysda,SPACE=(TRK,(10,5))
//SYSUT6 DD UNIT=sysda,SPACE=(TRK,(10,5))
//SYSUT7 DD UNIT=sysda,SPACE=(TRK,(10,5))
//syslin DD DSN=&&OBJECT.,DISP=(NEW,PASS),
// UNIT=sysda,SPACE=(TRK,(10,5),RLSE),
// DCB=(RECFM=FB,LRECL=80,BLKSIZE=3120)
//SYSPRINT DD SYSPUT=A
//SYSIN DD DSN=&&SOURCE.,DISP=(OLD,DELETE)
CA IDMS Reference - 19.0

//*****************************************************************
//**                      LINK PROGRAM MODULE                      **
//*****************************************************************

//link EXEC PGM=HEWL,REGION=1024K,PARM='LET,LIST,MAP,XREF'
//SYSUT1 DD UNIT=sysda,SPACE=(TRK,(20,5))
//SYSLIB DD DSN=cobol.linklib,DISP=SHR
//vanilla DD DSN=idms.cagjload,DISP=SHR
//custom DD DSN=idms.custom.loadlib,DISP=SHR
//SYSLMOD DD DSN=idms.custom.loadlib,DISP=SHR
//SYSPRINT DD SYSOUT=A
//SYSLIN DD DSN=&&OBJECT.,DISP=(OLD,DELETE)
// DD *
INCLUDE vanilla(IDMS) required, except omit for CICS
INCLUDE vanilla(IDMSPANC) required for BATCH and DC-BATCH if using IDMS-STATUS module
INCLUDE custom(IDMSOPTI) optional; BATCH and DC-BATCH only
INCLUDE custom(idmscint) required for CICS, otherwise omit
ENTRY userentry
NAME userprog(R)
/*
//*

⚠️ **Note:** If using the IDMSOPTI module, you must assemble and link edit it before using the JCL above.

The link of CICS application programs that use IDMSCINT must incorporate JCL to resolve external reference DFHEI1. The particular JCL depends on the nature and language of your application. See the appropriate IBM CICS application programming documentation for details.

---

**prec** Options that control various aspects of the precompile process. See Passing Parameters to the Precompiler (see page 296) for a complete description of the options.

**ompi** Precompiler options:

- pas**

**idms** Data set name of the load library containing the DMCL and database name table database

**loadlib** idms

**idms** Data set name of the load library containing the vanilla CA IDMS executable modules

**loadlib** idms
cagjload

**idms** Data set name of the load library containing the customized CA IDMS executable modules

**loadlib** idms
custom

**sysctl** DDname of SYSCTL file

Data set name of SYSCTL file

---

15-Jan-2018
284/415
**idms**

-sysct

-**dcm** DDname of the system message (DDLDCMSG) area

-sg

**idms** Data set name of the system message (DDLDCMSG) area

-sysm

-sg.

-ddld

-ddld

**ddlcmsg**

**sysd** Symbolic device name for work files

-**sysid** Parameters that specify physical requirements of the environment, runtime directives, or operating system-dependent file information. For a complete description of all SYSIDMS parameters and syntax, see **CA IDMS Common Administering section**. Also see Passing Parameters to the Precompiler (see page 296) for a discussion of parameters that can be passed using the PARM=SYSIDMS input statement.

-**dmcl** Specifies the name of the DMCL that the precompiler should use to access the message dictionary

-**igycr** Program name of the COBOL compiler

-**com** Parameters that specify options that are appropriate to your version of the COBOL compiler. Parameters to the Precompiler (see page 296) for a discussion of parameters that can be passed using the PARM=SYSIDMS input statement.

-**cobl** Load library that contains COBOL compiler

-**loadl**

-**ib**

-**sysln** DDname of the object data set output by the COBOL compiler

-**cobl** Load library that contains COBOL support modules

-**linkli**

-b

-**user.** User application load library

-**loadl**

-**ib**

-**idms** Load module created by compiling IDMSCINT or IDMSCINL. For more information, see the **CA IDMS System Reference section**.

-Name of program entry point
**User** entry

Name of program in load library

**Note:** Depending on the central version operating environment, an IDMSOPTI module link edited with IDMSDMLC can be used in place of or in addition to the SYSCTL file.

### Local Mode JCL

To execute the compiler in local mode, remove the SYSCTL statement from the precompile step and replace it with the following:

```plaintext
//dictdb   DD DSN=idms.appldict.ddldml,DISP=SHR
//sysjrnl  DD DSN=idms.tapejrnl,DISP=(NEW,CATLG),UNIT=tape
```

<table>
<thead>
<tr>
<th><strong>dictdb</strong></th>
<th>DDName of the application dictionary DDLDDML area</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>idms.appldict.ddldml</strong></td>
<td>Data set name of application dictionary</td>
</tr>
<tr>
<td><strong>sysjrnl</strong></td>
<td>DDName of the tape journal file</td>
</tr>
<tr>
<td><strong>idms.tapejrnl</strong></td>
<td>Data set name of the tape journal file</td>
</tr>
<tr>
<td><strong>tape</strong></td>
<td>Symbolic device name of the tape journal file</td>
</tr>
</tbody>
</table>

### z/VSE JCL

**IDMSDMLC ('VSE')**

```plaintext
/*******************************************************************************/
/** PRECOMPILE PROGRAM **
/*******************************************************************************/
* step1
  // EXEC PROC=IDMSLBBLS
  // UPSI b  if specified in IDMSOPTI module// DLBL sysctl,'idms.sysctl',0
  // EXTENT SYS000,nnnnnn,ssss,llll// ASSGN SYS000,DISK,VOL=nnnnnn,SHR
  // DLBL idmspch,'temp.dmlc',0
  // EXTENT SYS020,nnnnnn,ssss,llll// ASSGN SYS020,DISK,VOL=nnnnnn,SHR
  // DLBL SYS001,'wkfile1',0
  // EXTENT SYS001,nnnnnn,ssss,llll// ASSGN SYS001,DISK,VOL=nnnnnn,SHR
  // DLBL SYS002,'wkfile2',0
  // EXTENT SYS002,nnnnnn,ssss,llll// ASSGN SYS002,DISK,VOL=nnnnnn,SHR
  // DLBL SYS003,'wkfile3',0
  // EXTENT SYS003,nnnnnn,ssss,llll// ASSGN SYS003,DISK,VOL=nnnnnn,SHR
  // EXEC IDMSDMLC,PARM='COBOL=2'
  Input SYSDMS parameters here, as required
/*
COBOL/DML source statements
*******************************************************************************/
/** COMPIL PROGRAM **
*******************************************************************************/
/*
  // DLBL IJSYSIN,'temp.dmlc',0
  // EXTENT SYSIPT,nnnnnn ASSGN SYSIPT,DISK,VOL=nnnnnn,SHR
```
INCLUDE statements should be provided in local mode or central version JCL as follows:

<p>| |
||</p>
<table>
<thead>
<tr>
<th>idms.tapejrn1</th>
<th>file ID of tape journal file</th>
</tr>
</thead>
<tbody>
<tr>
<td>f</td>
<td>file number of tape journal file</td>
</tr>
<tr>
<td>sys009</td>
<td>logical unit assignment for journal file</td>
</tr>
</tbody>
</table>

You can use SYSIDMS parameters to specify information about your runtime environment.

**Local Mode**

To execute the IDMSDMLC precompiler in local mode:

- Remove the UPSI specification, if present, or remove the JCL for the SYSCTL file from the precompiler step.
- Add the following statements in step 1 (the IDMSDMLC step):

  ```
  // TLBL   sysjrnl,'idms.tapejrn1',,nnnnnn,,f
  // ASSGN  SYS009,TAPE,VOL=nnnnnn
  ```
INCLUDE IDMS        IDMS interface
INCLUDE IDMSOPTI    IDMSOPTI module
INCLUDE IDMSCANC    Local mode abort entry point
omit IDMSCANC if TP application
INCLUDE IDMSCINT    For CICS only, replaces INCLUDE IDMS

INCLUDE IDMSOPTI can be omitted for programs executed in local mode.

Note: COBOL overlay programs must resolve references to IDMS within their root segment; care must be taken to prevent the overlaying of the IDMS interface. Use of IDMS and IDMSLDPT is recommended for these programs.

IDMSLBLS Procedure

IDMSLBLS is a procedure provided during an CA IDMS z/VSE installation. It contains file definitions for the CA IDMS components listed below. These components are provided during installation:

- Dictionaries
- Sample databases
- Disk journal files
- SYSIDMS file

Tailor the IDMSLBLS procedure to reflect the filenames and definitions in use at your site and include this procedure in z/VSE JCL job streams.

The sample z/VSE JCL provided in this document includes the IDMSLBLS procedure. Therefore, individual file definitions for CA IDMS dictionaries, sample databases, disk journal files, and SYSIDMS files are not included in the sample JCL.

IDMSLBLS procedure (z/VSE)

* --------- LIBDEFS ---------
// LIBDEF *,SEARCH=idsms.lib.sublib// LIBDEF *,CATALOG=user.sublib/* --------------
------------ LABELS --------------
// DLBL  idsmslib,'idsms.library',1999/365
// EXTENT ,nnnnnn,,ssss,1500
// DLBL  dccat,'idsms.system.dccat',1999/365,DA
// EXTENT SYSnnn,nnnnnn,,ssss,31
// ASSGN  SYSnnn,DISK,VOL=nnnnnn,SHR
// DLBL  dccatl,'idsms.system.dccatlod',1999/365,DA
// EXTENT SYSnnn,nnnnnn,,ssss,6
// ASSGN  SYSnnn,DISK,VOL=nnnnnn,SHR
// DLBL  dccatx,'idsms.system.dccatx',1999/365,DA
// EXTENT SYSnnn,nnnnnn,,ssss,11
// ASSGN  SYSnnn,DISK,VOL=nnnnnn,SHR
// DLBL  dcdml,'idsms.system.ddldml',1999/365,DA
// EXTENT SYSnnn,nnnnnn,,ssss,101
// ASSGN  SYSnnn,DISK,VOL=nnnnnn,SHR
// DLBL  dcdlod,'idsms.system.ddldclod',1999/365,DA
// EXTENT SYSnnn,nnnnnn,,ssss,21
// ASSGN SYSnnn,DISK,VOL=nnnnnn,SHR
// DLBL dclog,'idms.system.ddldclog',1999/365,DA
// EXTENT SYSnnn,nnnnnn,,ssss,401
// ASSGN SYSnnn,DISK,VOL=nnnnnn,SHR
// DLBL dcrun,'idms.system.ddldcrun',1999/365,DA
// EXTENT SYSnnn,nnnnnn,,ssss,68
// ASSGN SYSnnn,DISK,VOL=nnnnnn,SHR
// DLBL dcsr,'idms.system.ddldcsr',1999/365,DA
// EXTENT SYSnnn,nnnnnn,,ssss,201
// ASSGN SYSnnn,DISK,VOL=nnnnnn,SHR
// DLBL dclscr,'idms.sysloc.ddlocscr',1999/365,DA
// EXTENT SYSnnn,nnnnnn,,ssss,6
// ASSGN SYSnnn,DISK,VOL=nnnnnn,SHR
// DLBL dirlab,'idms.sysdirl.ddldml',1999/365,DA
// EXTENT SYSnnn,nnnnnn,,ssss,201
// ASSGN SYSnnn,DISK,VOL=nnnnnn,SHR
// DLBL dirllod,'idms.sysdirl.ddldclod',1999/365,DA
// EXTENT SYSnnn,nnnnnn,,ssss,2
// ASSGN SYSnnn,DISK,VOL=nnnnnn,SHR
// DLBL empdemo,'idms.empdemo1',1999/365,DA
// EXTENT SYSnnn,nnnnnn,,ssss,11
// ASSGN SYSnnn,DISK,VOL=nnnnnn,SHR
// DLBL insdemo,'idms.insdemo1',1999/365,DA
// EXTENT SYSnnn,nnnnnn,,ssss,6
// ASSGN SYSnnn,DISK,VOL=nnnnnn,SHR
// DLBL orgdemo,'idms.orgdemo1',1999/365,DA
// EXTENT SYSnnn,nnnnnn,,ssss,6
// ASSGN SYSnnn,DISK,VOL=nnnnnn,SHR
// DLBL empldem,'idms.sqldemo.empldemo',1999/365,DA
// EXTENT SYSnnn,nnnnnn,,ssss,11
// ASSGN SYSnnn,DISK,VOL=nnnnnn,SHR
// DLBL infodem,'idms.sqldemo.infodemo',1999/365,DA
// EXTENT SYSnnn,nnnnnn,,ssss,6
// ASSGN SYSnnn,DISK,VOL=nnnnnn,SHR
// DLBL projdem,'idms.projseg projdemo',1999/365,DA
// EXTENT SYSnnn,nnnnnn,,ssss,6
// ASSGN SYSnnn,DISK,VOL=nnnnnn,SHR
// DLBL indxdem,'idms.sqldemo.indxdemo',1999/365,DA
// EXTENT SYSnnn,nnnnnn,,ssss,6
// ASSGN SYSnnn,DISK,VOL=nnnnnn,SHR
// DLBL sysctl,'idms.sysctl',1999/365,SD
// EXTENT SYSnnn,nnnnnn,,ssss,2
// ASSGN SYSnnn,DISK,VOL=nnnnnn,SHR
// DLBL secdd,'idms.syssuser.ddlsec',1999/365,DA
// EXTENT SYSnnn,nnnnnn,,ssss,26
// ASSGN SYSnnn,DISK,VOL=nnnnnn,SHR
// DLBL dictdb,'idms.appldict.ddldml',1999/365,DA
// EXTENT SYSnnn,nnnnnn,,ssss,51
// ASSGN SYSnnn,DISK,VOL=nnnnnn,SHR
// DLBL dloddb,'idms.appldict.ddldclod',1999/365,DA
// EXTENT SYSnnn,nnnnnn,,ssss,51
// ASSGN SYSnnn,DISK,VOL=nnnnnn,SHR
// DLBL sqldd,'idms.syssql.ddlcat',1999/365,DA
// EXTENT SYSnnn,nnnnnn,,ssss,101
// ASSGN SYSnnn,DISK,VOL=nnnnnn,SHR
// DLBL sqllod,'idms.syssql.ddlcatl',1999/365,DA
// EXTENT SYSnnn,nnnnnn,,ssss,51
// ASSGN SYSnnn,DISK,VOL=nnnnnn,SHR
// DLBL sqlxd,'idms.syssql.ddlcatx',1999/365,DA
// EXTENT SYSnnn,nnnnnn,,ssss,26
// ASSGN SYSnnn,DISK,VOL=nnnnnn,SHR
// DLBL asfdml,'idms.asfdict.ddldml',1999/365,DA
// EXTENT SYSnnn,nnnnnn,,ssss,201
// ASSGN SYSnnn,DISK,VOL=nnnnnn,SHR
// DLBL asfld,'idms.asfdict.asflod',1999/365,DA
// EXTENT SYSnnn,nnnnnn,,ssss,401
// ASSGN SYSnnn,DISK,VOL=nnnnnn,SHR
// DLBL asfdata,'idms.asfdict.asfdata'
// DLBL asfdata,'idms.asfdict.asfdata',1999/365,DA
// EXTENT SYSnnn,nnnnnn,,ssss,201
// ASSGN SYSnnn,DISK,VOL=nnnnnn,SHR
// DLBL ASFDEFN,'idms.asfdict.asfdefn',1999/365,DA
// EXTENT SYSnnn,nnnnnn,,ssss,101
// ASSGN SYSnnn,DISK,VOL=nnnnnn,SHR
// DLBL j1jrnl,'idms.j1jrnl',1999/365,DA
// EXTENT SYSnnn,nnnnnn,,ssss,54
// ASSGN SYSnnn,DISK,VOL=nnnnnn,SHR
// DLBL j2jrnl,'idms.j2jrnl',1999/365,DA
// EXTENT SYSnnn,nnnnnn,,ssss,54
// ASSGN SYSnnn,DISK,VOL=nnnnnn,SHR
// DLBL j3jrnl,'idms.j3jrnl',1999/365,DA
// EXTENT SYSnnn,nnnnnn,,ssss,54
// ASSGN SYSnnn,DISK,VOL=nnnnnn,SHR
// DLBL SYSIDMS,'#SYSIPT',0,SD
/+/*

<table>
<thead>
<tr>
<th>idmslib.sublib</th>
<th>name of the sublibrary within the library containing CA IDMS modules</th>
</tr>
</thead>
<tbody>
<tr>
<td>user.sublib</td>
<td>name of the sublibrary within the library containing user modules</td>
</tr>
<tr>
<td>idmslib</td>
<td>filename of the file containing CA IDMS modules</td>
</tr>
<tr>
<td>idms.library</td>
<td>file-ID associated with the file containing CA IDMS modules</td>
</tr>
<tr>
<td>SYSnnn</td>
<td>logical unit of the volume for which the extent is effective</td>
</tr>
<tr>
<td>nnnnnn</td>
<td>volume serial identifier of appropriate disk volume</td>
</tr>
<tr>
<td>ssss</td>
<td>starting track (CKD) or block (FBA) of disk extent</td>
</tr>
<tr>
<td>dccat</td>
<td>filename of the system dictionary catalog (DDLCAT) area</td>
</tr>
<tr>
<td>idms.system.dccat</td>
<td>file-ID of the system dictionary catalog (DDLCAT) area</td>
</tr>
<tr>
<td>dccatl</td>
<td>filename of the system dictionary catalog load (DDLCATLOD) area</td>
</tr>
<tr>
<td>idms.system.dccatlod</td>
<td>file-ID of the system dictionary catalog load (DDLCATLOD) area</td>
</tr>
<tr>
<td>dcatx</td>
<td>filename of the system dictionary catalog index (DDLCATX) area</td>
</tr>
<tr>
<td>idms.system.dccatx</td>
<td>file-ID of the system dictionary catalog index (DDLCATX) area</td>
</tr>
<tr>
<td>dcdml</td>
<td>filename of the system dictionary definition (DDLDML) area</td>
</tr>
<tr>
<td>idms.system.ddldml</td>
<td>file-ID of the system dictionary definition (DDLDML) area</td>
</tr>
<tr>
<td>dclog</td>
<td>filename of the system log area (DDLCLOG) area</td>
</tr>
<tr>
<td>idms.system.ddldclog</td>
<td>file-ID of the system log (DDLCLOG) area</td>
</tr>
<tr>
<td>dcrun</td>
<td>filename of the system queue (DDLCRUN) area</td>
</tr>
<tr>
<td>idms.system.ddldcrun</td>
<td>file-ID of the system queue (DDLCRUN) area</td>
</tr>
<tr>
<td>dcscr</td>
<td>filename of the system scratch (DDLCSCR) area</td>
</tr>
<tr>
<td>idms.system.ddldcscr</td>
<td>file-ID of the system scratch (DDLCSCR) area</td>
</tr>
<tr>
<td>dcmsg</td>
<td>filename of the system message (DDLDCMSG) area</td>
</tr>
<tr>
<td>idms.sysmsg.ddldcmsg</td>
<td>file-ID of the system message (DDLDCMSG) area</td>
</tr>
<tr>
<td>dlocscr</td>
<td>filename of the local mode system scratch (DDLOCSCR) area</td>
</tr>
<tr>
<td>idms.sysloc.ddlocscr</td>
<td>file-ID of the local mode system scratch (DDLOCSCR) area</td>
</tr>
<tr>
<td>Variable</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>dirldb</td>
<td>filename of the IDMSDIRL definition (DDLDML) area</td>
</tr>
<tr>
<td>idms.sysdirl.ddldml</td>
<td>file-ID of the IDMSDIRL definition (DDLDML) area</td>
</tr>
<tr>
<td>dirlld</td>
<td>filename of the IDMSDIRL definition load (DDLDCLOD) area</td>
</tr>
<tr>
<td>idms.sysdirl.dirllod</td>
<td>file-ID of the IDMSDIRL definition load (DDLDCLOD) area</td>
</tr>
<tr>
<td>empdemo</td>
<td>filename of the EMPDEMO area</td>
</tr>
<tr>
<td>idms.empdemo1</td>
<td>file-ID of the EMPDEMO area</td>
</tr>
<tr>
<td>insdemo</td>
<td>filename of the INSDEMO area</td>
</tr>
<tr>
<td>idms.insdemo1</td>
<td>file-ID of the INSDEMO area</td>
</tr>
<tr>
<td>orgdemo</td>
<td>filename of the ORGDEMO area</td>
</tr>
<tr>
<td>idms.orgdemo1</td>
<td>file-ID of the ORGDEMO area</td>
</tr>
<tr>
<td>empldem</td>
<td>filename of the EMPLDEMO area</td>
</tr>
<tr>
<td>idms.sqldemo.empldemo</td>
<td>file-ID of the EMPLDEMO area</td>
</tr>
<tr>
<td>infodem</td>
<td>filename of the INFODEMO area</td>
</tr>
<tr>
<td>idms.sqldemo.infodemo</td>
<td>file-ID of the INFODEMO area</td>
</tr>
<tr>
<td>projdem</td>
<td>filename of the PROJDEMO area</td>
</tr>
<tr>
<td>idms.projseg.projdemo</td>
<td>file-ID of the PROJDEMO area</td>
</tr>
<tr>
<td>indxdem</td>
<td>filename of the INDXDEMO area</td>
</tr>
<tr>
<td>idms.sqldemo.indxdemo</td>
<td>file-ID of the INDXDEMO area</td>
</tr>
<tr>
<td>sysctl</td>
<td>filename of the SYSCTL file</td>
</tr>
<tr>
<td>idms.sysctl</td>
<td>file-ID of the SYSCTL file</td>
</tr>
<tr>
<td>secdcl</td>
<td>filename of the system user catalog (DDLSEC) area</td>
</tr>
<tr>
<td>idms.sysuser.ddlsec</td>
<td>file-ID of the system user catalog (DDLSEC) area</td>
</tr>
<tr>
<td>dictdb</td>
<td>filename of the application dictionary definition area</td>
</tr>
<tr>
<td>idms.appldict.ddldml</td>
<td>file-ID of the application dictionary definition (DDLDML) area</td>
</tr>
<tr>
<td>dloddb</td>
<td>filename of the application dictionary definition load area</td>
</tr>
<tr>
<td>idms.appldict.ddldclod</td>
<td>file-ID of the application dictionary definition load (DDLDCLOD) area</td>
</tr>
<tr>
<td>sqldd</td>
<td>filename of the SQL catalog (DDLCAT) area</td>
</tr>
<tr>
<td>idms.syssql.ddlcat</td>
<td>file-ID of the SQL catalog (DDLCAT) area</td>
</tr>
<tr>
<td>sqllod</td>
<td>filename of the SQL catalog load (DDLCATL) area</td>
</tr>
<tr>
<td>idms.syssql.ddlcatl</td>
<td>file-ID of SQL catalog load (DDLCATL) area</td>
</tr>
<tr>
<td>sqlxdd</td>
<td>filename of the SQL catalog index (DDLCATX) area</td>
</tr>
<tr>
<td>idms.syssql.ddlcatx</td>
<td>file-ID of the SQL catalog index (DDLCATX) area</td>
</tr>
<tr>
<td>ascii</td>
<td>filename of the asf dictionary definition (DDLDML) area</td>
</tr>
<tr>
<td>idms.asfdict.ddldml</td>
<td>file-ID of the asf dictionary definition (DDLDML) area</td>
</tr>
<tr>
<td>asflod</td>
<td>filename of the asf dictionary definition load (ASFLOD) area</td>
</tr>
<tr>
<td>idms.asfdict.asflod</td>
<td>file-ID of the asf dictionary definition load (ASFLOD) area</td>
</tr>
</tbody>
</table>
**asfdata**
filename of the asf data (ASFDATA) area

**idms.asfdict.asfdata**
file-ID of the asf data area (ASFDATA) area

**ASFDEFN**
filename of the asf data definition (ASFDEFN) area

**idms.asfdict.asfdefn**
file-ID of the asf data definition area (ASFDEFN) area

**j1jrnl**
filename of the first disk journal file

**idms.j1jrnl**
file-ID of the first disk journal file

**j2jrnl**
filename of the second disk journal file

**idms.j2jrnl**
file-ID of the second disk journal file

**j3jrnl**
filename of the third disk journal file

**idms.j3jrnl**
file-ID of the third disk journal file

**SYSIDMS**
filename of the SYSIDMS parameter file

**IDMSDMLC**

```plaintext
/ADD-FILE-LINK
  L-NAME=CDMSLIB,F-NAME=idms.dba.loadlib
/ADD-FILE-LINK L-NAME=CDMSLIB1,F-NAME=idms.loadlib
/ADD-FILE-LINK L-NAME=CDMSTOR,F-NAME=idms.loadlib
/ADD-FILE-LINK L-NAME=sysctl,F-NAME=idms.sysctl,SHARED-UPD=*YES
/ADD-FILE-LINK L-NAME=SYSIDMS,F-NAME=*DUMMY
/ASSIGN-SYSOPT TO=*SYSCMD
/START-PROG
  *MOD(ELEM=IDMSDMLC,LIB=idms.dba.loadlib,RUN-MODE=*ADV)
  DICTNAME=dictionary-name DMCL=dmcl-name sysidms-input-parms
  PARM='precompiler-options' END-SYSIDMS
```

**COBOL/DML source statements**

```plaintext
/ASSIGN-SYSOPT TO=*PRIMARY
/ASSIGN-SYSDTA TO=temp.punch
/START-COBOL85-COMPILER
  /MODULE-OUTPUT=LIB-ELEM(LIB=idms.objlib.user,ELEM=userprog)
  /COMPILER-ACTION=MODULE-GENERATION(MODULE-FORMAT=OM)
  /LISTING=(SOURCE=YES,DIAGNOSTICS=YES,OUTPUT=SYSLIST)
/START-BINDER
  /START-LLM-CREATION INTERNAL-NAME=userprog
  /INC-MOD LIB=idms.objlib.user,ELEM=userprog
  /INC-MOD LIB=idms.loadlib,ELEM=IDMSPBS2 For DC, BATCH and DCBATCH
  /INC-MOD LIB=idms.loadlib,ELEM=IDMSTCM UTM only
  /RESOLVE-BY-AUTOLINK LIB=cobol.objlib
  /SAVE-LLM LIB=idms.loadlib.user,ELEM=userprog(VER=@),OVER=YES
/END
/DELETE-FILE temp.punch
```

**idms.**
filename of the load library containing the CA IDMS executable modules

**loadlib**

**idms.**
filename of the load library containing the DMCL and database name table load modules

**dba.**

**loadlib**

**sysctl**
linkname of SYSCTL file

**idms.**
filename of SYSCTL file

**sysctl**
temp.  filename of temporary file that contains DML compiler output
punch

sysidms  parameters that specify physical requirements of the environment, runtime directives, or
-input-
 parms  operating system-dependent file information. For a complete description of all SYSIDMS
parameters and syntax, see CA IDMS Common Administering section.

precom  options that control various aspects of the precompile process. See Passing Parameters to
piler-
 options  the Precompiler (see page 296) for a complete description of the options.

idms.  filename of user object library
objlib.
user

userpro  name of user application program

cobol.  filename of the COBOL runtime object library
objlib

idms.  filename of the user load library
loadlib.
user

Note: Depending on the CV operating environment, an IDMSOPTI module link edited with
the DML compiler can be used in place of or in addition to the SYSCTL file.

Local Mode

To execute the compiler in local mode:

- Remove the SYSCTL ADD-FILE-LINK command
- Add:

  /ADD-FILE-LINK L-NAME=dictdb,F-NAME=idms.appldict.ddldml,SHARED-UPD=*YES
  [/CREATE-FILE F-NAME=idms.tapejrnl,SUPPRESS-ERRORS=*FILE-EXIST,       -
   / SUP=*TAPE(VOLUME=nnnnnn,DEVICE=tape)]
  /ADD-FILE-LINK L-NAME=sysjrnl,F-NAME=idms.tapejrnl [,BUF-LEN=bbbb,       -
   / SUP=*TAPE(F-SEQ=1)]

  Statements and parameters between brackets must be specified only
  when using the journal file on tape.

<table>
<thead>
<tr>
<th>dictdb</th>
<th>linkname of the data dictionary file</th>
</tr>
</thead>
<tbody>
<tr>
<td>idms.appldict.ddldml</td>
<td>filename of the data dictionary file</td>
</tr>
<tr>
<td>sysjrnl</td>
<td>linkname of the tape journal file</td>
</tr>
<tr>
<td>idms.tapejrnl</td>
<td>filename of the tape journal file</td>
</tr>
<tr>
<td>bbbb</td>
<td>page size of the file</td>
</tr>
<tr>
<td>nnnnnn</td>
<td>volume serial number of the tape archive file</td>
</tr>
<tr>
<td>tape</td>
<td>device name for the tape journal file</td>
</tr>
</tbody>
</table>
CMS Commands

**IDMSDMLC ('CMS')**

FILEDEF SYSIPT DISK systipt data a (RECFM F LRECL ppp BLKSIZE nnn) FILEDEF SYSPCH DISK prgne cobol a
FILEDEF SYSIDMS DISK sysidms parms a (RECFM F LRECL ppp. BLKSIZE nnnEXEC IDMSFD
OSRUN IDMSDMLC PARM='CVMACH=vmid,precompiler-options'
FILEDEF TEXT DISK prgne TEXT A
GLOBAL TXTLIB coblibvs IDMSLIBI
COBOL prgname (OSEDECK APOST LIB COBOL compile stepTXTLIB DEL utextlib prgnameTXTLIB
IB ADD utextlib prgnameFILEDEF SYSMOD uloadlib LOADLIB a (RECFM V LRECL 1024 BLKSIZE 1
0 24
FILEDEF objlibl DISK IDMSLIBI TXTLIB A
FILEDEF objlib D UTEXTLIB TXTLIB a
FILEDEF SYSLIB DISK coblibvs TXTLIB p
FILEDEF SYS001 DISK wfn wft wflked linkctl (LIST XREF LET MAP RENT NOTERM PRINT SIZE
512K 64K

Link edit step

<table>
<thead>
<tr>
<th>systipt data a</th>
<th>Filename, type, and mode of the file containing the COBOL/DML source statements</th>
</tr>
</thead>
<tbody>
<tr>
<td>ppp</td>
<td>Record length of the data file</td>
</tr>
<tr>
<td>nnn</td>
<td>Block size of the data file</td>
</tr>
<tr>
<td>prgne cobol a</td>
<td>Filename of the COBOL program</td>
</tr>
<tr>
<td>sysidms parms a</td>
<td>Filename, filetype, and filemode of the file that contains SYSIDMS parameters</td>
</tr>
<tr>
<td>vmid</td>
<td>ID of the virtual machine running the central version</td>
</tr>
<tr>
<td>precompiler-options</td>
<td>options that control various aspects of the precompile process. See Passing Parameters to the Precompiler (see page 296) for a complete description of the options.</td>
</tr>
<tr>
<td>coblibvs</td>
<td>Filename of the library that contains COBOL logic modules</td>
</tr>
<tr>
<td>utextlib</td>
<td>Filename of the user text library</td>
</tr>
<tr>
<td>uloadlib</td>
<td>Filename, filetype, and filemode of the user load library</td>
</tr>
<tr>
<td>objlib1</td>
<td>DDname of the first CA IDMS object library</td>
</tr>
<tr>
<td>objlib</td>
<td>DDname of the user object library</td>
</tr>
<tr>
<td>coblibvs TXTLIB p</td>
<td>Filename, filetype, and filemode of the library that contains COBOL logic modules</td>
</tr>
<tr>
<td>wfn wft wfm</td>
<td>Filename, type, and mode of the files to be used as intermediate work files by IDMSDMLC</td>
</tr>
<tr>
<td>linkctl</td>
<td>Filename of the file that contains the linkage editor control statements</td>
</tr>
</tbody>
</table>

How to Edit the SYSIDMS File

To create the SYSIDMS file, enter these CMS commands:

```
XEDIT sysidms parms a (NOPROF
INPUT
```

15-Jan-2018  294/415
To run IDMSDMLC, you must include the NODENAME and DICTNAME SYSIDMS parameters.

⚠️ **Note:** For more information about SYSIDMS parameters, see the *CA IDMS Common Administrating section*.

### How to Create the SYSIPT File

To create the SYSIPT file, enter these CMS commands:

```cml
XEDIT sysipt data a (NOPROF
INPUT
.
.
DML source statements
.
.
FILE
```

### How to Create the LINKCTL File

To create the LINKCTL file, enter these CMS commands:

```cml
XEDIT linkctl data a (NOPROF
INPUT
.
.
INCLUDE objlib(prgname)INCLUDE objlib1(IDMS)  IDMS is required, omit for CICS
INCLUDE objlib1(IDMSCIINT) for CICS onlyINCLUDE objlib1(IDMSCANC) IDMSCANC for BATCH and
ENTRY NAME (R) prgname ENTRY prgname(R)
.
.
FILE
```

### Executing in Local Mode

To execute IDMSDMLC in local mode, remove the CVMACH parameter from OSRUN, and do one of the following:

- Link IDMSDMLC with an IDMSOPTI program that specifies local execution mode
- Specify *LOCAL* as the first input parameter in the file specified in the FILEDEF SYSIPT statement
- Modify the OSRUN statement, as follows:
  ```cml
  OSRUN IDMSDMLC PARM='*LOCAL*
  ```
Note: This option is valid only if the OSRUN command is issued from a System Product Interpreter or from an EXEC2 file.

Link-Edit Considerations

The modules involved in the link edit of an application program contain three external references. Some must be resolved, others can be left unresolved depending on the mode of operation. The table below lists and explains the external references.

<table>
<thead>
<tr>
<th>Reference</th>
<th>Referenced by</th>
<th>Resolved by</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABORT</td>
<td>Application program</td>
<td>IDMSCANC</td>
<td>Should be resolved ONLY in a batch environment; should NOT be included in a tp environment.</td>
</tr>
<tr>
<td>IDMS</td>
<td>Application program</td>
<td>IDMS</td>
<td>Must be resolved</td>
</tr>
<tr>
<td>IDMSOPTI</td>
<td>IDMSOPTI module</td>
<td>IDMSOPTI</td>
<td>Must be resolved if using the central version without a SYSCTL file</td>
</tr>
</tbody>
</table>

* IDMSOPTI is a weak external reference (WXTRN).

Passing Parameters to the Precompiler

A number of parameters can be provided to control the action taken by the precompiler. The parameters can be specified in one of three ways:

- An IDMSPPRM module can be compiled with parameter values that are always appropriate to a particular operating system or client site. IDMSPPRM must be a stand-alone assembler module that will be loaded by the precompiler at run-time. The module must consist of a string of characters terminated by a binary zero.

- A PARM= clause can be coded on the EXEC statement that invokes IDMSDMLC in a z/OS, or z/VSE environment or on the OSRUN statement that invokes IDMSDMLC in a CMS environment. Any option that is specified on the EXEC or OSRUN statement will take precedence over the same parameter if it is coded with a different value in the IDMSPPRM module.

- A PARM= statement can be coded as a SYSIDMS input parameter. See CA IDMS Common Administering section for more information about using SYSIDMS. Any option that is specified in the PARM= statement will take precedence over the same parameter if it is coded with a different value in the IDMSPPRM module. Note that if PARM= is specified both as a SYSIDMS input statement and on an EXEC or OSRUN statement, the PARM= clause on the EXEC or OSRUN statement will be ignored completely.

Precompiler Options

Parameter options available to code in the EXEC statement of the precompile step are:
• **RCM=rcm-name**
  Specifies the name of the RCM created for the program by the precompiler if the program uses SQL access.

• **RCMVERSION=rcm-version-number**
  Specifies the version number of the RCM created for the program by the precompiler.

• **AM=access-module-name**
  Specifies the name of the access module to be executed for the program at runtime if the program uses SQL access.

• **SCHEMA=schema-name**
  Specifies the default schema-name qualifier for the precompiler to use when processing an INCLUDE TABLE statement that does not supply a qualifier.

• **NOINSTALL**
  Specifies that the precompiler should only check syntax.

• **DICTNAME=dictionary-name**
  Specifies the name of the dictionary the precompiler should access.

• **SQL=NO/89/FIPS/DISABLED**
  Specifies the SQL syntax standard that the precompiler should apply when checking the validity of SQL statements in the program.
  Option NO, the default, means that compliance with a named SQL standard is not checked or enforced, and all CA IDMS extensions are permitted.
  Option 89 directs the precompiler to use ANSI X3.135-1989 (Rev), Database Language SQL with integrity enhancement as the standard for compliance.
  Option FIPS directs the precompiler to use FIPS PUB 127-1, *Database Language SQL* as the standard for compliance.
  Option DISABLED directs the precompiler not to process any SQL commands (denoted by EXEC SQL, END-EXEC delimiters) in the program.

• **LIST/NOList**
  LIST directs the precompiler to create a listing of the program with precompiler messages. NOList directs the compiler not to create a listing of the program with precompiler messages.

• **DATE=ISO/USA/EUR/JIS**
  Specifies the format of the DATE data type to be used for communication between the program and the database when the access module is executed.

• **TIME=ISO/USA/EUR/JIS**
  Specifies the format of the TIME data type to be used for communication between the program and the database when the access module is executed.

• **COBOL=1/2/85**
  Specifies the version of COBOL with which COBOL statements generated by the precompiler must comply.
  Option 1 directs the precompiler to generate statements that comply with any of the following: Versions of VS COBOL that precede VS COBOL II for z/OS, or z/VSE operating systems all CBOL compiler versions for BS2000 operating systems.
Option 2, the default, directs the precompiler to generate statements that comply with VS COBOL II or LE-compliant COBOL compilers. Option 85 directs the precompiler to comply with COBOL85, the version of COBOL required for the Fujitsu and Hitachi compilers.

- **SR1SR7 = YES/NO**
  If YES is specified then SR1 and SR7 will be emitted in SUBSCHEMA-RECNAMES. NO is the default.

⚠️ **Note:** For more information about SQL-related parameter options, see the SQL Administrating section.

### Site-specific Parameters

The following sample IDMSPPRM source will change the default for the COBOL parameter from COBOL=2 to COBOL=1 and will direct the precompiler not to produce a listing of the source program.

```cobol
EDBPPARM CSECT
DC C'COBOL=1,NOLIST'
DC X'00'
END
```
Sample Batch Program

Contents

- Sample Batch Program as Input to the DML Compiler (see page 299)
- Sample Batch Program as Output from the DML Compiler (see page 305)
- Sample Batch Program from the COBOL Precompiler (see page 317)

This section contains a sample batch COBOL program that accesses database records using navigational DML statements. The following figure shows the program as it appears in the various stages of the compilation process. You create a program using COBOL and DML statements. This program is input to the DML compiler, which produces a listing that contains diagnostics and, optionally, DML source statements. The expanded code is input to the COBOL compiler, which generates a listing of the fully expanded code and diagnostics.

Compilation Process

About This Appendix

Sample Batch Program as Input to the DML Compiler

The sample program contains COBOL and DML source statements.

*RETRIEVAL
*DMLIST
*NO-ACTIVITY-LOG
*SCHEMA-COMMENTS

IDENTIFICATION DIVISION.

PROGRAM-ID. DEPTRPT.

AUTHOR. COMPUTER ASSOCIATES INTERNATIONAL.

DATE-WRITTEN. APRIL 1995.
REMARKS. THIS PROGRAM DEMONSTRATES CA IDMS DATABASE ACCESS USING COBOL DML STATEMENTS. IT READS DEPARTMENT ID NUMBERS AND RETRIEVES RELATED RECORD OCCURRENCES, PRINTING A REPORT THAT INCLUDES DEPARTMENT, EMPLOYEE, JOB, AND OFFICE INFORMATION.

**********************************************************************
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
  SELECT DEPT-FILE-IN ASSIGN TO INFILE.
  SELECT DEPT-FILE-OUT ASSIGN TO OUTFILE.
  SELECT ERR-FILE-OUT ASSIGN TO ERRFILE.
**********************************************************************
IDMS-CONTROL SECTION.
PROTOCOL. MODE IS BATCH DEBUG IDMS-RECORDS MANUAL.
  SKIP3
**********************************************************************
DATA DIVISION.
SCHEMA SECTION.
DB EMPSS01 WITHIN EMPSCHM.
**********************************************************************
FILE SECTION.
FD DEPT-FILE-IN
  RECORD CONTAINS 80
  BLOCK CONTAINS 80 CHARACTERS
  RECORDING MODE IS F
  LABEL RECORDS ARE OMITTED.
01 DEPT-REC-IN.
  02 DEPT-ID-IN PIC 9(4).
  02 DEPT-IN-FILLER PIC X(76).
FD DEPT-FILE-OUT
  RECORD CONTAINS 133
  BLOCK CONTAINS 133 CHARACTERS
  RECORDING MODE IS F
  LABEL RECORDS ARE OMITTED.
01 DEPT-REC-OUT.
  02 CC PIC X.
  02 PRINT-LINE PIC X(132).
FD ERR-FILE-OUT
  RECORD CONTAINS 133
  BLOCK CONTAINS 133 CHARACTERS
  RECORDING MODE IS F
  LABEL RECORDS ARE OMITTED.
01 ERR-REC-OUT.
  02 ERR-CC PIC X.
  02 ERR-LINE PIC X(132).
**********************************************************************
WORKING-STORAGE SECTION.
  01 EOF-SW PIC X VALUE 'N'.
    88 END-OF-FILE VALUE 'Y'.
  01 LINE-COUNT PIC 99 VALUE 0.
  01 ERR-LINE-COUNT PIC 99 VALUE 0.
  01 LINE-MAX PIC 99 VALUE 50.
**********************************************************************
  01 DEPT-HEADER.
    05 FILLER PIC X(30) VALUE SPACES.
05 FILLER PIC X(13) VALUE 'DEPARTMENT ID'.
05 FILLER PIC X(10) VALUE SPACES.
05 FILLER PIC X(9) VALUE 'DEPT NAME'.
05 FILLER PIC X(70) VALUE SPACES.
01 DEPT-DETAIL-LINE.
  05 FILLER PIC X(33) VALUE SPACES.
  05 DEPT-ID-OUT PIC X(4).
  05 FILLER PIC X(16) VALUE SPACES.
  05 DEPT-NAME-OUT PIC X(45).
  05 FILLER PIC X(34) VALUE SPACES.
01 EMP-HEADER.
  05 FILLER PIC X(5) VALUE SPACES.
  05 FILLER PIC X(6) VALUE 'EMP ID'.
  05 FILLER PIC X(2) VALUE SPACES.
  05 FILLER PIC X(9) VALUE 'LAST NAME'.
  05 FILLER PIC X(8) VALUE SPACES.
  05 FILLER PIC X(10) VALUE 'FIRST NAME'.
  05 FILLER PIC X(3) VALUE SPACES.
  05 FILLER PIC X(10) VALUE 'START DATE'.
  05 FILLER PIC X(2) VALUE SPACES.
  05 FILLER PIC X(9) VALUE 'JOB TITLE'.
  05 FILLER PIC X(13) VALUE SPACES.
  05 FILLER PIC X(14) VALUE 'OFFICE ADDRESS'.
01 EMP-DETAIL-LINE.
  05 FILLER PIC X(5) VALUE SPACES.
  05 ID-OUT PIC X(4).
  05 LAST-OUT PIC X(15).
  05 FILLER PIC X(2) VALUE SPACES.
  05 FIRST-OUT PIC X(10).
  05 FILLER PIC X(3) VALUE SPACES.
  05 SD-OUT.
    10 SD-MM PIC XX.
    10 FILLER PIC X VALUE '/'.
    10 SD-DD PIC XX.
    10 FILLER PIC X VALUE '/'.
    10 SD-YY PIC XX.
  05 FILLER PIC X(4) VALUE SPACES.
  05 TITLE-OUT PIC X(20).
  05 FILLER PIC X(2) VALUE SPACES.
  05 OFF-ADDRESS-OUT.
    10 STREET-OUT PIC X(20).
    10 FILLER PIC XX VALUE SPACES.
    10 CITY-OUT PIC X(15).
    10 FILLER PIC XX VALUE SPACES.
    10 STATE-OUT PIC XX.
    10 FILLER PIC XX VALUE SPACES.
    10 ZIP-OUT PIC X(5).
  05 FILLER PIC X(8) VALUE SPACES.
01 ERR-HEADER-1.
  05 FILLER PIC X(40) VALUE SPACES.
  05 FILLER PIC X(12) VALUE 'ERROR REPORT'.
  05 FILLER PIC X(80) VALUE SPACES.
01 ERR-HEADER-2.
  05 FILLER PIC X(10) VALUE SPACES.
  05 FILLER PIC X(4) VALUE '***'.
  05 FILLER PIC X(51) VALUE 'THIS REPORT LISTS EMPTY AND NONEXISTENT DEPARTMENTS'.
  05 FILLER PIC X(4) VALUE '***'.
  05 FILLER PIC X(63) VALUE SPACES.
01 ERR-HEADER-3.
  05 FILLER PIC X(20) VALUE SPACES.
  05 FILLER PIC X(7) VALUE 'DEPT ID'.
  05 FILLER PIC X(9) VALUE SPACES.
  05 FILLER PIC X(7) VALUE 'MESSAGE'.
  05 FILLER PIC X(89) VALUE SPACES.
01 ERR-DETAIL-LINE.
  05 FILLER PIC X(20) VALUE SPACES.
  05 ERR-ID-OUT PIC X(4).
  05 FILLER PIC X(12) VALUE SPACES.
05 ERR-MESS-OUT PIC X(15).
05 FILLER PIC X(79) VALUE SPACES.

***********************************************************************

01 MESSAGES.
  05 NO-JOB-MESSAGE.
    10 FILLER PIC X(20) VALUE 'NO JOB ASSIGNED'.
  05 NO-OFFICE-MESSAGE.
    10 FILLER PIC X(20) VALUE 'NO OFFICE ASSIGNED'.
  05 NO-DEPT-MESSAGE.
    10 FILLER PIC X(15) VALUE 'DOES NOT EXIST'.
  05 NO-EMP-MESSAGE.
    10 FILLER PIC X(15) VALUE 'IS EMPTY'.
  05 NO-INPUT-MESSAGE.
    10 FILLER PIC XX VALUE SPACES.
    10 FILLER PIC X(11) VALUE '========>> '.
    10 FILLER PIC X(8) VALUE 'NO INPUT'.
    10 FILLER PIC X(11) VALUE ' <<========='.
    10 FILLER PIC X(100) VALUE SPACES.

01 COPY IDMS SUBSCHEMA-CTRL.
01 COPY IDMS SUBSCHEMA-SSNAME.
01 COPY IDMS SUBSCHEMA-RECNAMES.
01 COPY IDMS SUBSCHEMA-SETNAMES.
01 COPY IDMS RECORD EMPLOYEE.
01 COPY IDMS RECORD DEPARTMENT.
01 COPY IDMS RECORD JOB.
01 COPY IDMS RECORD EMPOSITION.
01 COPY IDMS RECORD OFFICE.
EJECT

PROCEDURE DIVISION.

* ************************************************************
* * PROCEDURE DIVISION GENERAL STRATEGY: *                 *
* *  1) READ DEPT-ID-IN, WHICH CONTAINS THE *                *
* *     DEPT-ID NUMBER *                                   *
* *  2) ACCESS THE DATABASE USING THE DEPT-ID NUMBER *      *
* *     WITH AN OBTAIN CALC ON THE DEPARTMENT RECORD *     *
* *  3) ACCESS ALL EMPLOYEES IN THE DEPT-EMPLOYEE SET *    *
* *     AND RETRIEVE RELATED JOB AND OFFICE DATA *         *
* *  4) PRINT A REPORT FOR EACH DEPARTMENT *               *
* *  5) PRINT AN ERROR REPORT FOR EMPTY DEPARTMENTS *      *
* *     AND NONEXISTENT DEPARTMENTS (NO MATCHING *        *
* *     DEPT-ID) *                                       *
* ************************************************************

MAIN-LINE.
PERFORM INIT-FILES.
  IF END-OF-FILE
    PERFORM EMPTY-INPUT-PROCESSING
  ELSE
    PERFORM INIT-BIND-READY
    PERFORM U220-ERR-HEADER
    PERFORM DEPT-PROCESSING THRU DEPT-PROCESSING-EXIT
    PERFORM END-PROCESSING.
    GOBACK.

INIT-BIND-READY.

* THE BIND STATEMENTS ARE PERFORMED INDIVIDUALLY (RATHER *
* THAN BY USING A COPY IDMS SUBSCHEMA-BINDS) IN ORDER TO *
* CHECK EACH ERROR-STATUS BY PERFORMING THE IDMS-STATUS *
CA IDMS Reference - 19.0

* ROUTINE.

**************************************************************
MOVE 'DEPTRPT' TO PROGRAM-NAME.
BIND RUN-UNIT.
PERFORM IDMS-STATUS.
BIND EMPLOYEE.
PERFORM IDMS-STATUS.
BIND DEPARTMENT.
PERFORM IDMS-STATUS.
BIND JOB.
PERFORM IDMS-STATUS.
BIND EMPOSITION.
PERFORM IDMS-STATUS.
BIND OFFICE.
PERFORM IDMS-STATUS.
READY.
PERFORM IDMS-STATUS.

INIT-FILES.
OPEN INPUT DEPT-FILE-IN.
OPEN OUTPUT DEPT-FILE-OUT.
OPEN OUTPUT ERR-FILE-OUT.
MOVE SPACES TO PRINT-LINE.
MOVE SPACES TO ERR-LINE.
READ DEPT-FILE-IN AT END MOVE 'Y' TO EOF-SW.

EMPTY-INPUT-PROCESSING.
MOVE NO-INPUT-MESSAGE TO PRINT-LINE.
MOVE '1' TO CC.
PERFORM U000-WRITE-LINE.

**************************************************************
* THIS PARAGRAPH ACCESSES THE DATABASE USING THE DEPT-ID-0415 *
* CALCKEY VALUE.                       *
**************************************************************

DEPT-PROCESSING.
MOVE DEPT-ID-IN TO DEPT-ID-0410.
OBTAIN CALC DEPARTMENT.
IF DB-REC-NOT-FOUND THEN
  PERFORM NO-DEPT-PROCESSING
ELSE
  PERFORM IDMS-STATUS.
  IF DEPT-EMPLOYEE IS NOT EMPTY THEN
    PERFORM U020-VALID-HEADER
    MOVE DEPT-ID-0410 TO DEPT-ID-OUT
    MOVE DEPT-NAME-0410 TO DEPT-NAME-OUT
    MOVE DEPT-DETAIL-LINE TO PRINT-LINE
    PERFORM U000-WRITE-LINE
    PERFORM U030-EMP-HEADERS
    PERFORM SET-WALK THRU SET-WALK-EXIT
    UNTIL DB-END-OF-SET
  ELSE
    PERFORM EMPTY-SET.
  READ DEPT-FILE-IN AT END MOVE 'Y' TO EOF-SW.
  DEPT-PROCESSING-EXIT.

EXIT.

**************************************************************
* THIS PARAGRAPH RETRIEVES EMPLOYEE, JOB, AND OFFICE DATA   *
* FOR EACH EMPLOYEE IN THE DEPT-EMPLOYEE SET.         *
**************************************************************

SET-WALK.
OBTAIN NEXT EMPLOYEE WITHIN DEPT-EMPLOYEE.
IF DB-END-OF-SET
  GO TO SET-WALK-EXIT
ELSE
  PERFORM IDMS-STATUS.
  MOVE EMP-ID-0415 TO ID-OUT.
  MOVE EMP-LAST-NAME-0415 TO LAST-OUT.
  MOVE EMP-FIRST-NAME-0415 TO FIRST-OUT.
  MOVE START-YEAR-0415 TO SD-YY.
MOVE START-MONTH-0415 TO SD-MM.
MOVE START-DAY-0415 TO SD-DD.
IF EMP-EMPOSITION IS EMPTY
  MOVE NO-JOB-MESSAGE TO TITLE-OUT
ELSE
  FIND FIRST WITHIN EMP-EMPOSITION
  PERFORM IDMS-STATUS
  IF NOT JOB-EMPOSITION MEMBER
    MOVE NO-JOB-MESSAGE TO TITLE-OUT
  ELSE
    OBTAIN OWNER WITHIN JOB-EMPOSITION
    PERFORM IDMS-STATUS
    MOVE TITLE-0440 TO TITLE-OUT.
  IF OFFICE-EMPLOYEE IS EMPTY
  MOVE NO-OFFICE-MESSAGE TO STREET-OUT
  MOVE SPACES TO CITY-OUT
  MOVE SPACES TO STATE-OUT
  MOVE SPACES TO ZIP-OUT
ELSE
  OBTAIN OWNER WITHIN OFFICE-EMPLOYEE
  PERFORM IDMS-STATUS
  MOVE OFFICE-STREET-0450 TO STREET-OUT
  MOVE OFFICE-CITY-0450 TO CITY-OUT
  MOVE OFFICE-STATE-0450 TO STATE-OUT
  MOVE OFFICE-ZIP-FIRST-FIVE-0450 TO ZIP-OUT
  MOVE EMP-DETAIL-LINE TO PRINT-LINE.
  PERFORM U000-WRITE-LINE.
SET-WALK-EXIT.
EXIT.
END-PROCESSING.
FINISH.
PERFORM IDMS-STATUS.
CLOSE DEPT-FILE-OUT.
CLOSE ERR-FILE-OUT.
CLOSE DEPT-FILE-IN.
EMPTY-SET.
MOVE SPACES TO ERR-LINE.
MOVE DEPT-ID-0410 TO ERR-ID-OUT.
MOVE NO-EMP-MESSAGE TO ERR-MESS-OUT.
MOVE ERR-DETAIL-LINE TO ERR-LINE.
PERFORM U200-WRITE-ERR-LINE.
NO-DEPT-PROCESSING.
MOVE DEPT-ID-IN TO ERR-ID-OUT.
MOVE NO-DEPT-MESSAGE TO ERR-MESS-OUT.
MOVE ERR-DETAIL-LINE TO ERR-LINE.
PERFORM U200-WRITE-ERR-LINE.
U000-WRITE-LINE.
WRITE DEPT-REC-OUT AFTER POSITIONING CC.
  IF CC = '1' THEN MOVE 0 TO LINE-COUNT
  ELSE IF CC = ' ' THEN ADD 1 TO LINE-COUNT
  ELSE IF CC = '0' THEN ADD 2 TO LINE-COUNT.
  IF LINE-COUNT > LINE-MAX
    THEN PERFORM U010-NEW-PAGE-Routine.
U010-NEW-PAGE-Routine.
PERFORM U020-VALID-HEADER.
MOVE DEPT-DETAIL-LINE TO PRINT-LINE.
PERFORM U000-WRITE-LINE.
PERFORM U030-EMP-HEADERS.
U020-VALID-HEADER.
MOVE DEPT-HEADER TO PRINT-LINE.
MOVE '1' TO CC.
PERFORM U000-WRITE-LINE
MOVE ' ' TO CC.
U030-EMP-HEADERS.
MOVE '0' TO CC.
MOVE EMP-HEADER TO PRINT-LINE.
PERFORM U000-WRITE-LINE.
Sample Batch Program as Output from the DML Compiler

Since the *DMLIST option is specified in the program's IDENTIFICATION DIVISION, printed output consists of expanded code as well as diagnostics. This output is in the following format:

- **Heading** -- The top of each page of the listing contains the name of the DML compiler being used (IDMSDMLC), the release number of the processor (Release 10.0), the name of the listing (Listing of Messages), the date, the time, and the page number.

- **Input listing and DML compiler-generated code** -- The body of the printout contains the program input listing along with the DML compiler-generated code, formatted as follows:

<table>
<thead>
<tr>
<th>Column</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Sequence numbers generated by the DML compiler</td>
</tr>
<tr>
<td>12</td>
<td>Line numbers generated by the DML compiler</td>
</tr>
<tr>
<td>19</td>
<td>Line numbers generated by the user program</td>
</tr>
<tr>
<td>26</td>
<td>Text of the COBOL source code including text generated by the DML compiler</td>
</tr>
</tbody>
</table>

- **Warnings and Status Messages** -- Diagnostics are imbedded in the input listing and DML compiler-generated code following the errant lines of source code.

⚠️ **Note**: For more information about the DML compiler status messages, see the *CA IDMS Messages and Codes Guide*. 

```
MOVE SPACES TO PRINT-LINE.
MOVE ' ' TO CC.
PERFORM U000-WRITE-LINE.

U200-WRITE-ERR-LINE.
WRITE ERR-REC-OUT AFTER POSITIONING ERR-CC.
IF ERR-CC = '1' THEN MOVE 0 TO ERR-LINE-COUNT
ELSE IF ERR-CC = ' ' THEN ADD 1 TO ERR-LINE-COUNT
ELSE IF ERR-CC = '0' THEN ADD 2 TO ERR-LINE-COUNT.
IF ERR-LINE-COUNT > LINE-MAX THEN
PERFORM U220-ERR-HEADER.
U220-ERR-HEADER.
MOVE ERR-HEADER-1 TO ERR-LINE.
MOVE '1' TO ERR-CC.
PERFORM U200-WRITE-ERR-LINE
MOVE '0' TO ERR-CC.
PERFORM U220-WRITE-ERR-LINE.
MOVE ERR-HEADER-2 TO ERR-LINE.
PERFORM U200-WRITE-ERR-LINE.
MOVE ERR-HEADER-3 TO ERR-LINE.
PERFORM U200-WRITE-ERR-LINE.
MOVE SPACES TO ERR-LINE.
MOVE ' ' TO ERR-CC.
PERFORM U200-WRITE-ERR-LINE.
IDMS-ABORT.
EXIT
IDMS-ABORT-EXIT.
COPY IDMS IDMS-STATUS.
```
This listing contains the sample batch program and partially expanded code generated by the DML compiler.

00001   *RETRIEVAL
00002   *DMLIST
00003   *NO-ACTIVITY-LOG
00004   *SCHEMA-COMMENTS
00005
00006   IDENTIFICATION DIVISION.
00007
00008       PROGRAM-ID. DEPTRPT.
00009
00010       AUTHOR. COMPUTER ASSOCIATES INTERNATIONAL.
00011
00012       DATE-WRITTEN. APRIL 1995.
00013
00014       REMARKS. THIS PROGRAM DEMONSTRATES
00015           CA IDMS DATABASE ACCESS USING
00016           COBOL DML STATEMENTS. IT READS
00017           DEPARTMENT ID NUMBERS AND RETRIEVES
00018           RELATED RECORD OCCURRENCES,
00019           PRINTING A REPORT THAT INCLUDES
00020           DEPARTMENT, EMPLOYEE, JOB, AND
00021           OFFICE INFORMATION.
00022
00023   ***************************************************************
00024   ENVIRONMENT DIVISION.
00025
00026   INPUT-OUTPUT SECTION.
00027
00028       SELECT DEPT-FILE-IN ASSIGN TO INFILE.
00029       SELECT DEPT-FILE-OUT ASSIGN TO OUTFILE.
00030       SELECT ERR-FILE-OUT ASSIGN TO ERRFILE.
00031
00032   ***************************************************************
00033   IDMS-CONTROL SECTION.
00034
00035       PROTOCOL. MODE IS BATCH DEBUG
00036       IDMS-RECORDS MANUAL.
00037       SKIP3
00038
00039   ***************************************************************
00040   DATA DIVISION.
00041
00042   FILE SECTION.
00043
00044       FD DEPT-FILE-IN
00045       RECORD CONTAINS 80
00046       BLOCK CONTAINS 80 CHARACTERS
00047       RECORDING MODE IS F
00048       LABEL RECORDS ARE OMITTED.
00049
00050       01 DEPT-REC-IN.
00051
00052           02 DEPT-ID-IN PIC 9(4).
00053           02 DEPT-IN-FILLER PIC X(76).
00054
00055       FD DEPT-FILE-OUT
00056       RECORD CONTAINS 133
00057       BLOCK CONTAINS 133 CHARACTERS
00058       RECORDING MODE IS F
00059       LABEL RECORDS ARE OMITTED.
00060
00061       01 DEPT-REC-OUT.
00062
00063           02 CC PIC X.
00064           02 PRINT-LINE PIC X(132).
00065
00066       FD ERR-FILE-OUT
00067       RECORD CONTAINS 133
00068       BLOCK CONTAINS 133 CHARACTERS
00069       RECORDING MODE IS F
00069        LABEL RECORDS ARE OMITTED.
00070
00071        01 ERR-REC-OUT.
00072            02 ERR-CC        PIC X.
00073            02 ERR-LINE       PIC X(132).
00074
00075        ***************************************************************
00076        WORKING-STORAGE SECTION.
00077            01 EOF-SW         PIC X     VALUE 'N'.
00078            88 END-OF-FILE   VALUE 'Y'.
00079            01 LINE-COUNT     PIC 99   VALUE 0.
00080            01 ERR-LINE-COUNT PIC 99   VALUE 0.
00081            01 LINE-MAX       PIC 99   VALUE 50.
00082        ***************************************************************
00083        01 DEPT-HEADER.
00084            05 FILLER         PIC X(30) VALUE SPACES.
00085            05 FILLER         PIC X(13) VALUE 'DEPARTMENT ID'.
00086            05 FILLER         PIC X(10) VALUE SPACES.
00087            05 FILLER         PIC X(9)  VALUE 'DEPT NAME'.
00088            05 FILLER         PIC X(70) VALUE SPACES.
00089        01 DEPT-DETAIL-LINE.
00090            05 FILLER         PIC X(33) VALUE SPACES.
00091            05 DEPT-ID-OUT PIC X(4).
00092            05 FILLER         PIC X(16) VALUE SPACES.
00093            05 DEPT-NAME-OUT PIC X(45).
00094            05 FILLER         PIC X(34) VALUE SPACES.
00095        01 EMP-HEADER.
00096            05 FILLER         PIC X(5)  VALUE SPACES.
00097            05 FILLER         PIC X(6)  VALUE 'EMP ID'.
00098            05 FILLER         PIC X(2)  VALUE SPACES.
00099            05 FILLER         PIC X(9)  VALUE 'LAST NAME'.
00100            05 FILLER         PIC X(8)  VALUE SPACES.
00101            05 FILLER         PIC X(10) VALUE 'FIRST NAME'.
00102            05 FILLER         PIC X(3)  VALUE SPACES.
00103            05 FILLER         PIC X(10) VALUE 'START DATE'.
00104            05 FILLER         PIC X(2)  VALUE SPACES.
00105            05 FILLER         PIC X(9)  VALUE 'JOB TITLE'.
00106            05 FILLER         PIC X(13) VALUE SPACES.
00107            05 FILLER         PIC X(14) VALUE 'OFFICE ADDRESS'.
00108            05 FILLER         PIC X(42) VALUE SPACES.
00109        01 EMP-DETAIL-LINE.
00110            05 FILLER         PIC X(5)  VALUE SPACES.
00111            05 ID-OUT        PIC X(4).
00112            05 FILLER         PIC X(4)  VALUE SPACES.
00113            05 LAST-OUT      PIC X(15).
00114            05 FILLER         PIC X(2)  VALUE SPACES.
00115            05 FIRST-OUT     PIC X(10).
00116            05 FILLER         PIC X(3)  VALUE SPACES.
00117            05 SD-OUT.
00118                10 SD-MM      PIC XX.
00119                10 FILLER       PIC X     VALUE '/'.
00120                10 SD-DD      PIC XX.
00121                10 FILLER       PIC X     VALUE '/'.
00122                10 SD-YY      PIC XX.
00123            05 FILLER         PIC X(4)  VALUE SPACES.
00124            05 TITLE-OUT    PIC X(20).
00125            05 FILLER         PIC X(2)  VALUE SPACES.
00126            05 OFF-ADDRESS-OUT.
00127                10 STREET-OUT PIC X(20).
00128                10 FILLER       PIC XX VALUE SPACES.
00129                10 CITY-OUT    PIC X(15).
00130                10 FILLER       PIC XX VALUE SPACES.
00131                10 STATE-OUT  PIC XX.
00132                10 FILLER       PIC XX VALUE SPACES.
00133                10 ZIP-OUT     PIC X(5).
00134            05 FILLER         PIC X(8)  VALUE SPACES.
00135        01 ERR-HEADER-1.
00136            05 FILLER         PIC X(40) VALUE SPACES.
00137            05 FILLER         PIC X(12) VALUE 'ERROR REPORT'.
00138            05 FILLER         PIC X(80) VALUE SPACES.
00139        01 ERR-HEADER-2.
00140 05 FILLER PIC X(10) VALUE SPACES.
00141 05 FILLER PIC X(4) VALUE '****'.
00142 05 FILLER PIC X(51) VALUE
00143 'THIS REPORT LISTS EMPTY AND NONEXISTENT DEPARTMENTS'.
00144 05 FILLER PIC X(4) VALUE '***'.
00145 05 FILLER PIC X(63) VALUE SPACES.
00146 01 ERR-HEADER-3.
00147 05 FILLER PIC X(20) VALUE SPACES.
00148 05 FILLER PIC X(7) VALUE 'DEPT ID'.
00149 05 FILLER PIC X(9) VALUE SPACES.
00150 05 FILLER PIC X(7) VALUE 'MESSAGE'.
00151 05 FILLER PIC X(89) VALUE SPACES.
00152 01 ERR-DETAIL-LINE.
00153 05 FILLER PIC X(20) VALUE SPACES.
00154 05 ERR-ID-OUT PIC X(4).
00155 05 FILLER PIC X(12) VALUE SPACES.
00156 05 ERR-MESS-OUT PIC X(15).
00157 05 FILLER PIC X(79) VALUE SPACES.
00158 ***************************************************************
00159 01 MESSAGES.
00160 05 NO-JOB-MESSAGE.
00161 10 FILLER PIC X(20) VALUE 'NO JOB ASSIGNED'.
00162 05 NO-OFFICE-MESSAGE.
00163 10 FILLER PIC X(20)
00164  VALUE 'NO OFFICE ASSIGNED'.
00165 05 NO-DEPT-MESSAGE.
00166 10 FILLER PIC X(15) VALUE 'DOES NOT EXIST'.
00167 05 NO-EMP-MESSAGE.
00168 10 FILLER PIC X(15) VALUE 'IS EMPTY'.
00169 05 NO-INPUT-MESSAGE.
00170 10 FILLER PIC XX VALUE SPACES.
00171 10 FILLER PIC X(11) VALUE '========>> '.
00172 10 FILLER PIC X(8) VALUE 'NO INPUT'.
00173 10 FILLER PIC X(11) VALUE '<== ========'.
00174 10 FILLER PIC X(100) VALUE SPACES.
00175 DMLC
00176 01 COPY IDMS SUBSCHEMA-CTRL.
00177 01 SUBSCHEMA-CTRL.
00178 03 PROGRAM-NAME PIC X(8)
00179  VALUE SPACES.
00180 03 ERROR-STATUS PIC X(4)
00181  VALUE '1400'.
00182 88 DB-STATUS-OK
00183  VALUE '0000'.
00184 88 ANY-STATUS
00185  VALUE ' ' THRU '9999'.
00186 88 ANY-ERROR-STATUS
00187  VALUE '0001' THRU '9999'.
00188 88 DB-END-OF-SET
00189  VALUE '0307'.
00190 88 DB-REC-NOT-FOUND
00191  VALUE '0326'.
00192 03 DBKEY PIC S9(8) COMP SYNC.
00193 03 RECORD-NAME PIC X(16)
00194  VALUE SPACES.
00195 03 RRECORD-NAME REDEFINES RECORD-NAME.
00196 05 SSC-NODN PIC X(8).
00197 05 SSC-DBN PIC X(8).
00198 03 AREA-NAME PIC X(16)
00199  VALUE SPACES.
00200 03 AREA-RNAME REDEFINES AREA-NAME.
00201 05 SSC-DNO PIC X(8).
00202 05 SSC-DNA PIC X(8).
00203 03 ERROR-SET PIC X(16)
00204  VALUE SPACES.
00205 03 ERROR-RECORD PIC X(16)
00206  VALUE SPACES.
00207 03 ERROR-AREA PIC X(16)
00208  VALUE SPACES.
00209 03 IDBMSCOM-AREA PIC X(100)
00210  VALUE LOW-VALUE
00211 03 IDBMSCOM     REDEFINES IDBMSCOM-AREA
00212     PIC X
00213     OCCURS 100.
00214 03 RDBMSCOM    REDEFINES IDBMSCOM-AREA.
00215      05 DB-SUB-ADDR   PIC X(4).
00216      05 FILLER       PIC X(96).
00217 03 RDBMSCOM    REDEFINES IDBMSCOM-AREA.
00218      05 PAGE-INFO.
00219   07 PAGE-INFO-GROUP PIC S9(4) COMP.
00220   07 PAGE-INFO-DBK-FORMAT
00221   PIC 9(4) COMP.
00222      05 SSC-IDSMS-STATUS-WRK.
00223   07 SSC-IND01-REQ-WK.
00224   09 SSC-IND01-REQ-CODE
00225   PIC S9(8) COMP.
00226   09 SSC-IND01-REQ-RETURN
00227   PIC S9(8) COMP.
00228   07 SSC-STATUS-LINE.
00229   09 SSC-STATUS-LABEL PIC X(16).
00230   09 SSC-STATUS-VALUE PIC X(12).
00231      05 FILLER       PIC X(60).
00232 03 DIRECT-DBKEY      PIC S9(8) COMP SYNC.
00233 03 DIRECT-DBK       REDEFINES DIRECT-DBKEY
00234       PIC S9(8) COMP SYNC.
00235 03 DATABASE-STATUS.
00236      05 DBSTATEMENT-CODE  PIC X(2).
00237      05 DBSTATUS-CODE   PIC X(5).
00238      03 FILLER       PIC X.
00239 03 RECORD-OCUR       PIC S9(8) COMP SYNC.
00240      03 DML-SEQUENCE   PIC S9(8) COMP SYNC.
00241
DMLC 00242 01 COPY IDMS SUBSCHEMA-SSNAME.
00243 01 SUBSCHEMA-SSNAME   PIC X(8)
00244       VALUE 'EMPSS01'.
00245
DMLC 00246 01 COPY IDMS SUBSCHEMA-RECNAMES.
00247 01 SUBSCHEMA-RECNAMES.
00248      03 SR460      PIC X(16)
00249       VALUE 'STRUCTURE' .
00250      03 SR455      PIC X(16)
00251       VALUE 'SKILL' .
00252      03 SR450      PIC X(16)
00253       VALUE 'OFFICE' .
00254      03 SR445      PIC X(16)
00255       VALUE 'NON-HOSP-CLAIM' .
00256      03 SR440      PIC X(16)
00257       VALUE 'JOB' .
00258      03 SR435      PIC X(16)
00259       VALUE 'INSURANCE-PLAN' .
00260      03 SR430      PIC X(16)
00261       VALUE 'HOSPITAL-CLAIM' .
00262      03 SR425      PIC X(16)
00263       VALUE 'EXPERTISE' .
00264      03 SR420      PIC X(16)
00265       VALUE 'EMPOSITION' .
00266      03 SR415      PIC X(16)
00267       VALUE 'EMPLOYEE' .
00268      03 SR410      PIC X(16)
00269       VALUE 'DEPARTMENT' .
00270      03 SR405      PIC X(16)
00271       VALUE 'DENTAL-CLAIM' .
00272      03 SR400      PIC X(16)
00273       VALUE 'COVERAGE' .
00274
DMLC 00275 01 COPY IDMS SUBSCHEMA-SETNAMES.
00276 01 SUBSCHEMA-SETNAMES.
DMLC
00306  01 COPY IDMS RECORD EMPLOYEE.
00307
00308  01 EMPLOYEE.
00309
00310  02 EMP-ID-0415     PIC 9(4).
00311  02 EMP-NAME-0415.
00312  03 EMP-FIRST-NAME-0415 PIC X(10).
00313  03 EMP-LAST-NAME-0415 PIC X(15).
00314  02 EMP-ADDRESS-0415.
00315  03 EMP-STREET-0415 PIC X(20).
00316  03 EMP-CITY-0415    PIC X(15).
00317  03 EMP-STATE-0415   PIC X(2).
00318  03 EMP-ZIP-0415.
00319  04 EMP-ZIP-FIRST-FIVE-0415 PIC X(5).
00320  04 EMP-ZIP-LAST-FOUR-0415 PIC X(4).
00321  02 EMP-PHONE-0415   PIC 9(10).
00322  02 STATUS-0415     PIC X(2).
00323  08 ACTIVE-0415    VALUE ‘01’.  
00324
00325  08 ST-DISABIL-0415  VALUE ‘02’.
00326
00327  08 LT-DISABIL-0415  VALUE ‘03’.  
00328
00329  08 LEAVE-OF-ABSENCE-0415  VALUE ‘04’.  
00330
00331  08 TERMINATED-0415  VALUE ‘05’.  
00332
00333  02 SS-NUMBER-0415   PIC 9(9).
00334  02 START-DATE-0415.
00335
00336  03 START-YEAR-0415   PIC 9(4).
00337  03 START-MONTH-0415   PIC 9(2).
00338  03 START-DAY-0415     PIC 9(2).
00339  02 TERMINATION-DATE-0415.
00340  03 TERMINATION-YEAR-0415 PIC 9(4).
00341  03 TERMINATION-MONTH-0415 PIC 9(2).
00342  03 TERMINATION-DAY-0415 PIC 9(2).
00342  02 BIRTH-DATE-0415.
00343  03 BIRTH-YEAR-0415    PIC 9(4).
00344  03 BIRTH-MONTH-0415   PIC 9(2).
00345  03 BIRTH-DAY-0415     PIC 9(2).
00346
DMLC  00347  01 COPY IDMS RECORD DEPARTMENT.
00348   01 DEPARTMENT.
00349   02 DEPT-ID-0410      PIC 9(4).
00350   02 DEPT-NAME-0410    PIC X(45).
00351   02 DEPT-HEAD-ID-0410 PIC 9(4).
00352   02 FILLER            PIC XXX.
00353
DMLC  00354  01 COPY IDMS RECORD JOB.
00355   01 JOB.
00356   02 JOB-ID-0440       PIC 9(4).
00357   02 TITLE-0440       PIC X(20).
00358   02 DESCRIPTION-0440.
00359   03 DESCRIPTION-LINE-0440 PIC X(60) OCCURS 2.
00360   02 REQUIREMENTS-0440.
00361   03 REQUIREMENT-LINE-0440 PIC X(60) OCCURS 2.
00362   02 MINIMUM-SALARY-0440 PIC S9(6)V99.
00363   02 MAXIMUM-SALARY-0440 PIC S9(6)V99.
00364
DMLC  00365   02 SALARY-GRADES-0440 PIC 9(2) OCCURS 4.
00366   02 NUMBER-OF-POSITIONS-0440 PIC 9(3).
00367
DMLC  00368   02 START-DATE-0420.
00369   03 START-YEAR-0420   PIC 9(4).
00370   03 START-MONTH-0420  PIC 9(2).
00371
DMLC  00372   03 START-DAY-0420   PIC 9(2).
00373   02 FINISH-DATE-0420.
00374   03 FINISH-YEAR-0420  PIC 9(4).
00375   03 FINISH-MONTH-0420 PIC 9(2).
00376   03 FINISH-DAY-0420   PIC 9(2).
00377
DMLC  00378   02 SALARY-GRADE-0420 PIC 9(2).
00379   02 SALARY-AMOUNT-0420 PIC S9V999 COMP-3.
00380   02 BONUS-PERCENT-0420 PIC SV999 COMP-3.
00381   02 COMMISSION-PERCENT-0420 PIC SV999 COMP-3.
00382
DMLC  00383   02 OVERTIME-RATE-0420 PIC S9V99 COMP-3.
00384   02 FILLER            PIC XXX.
00385
DMLC  00386   01 COPY IDMS RECORD OFFICE.
00387   01 OFFICE.
00388   02 OFFICE-CODE-0450   PIC X(3).
00389   02 OFFICE-ADDRESS-0450.
00390   02 OFFICE-AREA-CODE-0450.
00391   04 OFFICE-ZIP-FIRST-FIVE-0450 PIC X(5).
00392   04 OFFICE-ZIP-LAST-FOUR-0450 PIC X(4).
00393   04 OFFICE-ZIP-LAST-FOUR-0450 PIC X(4).
00394   04 OFFICE-AREA-CODE-0450 PIC X(3).
00395
DMLC  00396   02 SPEED-DIAL-0450   PIC X(3).
00397
00406          02 FILLER PIC X(4).
00407            EJECT
00408          PROCEDURE DIVISION.

00409          *  ******************************************************
00410          *  * PROCEDURE DIVISION GENERAL STRATEGY:    *
00411          *  * 1) READ DEPT-ID-IN, WHICH CONTAINS THE  *
00412          *  *    DEPT-ID NUMBER      *
00413          *  * 2) ACCESS THE DATABASE USING THE DEPT-ID NUMBER *
00414          *  *    WITH AN OBTAIN CALC ON THE DEPARTMENT RECORD *
00415          *  * 3) ACCESS ALL EMPLOYEES IN THE DEPT-EMPLOYEE SET *
00416          *  *    AND RETRIEVE RELATED JOB AND OFFICE DATA *
00417          *  * 4) PRINT A REPORT FOR EACH DEPARTMENT *
00418          *  * 5) PRINT AN ERROR REPORT FOR EMPTY DEPARTMENTS *
00419          *  *    AND NONEXISTENT DEPARTMENTS (NO MATCHING  *
00420          *  *    DEPT-ID) *
00421          *  ******************************************************

00422          MAIN-LINE.
00423            PERFORM INIT-FILES.
00424            IF END-OF-FILE
00425            PERFORM EMPTY-INPUT-PROCESSING
00426            ELSE
00427            PERFORM INIT-BIND-READY
00428              PERFORM U220-ERR-HEADER
00429          END-PROCESSING.
00430            GOBACK.

00431          INIT-BIND-READY.

00436          ************************************************************************
00437          * THE BIND STATEMENTS ARE PERFORMED INDIVIDUALLY (RATHER    *
00438          * THAN BY USING A COPY IDMS SUBSCHEMA-BINDS) IN ORDER TO    *
00439          * CHECK EACH ERROR-STATUS BY PERFORMING THE IDMS-STATUS    *
00440          * ROUTINE.                                             *
00441          ************************************************************************

00442              MOVE 'DEPTRPT' TO PROGRAM-NAME.
00443              BIND RUN-UNIT.
00444              MOVE 1 TO DML-SEQUENCE
00445              CALL 'IDMS' USING SUBSCHEMA-CTRL
00446              IDBMSCOM (59)
00447              SUBSCHEMA-CTRL
00448              SUBSCHEMA-SSNAME.
00449              PERFORM IDMS-STATUS.

00450              BIND EMPLOYEE.
00451              MOVE 2 TO DML-SEQUENCE
00452              CALL 'IDMS' USING SUBSCHEMA-CTRL
00453              IDBMSCOM (48)

00454              SR415
00455              EMPLOYEE.
00456              PERFORM IDMS-STATUS.

00457              BIND DEPARTMENT.
00458              MOVE 3 TO DML-SEQUENCE
00459              CALL 'IDMS' USING SUBSCHEMA-CTRL
00460              IDBMSCOM (48)
00461              SR410
00462              DEPARTMENT.
00463              PERFORM IDMS-STATUS.

00464              BIND JOB.
00465              MOVE 4 TO DML-SEQUENCE
00466              CALL 'IDMS' USING SUBSCHEMA-CTRL

00467              IDBMSCOM (48)
00468              SR440
00469              JOB.
00470  PERFORM IDMS-STATUS.
00471  BIND EMPOSITION.
    MOVE 5 TO DML-SEQUENCE
    CALL 'IDMS' USING SUBSCHEMA-CTRL
    IDBMSCOM (48)
    SR420
    EMPOSITION.
00477  PERFORM IDMS-STATUS.

DMLC0006
00478  BIND OFFICE.
    MOVE 6 TO DML-SEQUENCE
    CALL 'IDMS' USING SUBSCHEMA-CTRL
    IDBMSCOM (37).
00479  PERFORM IDMS-STATUS.

00485  READY.
    MOVE 7 TO DML-SEQUENCE
    CALL 'IDMS' USING SUBSCHEMA-CTRL
    IDBMSCOM (48)
    SR450
    OFFICE.
00484  PERFORM IDMS-STATUS.

00490  INIT-FILES.
00492  OPEN INPUT DEPT-FILE-IN.
00493  OPEN OUTPUT DEPT-FILE-OUT.
00494  OPEN OUTPUT ERR-FILE-OUT.
00495  MOVE SPACES TO PRINT-LINE.
00496  MOVE SPACES TO ERR-LINE.
00497  READ DEPT-FILE-IN AT END MOVE 'Y' TO EOF-SW.
00498  EMPTY-INPUT-PROCESSING.
00499  MOVE NO-INPUT-MESSAGE TO PRINT-LINE.
00501  MOVE '1' TO CC.
00502  PERFORM U000-WRITE-LINE.

00503

***********************************************************************
00504  * THIS PARAGRAPH ACCESSES THE DATABASE USING THE DEPT-ID-0415 *
00505  * CALCKEY VALUE.           *                             *
00506  ***********************************************************************
00507  DEPT-PROCESSING.
00508  MOVE DEPT-ID-IN TO DEPT-ID-0410.

DMLC0008
00509  OBTAIN CALC DEPARTMENT.
    MOVE 8 TO DML-SEQUENCE
    CALL 'IDMS' USING SUBSCHEMA-CTRL
    IDBMSCOM (32)
    SR410
    IDBMSCOM (43).
00514
    IF DB-REC-NOT-FOUND THEN
    00516  PERFORM NO-DEPT-PROCESSING
00517  ELSE
    00518  PERFORM IDMS-STATUS
00519  IF DEPT-EMPLOYEE IS NOT EMPTY
00520  MOVE DEPT-ID-IN TO DEPT-ID-0410.
00521  CALL 'IDMS' USING SUBSCHEMA-CTRL
00522  IDBMSCOM (65)
00523  DEPT-EMPLOYEE;
00524  IF ERROR-STATUS EQUAL TO '1601'
00525  THEN
00526  PERFORM U020-VALID-HEADER
00527  MOVE DEPT-ID-0410 TO DEPT-ID-OUT
00528  MOVE DEPT-NAME-0410 TO DEPT-NAME-OUT
00529  MOVE DEPT-DETAIL-LINE TO PRINT-LINE
00530  PERFORM U000-WRITE-LINE
00531  PERFORM U030-EMP-HEADERS
00532  PERFORM SET-WALK THRU SET-WALK-EXIT
00533   UNTIL DB-END-OF-SET
00534  ELSE

00535 PERFORM EMPTY-SET.
00536 READ DEPT-FILE-IN AT END MOVE 'Y' TO EOF-SW.
00537 DEPT-PROCESSING-EXIT.
00538 EXIT.
00539 *****************************************************
00540 * THIS PARAGRAPH RETRIEVES EMPLOYEE, JOB, AND OFFICE DATA *
00541 * FOR EACH EMPLOYEE IN THE DEPT-EMPLOYEE SET. *
00542 *****************************************************
00543 SET-WALK.

DMLC0010 00544 OBTAIN NEXT EMPLOYEE WITHIN DEPT-EMPLOYEE.
00545 MOVE 10 TO DML-SEQUENCE
00546 CALL 'IDMS' USING SUBSCHEMA-CTRL
00547 IDBMSCOM (10)
00548 SR415
00549 DEPT-EMPLOYEE
00550 IDBMSCOM (43).
00551 IF DB-END-OF-SET
00552 GO TO SET-WALK-EXIT
00553 ELSE
00554 PERFORM IDMS-STATUS.
00555 MOVE EMP-0415 TO ID-OUT.
00556 MOVE EMP-LAST-NAME-0415 TO LAST-OUT.
00557 MOVE EMP-FIRST-NAME-0415 TO FIRST-OUT.
00558 MOVE START-YEAR-0415 TO SD-YY.
00559 MOVE START-MONTH-0415 TO SD-MM.
00560 MOVE START-DAY-0415 TO SD-DD.

DMLC0011 00561 IF EMP-EMPOSITION IS EMPTY
00562 MOVE 11 TO DML-SEQUENCE
00563 CALL 'IDMS' USING SUBSCHEMA-CTRL
00564 IDBMSCOM (64)
00565 EMP-EMPOSITION;
00566 IF ERROR-STATUS EQUAL TO '0000'
00567 MOVE NO-JOB-MESSAGE TO TITLE-OUT
00568 ELSE
00569 FIND FIRST WITHIN EMP-EMPOSITION
00570 MOVE 12 TO DML-SEQUENCE
00571 CALL 'IDMS' USING SUBSCHEMA-CTRL
00572 IDBMSCOM (20)
00573 EMP-EMPOSITION;
00574 PERFORM IDMS-STATUS
00575 IF NOT JOB-EMPOSITION MEMBER
00576 MOVE 13 TO DML-SEQUENCE
00577 CALL 'IDMS' USING SUBSCHEMA-CTRL
00578 IDBMSCOM (62)
00579 JOB-EMPOSITION;
00580 IF ERROR-STATUS EQUAL TO '1601'
00581 MOVE NO-JOB-MESSAGE TO TITLE-OUT
00582 ELSE
00583 OBTAIN OWNER WITHIN JOB-EMPOSITION
00584 MOVE 14 TO DML-SEQUENCE
00585 CALL 'IDMS' USING SUBSCHEMA-CTRL
00586 IDBMSCOM (31)
00587 JOB-EMPOSITION
00588 IDBMSCOM (43);
00589 PERFORM IDMS-STATUS
00590 MOVE TITLE-0440 TO TITLE-OUT.

DMLC0014 00591 IF OFFICE-EMPLOYEE IS EMPTY
00592 MOVE 15 TO DML-SEQUENCE
00593 CALL 'IDMS' USING SUBSCHEMA-CTRL
00594 IDBMSCOM (64)
00595 OFFICE-EMPLOYEE;
00596 IF ERROR-STATUS EQUAL TO '0000'
00597 MOVE NO-OFFICE-MESSAGE TO STREET-OUT
00598 MOVE SPACES TO CITY-OUT
00599 MOVE SPACES TO STATE-OUT
00600  MOVE SPACES TO ZIP-OUT
00601  ELSE
DMLC0016  00602  OBTAIN OWNER WITHIN OFFICE-EMPLOYEE
00603  MOVE 16 TO DML-SEQUENCE
00604  CALL 'IDMS' USING SUBSCHEMA-CTRL
00605         IDBMSCOM (31)
00606  OFFICE-EMPLOYEE
00607         IDBMSCOM (43);
00608  PERFORM IDMS-STATUS
00609  MOVE OFFICE-STATE-0450 TO STATE-OUT
00610  MOVE OFFICE-CITY-0450 TO CITY-OUT
00611  MOVE OFFICE-STATE-0450 TO STATE-OUT
00612  MOVE OFFICE-ZIP-FIRST-FIVE-0450 TO ZIP-OUT
00613  MOVE EMP-DETAIL-LINE TO PRINT-LINE.
00614  PERFORM U000-WRITE-LINE.
00615  SET-WALK-EXIT.
00616  EXIT.
00617
00618  END-PROCESSING.
DMLC0017  00619  FINISH.
00620  MOVE 17 TO DML-SEQUENCE
00621  CALL 'IDMS' USING SUBSCHEMA-CTRL
00622         IDBMSCOM (2).
00623  PERFORM IDMS-STATUS.
00624  CLOSE DEPT-FILE-OUT.
00625  CLOSE ERR-FILE-OUT.
00626  CLOSE DEPT-FILE-IN.
00627
00628  EMPTY-SET.
00629  MOVE SPACES TO ERR-LINE.
00630  MOVE DEPT-ID-0410 TO ERR-ID-OUT.
00631  MOVE NO-EMP-MESSAGE TO ERR-MESS-OUT.
00632  MOVE ERR-DETAIL-LINE TO ERR-LINE.
00633  PERFORM U200-WRITE-ERR-LINE.
00634
00635  NO-DEPT-PROCESSING.
00636  MOVE DEPT-ID-IN TO ERR-ID-OUT.
00637  MOVE NO-DEPT-MESSAGE TO ERR-MESS-OUT.
00638  MOVE ERR-DETAIL-LINE TO ERR-LINE.
00639  PERFORM U200-WRITE-ERR-LINE.
00640
00641  U000-WRITE-LINE.
00642  WRITE DEPT-REC-OUT AFTER POSITIONING CC.
00643  IF CC = '1' THEN MOVE 0 TO LINE-COUNT
00644  ELSE IF CC = ' ' THEN ADD 1 TO LINE-COUNT
00645  ELSE IF CC = '0' THEN ADD 2 TO LINE-COUNT.
00646  IF LINE-COUNT > LINE-MAX
00647      THEN PERFORM U010-NEW-PAGE-Routine.
00648
00649  U010-NEW-PAGE-Routine.
00650  PERFORM U020-VALID-HEADER.
00651  MOVE DEPT-DETAIL-LINE TO PRINT-LINE.
00652  PERFORM U000-WRITE-LINE.
00653  PERFORM U030-EMP-HEADERS.
00654  U020-VALID-HEADER.
00655  MOVE DEPT-HEADER TO PRINT-LINE.
00656  MOVE '1' TO CC.
00657  PERFORM U000-WRITE-LINE
00658  MOVE ' ' TO CC.
00659  U030-EMP-HEADERS.
00660  MOVE '0' TO CC.
00661  MOVE EMP-HEADER TO PRINT-LINE.
00662  PERFORM U000-WRITE-LINE.
00663  MOVE ' ' TO CC.
00664  PERFORM U000-WRITE-LINE.
00665
00666  U200-WRITE-ERR-LINE.
CA IDMS Reference - 19.0

00667    WRITE ERR-REC-OUT AFTER POSITIONING ERR-CC.
00668    IF ERR-CC = '1' THEN MOVE 0 TO ERR-LINE-COUNT
00669    ELSE IF ERR-CC = ' ' THEN ADD 1 TO ERR-LINE-COUNT
00670    ELSE IF ERR-CC = '0' THEN ADD 2 TO ERR-LINE-COUNT.
00671    IF ERR-LINE-COUNT > LINE-MAX THEN
00672    PERFORM U220-ERR-HEADER.
00673    MOVE ERR-LINE-1 TO ERR-LINE.
00674    MOVE '1' TO ERR-CC.
00675    PERFORM U200-WRITE-ERR-LINE.
00676    MOVE '0' TO ERR-CC.
00677    PERFORM U200-WRITE-ERR-LINE.
00678    MOVE ERR-HEADER-2 TO ERR-LINE.
00679    PERFORM U200-WRITE-ERR-LINE.
00680    MOVE ERR-HEADER-3 TO ERR-LINE.
00681    EXIT.
00682    IDMS-ABORT.
00683    IDMS-ABORT-EXIT.
00684    IDMS-STATUS.
00685    COPY IDMS IDMS-STATUS.
00686
******************************************************************
00687    IDMS-STATUS-PARAGRAPH.
00688    IF DB-STATUS-OK GO TO ISABEX.
00689    PERFORM IDMS-ABORT.
00690    DISPLAY '**************************'
00691    ' ABORTING - ' PROGRAM-NAME
00692    ', ' ERROR-STATUS
00693    ', ' ERROR-RECORD
00694    ' **** RECOVER IDMS ****'
00695    UPON CONSOLE.
00696    DISPLAY 'PROGRAM NAME ------ ' PROGRAM-NAME.
00697    DISPLAY 'ERROR STATUS ------ ' ERROR-STATUS.
00698    DISPLAY 'ERROR RECORD ------ ' ERROR-RECORD.
00699    DISPLAY 'ERROR SET ------ ' ERROR-SET.
00700    DISPLAY 'ERROR AREA ------ ' ERROR-AREA.
00701    DISPLAY 'LAST GOOD RECORD --- ' RECORD-NAME.
00702    DISPLAY 'LAST GOOD AREA ---- ' AREA-NAME.
00703    MOVE 39 TO SSC-IN01-REQ-CODE.
00704    MOVE 0 TO SSC-IN01-REQ-RETURN.
00705    MOVE ' ' TO SSC-STATUS-LABEL.
00706    PERFORM IDMS-STATUS-LOOP UNTIL SSC-IN01-REQ-RETURN > 0.
00707    IDMS-STATUS-LOOP.
00708    CALL 'IDMSIN1' USING IDBMSCOM(41)
00709    SSC-IN01-REQ-WK
00710    SUBSCHEMA-CTRL
00711    IDBMSCOM(67).
00712    CALL 'ABORT'.
00713    GO TO ISABEX.
00714    MOVE 18 TO DML-SEQUENCE.
00715    CALL 'IDMS' USING SUBSCHEMA-CTRL
00716    IDBMSCOM (67).
00717    CALL 'ABORT'.
00718    GO TO ISABEX.
00719    IDMS-STATUS-LOOP.
00720    CALL 'IDMSIN1' USING IDBMSCOM(41)
00721    SSC-IN01-REQ-WK
00722    SUBSCHEMA-CTRL
00723    IDBMSCOM(1)
00724    DML-SEQUENCE.
00725    SSC-STATUS-LINE.
00726    IF SSC-IN01-REQ-RETURN GREATER THAN 4
00727    DISPLAY 'DML SEQUENCE ------ ' DML-SEQUENCE.
00728    ELSE
00729    DISPLAY SSC-STATUS-LABEL '---- ' SSC-STATUS-VALUE.
00730    ISABEX. EXIT.

NO MESSAGES FOR PROGRAM DEPTRPT
Sample Batch Program from the COBOL Precompiler

The following listing illustrates the sample batch program after precompilation by the COBOL precompiler. The original code is further expanded and includes the following:

- Line numbers generated by the COBOL compiler
- CA IDMS call statements for the requested DML functions
- Diagnostic messages

⚠️ **Note:** For more information about expanded code generated by the DML compiler, see CA IDMS Call Formats (see page 358).

This listing contains the sample program output from the COBOL compiler with the fully expanded code (including the calls to CA IDMS) generated by the DML compiler.

```cobol
00001 *DMLIST
00002 *NO-ACTIVITY-LOG
00003 *SCHEMA-COMMENTS
00004
00005 IDENTIFICATION DIVISION.
00006
00007 PROGRAM-ID.       DEPTRPT.
00008
00009 AUTHOR.           COMPUTER ASSOCIATES INTERNATIONAL.
00010
00011 DATE-WRITTEN.     APRIL 1995.
00012
00013 REMARKS.          THIS PROGRAM DEMONSTRATES
00014           CA IDMS DATABASE ACCESS USING
00015           COBOL DML STATEMENTS. IT READS
00016           DEPARTMENT ID NUMBERS AND RETRIEVES
00017           RELATED RECORD OCCURRENCES,
00018           PRINTING A REPORT THAT INCLUDES
00019           DEPARTMENT, EMPLOYEE, JOB, AND
00020           OFFICE INFORMATION.
00021 ***************************************************************
00022 ENVIRONMENT DIVISION.
00023 INPUT-OUTPUT SECTION.
00024 FILE-CONTROL.
00025 SELECT DEPT-FILE-IN ASSIGN TO INFILE.
00026 SELECT DEPT-FILE-OUT ASSIGN TO OUTFILE.
00027 SELECT ERR-FILE-OUT ASSIGN TO ERRFILE.
00028 ***************************************************************
00029 *IDMS-CONTROL SECTION.
00030 *
00031 *PROTOCOL.       MODE IS BATCH DEBUG
00032 *
00033 ***************************************************************
00034 DATA DIVISION.
00035
00036 *SCHEMA SECTION.
00037 *
00038 *DB EMPSS01 WITHIN EMPSCHM.
00039
00040 FILE SECTION.
00041
```
FD DEPT-FILE-IN
01 DEPT-REC-IN.
  02 DEPT-ID-IN PIC 9(4).
  02 DEPT-IN-FILLER PIC X(76).
FD DEPT-FILE-OUT
01 DEPT-REC-OUT.
  02 CC PIC X.
  02 PRINT-LINE PIC X(132).
FD ERR-FILE-OUT
01 ERR-REC-OUT.
  02 ERR-CC PIC X.
  02 ERR-LINE PIC X(132).
***************************************************************
WORKING-STORAGE SECTION.
01 EOF-SW PIC X VALUE 'N'.
  88 END-OF-FILE VALUE 'Y'.
01 LINE-COUNT PIC 99 VALUE 0.
01 ERR-LINE-COUNT PIC 99 VALUE 0.
01 LINE-MAX PIC 99 VALUE 50.
***************************************************************
01 DEPT-HEADER.
  05 FILLER PIC X(30) VALUE SPACES.
  05 FILLER PIC X(13) VALUE 'DEPARTMENT ID'.
  05 FILLER PIC X(10) VALUE 'DEPT NAME'.
  05 FILLER PIC X(9) VALUE SPACES.
  05 FILLER PIC X(70) VALUE SPACES.
01 DEPT-DETAIL-LINE.
  05 FILLER PIC X(33) VALUE SPACES.
  05 FILLER PIC X(16) VALUE SPACES.
  05 FILLER PIC X(45).
  05 FILLER PIC X(34) VALUE SPACES.
01 EMP-HEADER.
  05 FILLER PIC X(5) VALUE SPACES.
  05 FILLER PIC X(6) VALUE 'EMP ID'.
  05 FILLER PIC X(2) VALUE SPACES.
  05 FILLER PIC X(9) VALUE 'LAST NAME'.
  05 FILLER PIC X(8) VALUE SPACES.
  05 FILLER PIC X(10) VALUE 'FIRST NAME'.
  05 FILLER PIC X(3) VALUE SPACES.
  05 FILLER PIC X(10) VALUE 'START DATE'.
  05 FILLER PIC X(2) VALUE SPACES.
  05 FILLER PIC X(9) VALUE 'JOB TITLE'.
  05 FILLER PIC X(13) VALUE SPACES.
  05 FILLER PIC X(14) VALUE 'OFFICE ADDRESS'.
  05 FILLER PIC X(42) VALUE SPACES.
01 EMP-DETAIL-LINE.
  05 FILLER PIC X(5) VALUE SPACES.
  05 FILLER PIC X(4).
  05 FILLER PIC X(4) VALUE SPACES.
  05 FILLER PIC X(15).
  05 FILLER PIC X(2) VALUE SPACES.
00115  05 FILLER PIC X(3) VALUE SPACES.
00116  05 SD-OUT.
00117   10 SD-MM PIC XX.
00118   10 FILLER PIC X VALUE '/'.
00119   10 SD-DD PIC XX.
00120   10 FILLER PIC X VALUE '/'.
00121   10 SD-YY PIC XX.
00122  05 FILLER PIC X(4) VALUE SPACES.
00123  05 TITLE-OUT PIC X(20).
00124  05 OFF-ADDRESS-OUT.
00125   10 STREET-OUT PIC X(20).
00126   10 FILLER PIC XX VALUE SPACES.
00127   10 CITY-OUT PIC X(15).
00128   10 FILLER PIC XX VALUE SPACES.
00129   10 STATE-OUT PIC XX.
00130   10 FILLER PIC XX VALUE SPACES.
00131   10 ZIP-OUT PIC X(5).
00132  05 FILLER PIC X(20) VALUE SPACES.
00133  01 ERR-HEADER-1.
00134   05 FILLER PIC X(40) VALUE SPACES.
00135   05 FILLER PIC X(12) VALUE 'ERROR REPORT'.
00136   05 FILLER PIC X(80) VALUE SPACES.
00137  01 ERR-HEADER-2.
00138   05 FILLER PIC X(10) VALUE SPACES.
00139   05 FILLER PIC X(4) VALUE '***'.
00140   05 FILLER PIC X(51) VALUE
00141   'THIS REPORT LISTS EMPTY AND NONEXISTENT DEPARTMENTS'.
00142   05 FILLER PIC X(4) VALUE '***'.
00143   05 FILLER PIC X(63) VALUE SPACES.
00144  01 ERR-HEADER-3.
00145   05 FILLER PIC X(20) VALUE SPACES.
00146   05 FILLER PIC X(7) VALUE 'DEPT ID'.
00147   05 FILLER PIC X(9) VALUE 'MESSAGE'.
00148   05 FILLER PIC X(89) VALUE SPACES.
00149  01 ERR-DETAIL-LINE.
00150   05 FILLER PIC X(20) VALUE SPACES.
00151   05 ERR-ID-OUT PIC X(4).
00152   05 FILLER PIC X(12) VALUE SPACES.
00153   05 ERR-MESS-OUT PIC X(15).
00154   05 FILLER PIC X(79) VALUE SPACES.
00155  01 MESSAGES.
00156   05 NO-JOB-MESSAGE.
00157      10 FILLER PIC X(20) VALUE 'NO JOB ASSIGNED'.
00158   05 NO-OFFICE-MESSAGE.
00159      10 FILLER PIC X(20)
00160      VALUE 'NO OFFICE ASSIGNED'.
00161   05 NO-DEPT-MESSAGE.
00162      10 FILLER PIC X(15)
00163      VALUE 'DOES NOT EXIST'.
00164   05 NO-EMP-MESSAGE.
00165      10 FILLER PIC X(15)
00166      VALUE 'IS EMPTY'.
00167   05 NO-INPUT-MESSAGE.
00168      10 FILLER PIC X(100)
00169      VALUE SPACES.
00170   01 COPY IDMS SUBSCHEMA-CTRL.
00171   01 SUBSCHEMA-CTRL.
00172      03 PROGRAM-NAME PIC X(8)
00173      VALUE SPACES.
00174      03 ERROR-STATUS PIC X(4)
00175      VALUE '1400'.
00176      88 DB-STATUS-OK
00177      VALUE '0000'.
00178      88 ANY-STATUS
00179      88 ANY-ERROR-STATUS
00180      VALUE ' ' THRU '9999'.
00181      88 ANY-ERROR-STATUS
CA IDMS Reference - 19.0

00186      VALUE '0001' THRU '9999'.
00187      88 DB-END-OF-SET
00188      VALUE '0307'.
00189      88 DB-REC-NOT-FOUND
00190      VALUE '0326'.
00191      03 DBKEY PIC S9(8) COMP Sync.
00192      03 RECORD-NAME PIC X(16)
00193      VALUE SPACES .
00194      03 RECORD-NAME REDEFINES RECORD-NAME.
00195      05 SSC-NODN PIC X(8).
00196      05 SSC-DBN PIC X(8).
00197      03 AREA-NAME PIC X(16)
00198      VALUE SPACES .
00199      03 AREA-RNAME REDEFINES AREA-NAME.
00200      05 SSC-DO PIC X(8).
00201      05 SSC-DNA PIC X(8).
00202      03 ERROR-SET PIC X(16)
00203      VALUE SPACES .
00204      03 ERROR-RECORD PIC X(16)
00205      VALUE SPACES .
00206      03 ERROR-AREA PIC X(16)
00207      VALUE SPACES .
00208      03 IDBMSCOM-AREA PIC X(100)
00209      VALUE LOW-VALUE .
00210      03 IDBMSCOM REDEFINES IDBMSCOM-AREA
00211      PIC X
00212      OCCURS 100.
00213      03 RIDBMSCOM REDEFINES IDBMSCOM-AREA.
00214      05 DB-SUB-ADDR PIC X(4).
00215      05 FILLER PIC X(96).
00216      03 RIDBMSCOM REDEFINES IDBMSCOM-AREA.
00217      05 PAGE-INFO.
00218      07 PAGE-INFO-GROUP PIC S9(4) COMP.
00219      07 PAGE-INFO-DBK-FORMAT
00220      PIC 9(4) COMP.
00221      05 SSC-IDMS-STATUS-WRK.
00222      07 SSC-STATMENT-CODE-REQ.
00223      09 SSC-STATMENT-CODE-REQ-CODE
00224      PIC S9(8) COMP.
00225      09 SSC-STATMENT-CODE-REQ-RETURN
00226      PIC S9(8) COMP.
00227      07 SSC-STATUS-LINE.
00228      09 SSC-STATUS-LABEL PIC X(16).
00229      09 SSC-STATUS-VALUE PIC X(12).
00230      05 FILLER PIC X(60).
00231      03 DIRECT-DBKEY PIC S9(8) COMP Sync.
00232      03 DIRECT-DBKEY REDEFINES DIRECT-DBKEY
00233      PIC S9(8) COMP Sync.
00234      03 DATABASE-STATUS.
00235      05 DBSTATMENT-CODE PIC X(2).
00236      05 DBSTATMENT-CODE PIC X(5).
00237      03 FILLER PIC X.
00238      03 RECORD-OCCUR PIC S9(8) COMP Sync.
00239      03 DML-SEQUENCE PIC S9(8) COMP Sync.
00240      *01 COPY IDMS SUBSCHEMA-SSNAME.
00241      01 SUBSCHEMA-SSNAME PIC X(8)
00242      VALUE 'EMPSS01' .
00243      *01 COPY IDMS SUBSCHEMA-RECNAMES.
00244      01 SUBSCHEMA-RECNAMES.
00245      03 SR460 PIC X(16)
00246      03 SR455 PIC X(16)
00247      03 SR450 PIC X(16)
00248      03 SR445 PIC X(16)
00249      03 SR440 PIC X(16)
00250      03 SR435 PIC X(16)
00251      VALUE 'STRUCTURE' .
00252      VALUE 'SKILL' .
00253      VALUE 'OFFICE' .
00254      VALUE 'NON-HOSP-CLAIM' .
00255      VALUE 'JOB' .
00257 03 SR435 PIC X(16) VALUE 'INSURANCE-PLAN'.
00258 03 SR430 PIC X(16) VALUE 'HOSPITAL-CLAIM'.
00259 03 SR425 PIC X(16) VALUE 'EXPERTISE'.
00260 03 SR420 PIC X(16) VALUE 'EMPOSITION'.
00261 03 SR415 PIC X(16) VALUE 'EMPLOYEE'.
00262 03 SR410 PIC X(16) VALUE 'DEPARTMENT'.
00263 03 SR405 PIC X(16) VALUE 'DENTAL-CLAIM'.
00264 03 SR400 PIC X(16) VALUE 'COVERAGE'.

*01 COPY IDMS SUBSCHEMA-SETNAMES.
01 SUBSCHEMA-SETNAMES.
00274 03 COVERAGE-CLAIMS PIC X(16) VALUE 'COVERAGE-CLAIMS'.
00275 03 DEPT-EMPLOYEE PIC X(16) VALUE 'DEPT-EMPLOYEE'.
00276 03 EMP-COVERAGE PIC X(16) VALUE 'EMP-COVERAGE'.
00277 03 EMP-EXPERTISE PIC X(16) VALUE 'EMP-EXPERTISE'.
00278 03 EMP-NAME-NDX PIC X(16) VALUE 'EMP-NAME-NDX'.
00279 03 EMP-EMPOSITION PIC X(16) VALUE 'EMP-EMPOSITION'.
00280 03 JOB-EMPOSITION PIC X(16) VALUE 'JOB-EMPOSITION'.
00281 03 JOB-TITLE-NDX PIC X(16) VALUE 'JOB-TITLE-NDX'.
00282 03 MANAGES PIC X(16) VALUE 'MANAGES'.
00283 03 OFFICE-EMPLOYEE PIC X(16) VALUE 'OFFICE-EMPLOYEE'.
00284 03 REPORTS-TO PIC X(16) VALUE 'REPORTS-TO'.
00285 03 SKILL-EXPERTISE PIC X(16) VALUE 'SKILL-EXPERTISE'.
00286 03 SKILL-NAME-NDX PIC X(16) VALUE 'SKILL-NAME-NDX'.
00287 03 CALC PIC X(16) VALUE 'CALC'.

*01 COPY IDMS RECORD EMPLOYEE.
01 EMPLOYEE.
00288 02 EMP-ID-0415 PIC 9(4).
00289 02 EMP-FIRST-NAME-0415 PIC X(10).
00290 02 EMP-LAST-NAME-0415 PIC X(15).
00291 02 EMP-ADDRESS-0415 PIC X(20).
00292 02 EMP-CITY-0415 PIC X(15).
00293 02 EMP-STATE-0415 PIC X(2).
00294 02 EMP-ZIP-0415 PIC X(5).
00295 02 EMP-ZIP-FIRST-FIVE-0415 PIC X(4).
00296 02 EMP-PHONE-0415 PIC 9(10).
00297 02 STATUS-0415 PIC X(2).
00298 88 ACTIVE-0415 VALUE '01'.
00299 88 ST-DISABIL-0415 VALUE '02'.
00300 88 LT-DISABIL-0415 VALUE '03'.

15-Jan-2018 321/415
00328          88 LEAVE-OF-ABSENCE-0415
00329          VALUE '04'.
00330          88 TERMINATED-0415
00331          VALUE '05'.
00332          02 SS-NUMBER-0415 PIC 9(9).
00333          02 START-DATE-0415.
00334          03 START-YEAR-0415 PIC 9(4).
00335          03 START-MONTH-0415 PIC 9(2).
00336          03 START-DAY-0415 PIC 9(2).
00337          02 TERMINATION-DATE-0415.
00338          03 TERMINATION-YEAR-0415 PIC 9(4).
00339          03 TERMINATION-MONTH-0415 PIC 9(2).
00340          03 TERMINATION-DAY-0415 PIC 9(2).
00341          02 BIRTH-DATE-0415.
00342          03 BIRTH-YEAR-0415 PIC 9(4).
00343          03 BIRTH-MONTH-0415 PIC 9(2).
00344          03 BIRTH-DAY-0415 PIC 9(2).
00345
00346          *01 COPY IDMS RECORD DEPARTMENT.
00347          01 DEPARTMENT.
00348          02 DEPT-ID-0410 PIC 9(4).
00349          02 DEPT-NAME-0410 PIC X(45).
00350          02 DEPT-HEAD-ID-0410 PIC 9(4).
00351          02 FILLER PIC XXX.
00352
00353          *01 COPY IDMS RECORD JOB.
00354          01 JOB.
00355          02 JOB-ID-0440 PIC 9(4).
00356          02 TITLE-0440 PIC X(20).
00357          02 DESCRIPTION-0440.
00358          03 DESCRIPTION-LINE-0440 PIC X(60)
00359           OCCURS 2.
00360          02 REQUIREMENTS-0440.
00361          03 REQUIREMENT-LINE-0440 PIC X(60)
00362           OCCURS 2.
00363          02 MINIMUM-SALARY-0440 PIC S9(6)V99.
00364          02 MAXIMUM-SALARY-0440 PIC S9(6)V99.
00365          02 SALARY-GRADES-0440 PIC 9(2)
00366           OCCURS 4.
00367          02 NUMBER-OF-POSITIONS-0440
00368           PIC 9(3).
00369          02 NUMBER-OPEN-0440 PIC 9(3).
00370          02 FILLER PIC XX.
00371
00372          *01 COPY IDMS RECORD EMPOSITION.
00373          01 EMPOSITION.
00374          02 START-DATE-0420.
00375          03 START-YEAR-0420 PIC 9(4).
00376          03 START-MONTH-0420 PIC 9(2).
00377          03 START-DAY-0420 PIC 9(2).
00378          02 FINISH-DATE-0420.
00379          03 FINISH-YEAR-0420 PIC 9(4).
00380          03 FINISH-MONTH-0420 PIC 9(2).
00381          03 FINISH-DAY-0420 PIC 9(2).
00382          02 SALARY-GRADE-0420 PIC 9(2).
00383          02 SALARY-AMOUNT-0420 PIC S9(7)V99 COMP-3.
00384          02 BONUS-PERCENT-0420 PIC SV999 COMP-3.
00385          02 COMMISSION-PERCENT-0420 PIC SV999 COMP-3.
00386          02 OVERTIME-RATE-0420 PIC S9V99 COMP-3.
00387          02 FILLER PIC XXX.
00388
00389          *01 COPY IDMS RECORD OFFICE.
00390          01 OFFICE.
00391          02 OFFICE-CODE-0450 PIC X(3).
00392          02 OFFICE-ADDRESS-0450.
00393          03 OFFICE-STREET-0450 PIC X(20).
00394          03 OFFICE-CITY-0450 PIC X(15).
00395          03 OFFICE-STATE-0450 PIC X(2).
00396          03 OFFICE-ZIP-0450.
00397          04 OFFICE-ZIP-FIRST-FIVE-0450
00398          PIC X(5).
00400 02 OFFICE-PHONE-0450 PIC 9(7) OCCURS 3.
00401 02 OFFICE-AREA-CODE-0450 PIC X(3).
00402 02 SPEED-DIAL-0450 PIC X(3).
00403 02 FILLER PIC X(4).
00404 PROCEDURE DIVISION.
00405
00406 * *********************************************************
00407 * PROCEDURE DIVISION GENERAL STRATEGY: *
00408 * 1) READ DEPT-ID-IN, WHICH CONTAINS THE *
00409 * DEPT-ID NUMBER *
00410 * 2) ACCESS THE DATABASE USING THE DEPT-ID NUMBER *
00411 * 3) ACCESS ALL EMPLOYEES IN THE DEPT-EMPLOYEE SET *
00412 * AND RETRIEVE RELATED JOB AND OFFICE DATA *
00413 * 4) PRINT A REPORT FOR EACH DEPARTMENT *
00414 * 5) PRINT AN ERROR REPORT FOR EMPTY DEPARTMENTS *
00415 * AND NONEXISTENT DEPARTMENTS (NO MATCHING *
00416 * DEPT-ID) *
00417 * *********************************************************
00418
00419 MAIN-LINE.
00420 PERFORM INIT-FILES.
00421 IF END-OF-FILE
00422 PERFORM EMPTY-INPUT-PROCESSING
00423 ELSE
00424 PERFORM INIT-BIND-READY
00425 PERFORM U220-ERR-HEADER
00426 PERFORM DEPT-PROCESSING THRU DEPT-PROCESSING-EXIT
00427 UNTIL END-OF-FILE.
00428 PERFORM END-PROCESSING.
00429 GOBACK.
00430
00431 INIT-BIND-READY.
00432 ******************************************************
00433 * THE BIND STATEMENTS ARE PERFORMED INDIVIDUALLY (RATHER *
00434 * THAN BY USING A COPY IDMS SUBSCHEMA-BINDS) IN ORDER TO *
00435 * CHECK EACH ERROR-STATUS BY PERFORMING THE IDMS-STATUS *
00436 * ROUTINE. *
00437 ******************************************************
00438 MOVE 'DEPTRPT' TO PROGRAM-NAME.
00439
00440 * BIND RUN-UNIT. DMLC0001
00441 MOVE 1 TO DML-SEQUENCE
00442 CALL 'IDMS' USING SUBSCHEMA-CTRL
00443 IDBMSCOM (59)
00444 SUBSCHEMA-CTRL
00445 SUBSCHEMA-SSNAME.
00446 PERFORM IDMS-STATUS.
00447
00448 * BIND EMPLOYEE. DMLC0002
00449 MOVE 2 TO DML-SEQUENCE
00450 CALL 'IDMS' USING SUBSCHEMA-CTRL
00451 IDBMSCOM (48)
00452 SR415
00453 EMPLOYEE.
00454 PERFORM IDMS-STATUS.
00455
00456 * BIND DEPARTMENT. DMLC0003
00457 MOVE 3 TO DML-SEQUENCE
00458 CALL 'IDMS' USING SUBSCHEMA-CTRL
00459 IDBMSCOM (48)
00460 SR410
00461 DEPARTMENT.
00462 PERFORM IDMS-STATUS.
00463
00464 * BIND JOB. DMLC0004
00465 MOVE 4 TO DML-SEQUENCE
00466 CALL 'IDMS' USING SUBSCHEMA-CTRL
00467 IDBMSCOM (48)
00468 SR440
00469 JOB.
00470 PERFORM IDMS-STATUS.
00470   * BIND EMPOSITION.                  DMLC0005
00471      MOVE 5 TO DML-SEQUENCE
00472      CALL 'IDMS' USING SUBSCHEMA-CTRL
00473          IDBMSCOM (48)
00474      SR420
00475      EMPOSITION.
00476      PERFORM IDMS-STATUS.
00477   * BIND OFFICE.                      DMLC0006
00478      MOVE 6 TO DML-SEQUENCE
00479      CALL 'IDMS' USING SUBSCHEMA-CTRL
00480          IDBMSCOM (48)
00481      SR450
00482      OFFICE.
00483      PERFORM IDMS-STATUS.
00484   * READY.                              DMLC0007
00485      MOVE 7 TO DML-SEQUENCE
00486      CALL 'IDMS' USING SUBSCHEMA-CTRL
00487          IDBMSCOM (37).
00488      PERFORM IDMS-STATUS.
00489
00490 INIT-FILES.
00491 OPER OPEN INPUT DEPT-FILE-IN.
00492 OPEN OUTPUT DEPT-FILE-OUT.
00493 OPEN OUTPUT ERR-FILE-OUT.
00494 MOVE SPACES TO PRINT-LINE.
00495 MOVE SPACES TO ERR-LINE.
00496 READ DEPT-FILE-IN AT END MOVE 'Y' TO EOF-SW.
00497
00498 EMPTY-INPUT-PROCESSING.
00499 MOVE NO-INPUT-MESSAGE TO PRINT-LINE.
00500 MOVE '1' TO CC.
00501 PERFORM U000-WRITE-LINE.
00502
00503 ***************************************************************
00504 * THIS PARAGRAPH ACCESSES THE DATABASE USING THE DEPT-ID-0415 *
00505 * CALCKEY VALUE.                                            *
00506 ***************************************************************
00507 DEPT-PROCESSING.
00508      MOVE DEPT-ID-IN TO DEPT-ID-0410.
00509   * OBTAIN CALC DEPARTMENT.                                DMLC0008
00510      MOVE 8 TO DML-SEQUENCE
00511      CALL 'IDMS' USING SUBSCHEMA-CTRL
00512          IDBMSCOM (32)
00513      SR410
00514          IDBMSCOM (43).
00515      IF DB-REC-NOT-FOUND THEN
00516            PERFORM NO-DEPT-PROCESSING
00517       ELSE
00518            PERFORM IDMS-STATUS
00519   * IF DEPT-EMPLOYEE IS NOT EMPTY                         DMLC0009
00520       MOVE 9 TO DML-SEQUENCE
00521       CALL 'IDMS' USING SUBSCHEMA-CTRL
00522          IDBMSCOM (65)
00523            DEPT-EMPLOYEE;
00524       IF ERROR-STATUS EQUAL TO '1601'
00525         THEN
00526            PERFORM U020-VALID-HEADER
00527       MOVE DEPT-ID-0410 TO DEPT-ID-OUT
00528       MOVE DEPT-NAME-0410 TO DEPT-NAME-OUT
00529       MOVE DEPT-DETAIL-LINE TO PRINT-LINE
00530       PERFORM U000-WRITE-LINE
00531       PERFORM U030-EMP-HEADERS
00532       PERFORM SET-WALK THRU SET-WALK-EXIT
00533            UNTIL DB-END-OF-SET
00534       ELSE
00535            PERFORM EMPTY-SET.
00536       READ DEPT-FILE-IN AT END MOVE 'Y' TO EOF-SW.
00537       DEPT-PROCESSING-EXIT.
00538       EXIT.
00539
00540 ***************************************************************
00541 * THIS PARAGRAPH RETRIEVES EMPLOYEE, JOB, AND OFFICE DATA *
00542 * FOR EACH EMPLOYEE IN THE DEPT-EMPLOYEE SET.       *
00543 ************************************************************************
00544 SET-WALK.
00545 * OBTAIN NEXT EMPLOYEE WITHIN DEPT-EMPLOYEE.       DMLC0010
00546   MOVE 10 TO DML-SEQUENCE
00547   CALL 'IDMS' USING SUBSCHEMA-CTRL
00548     IDBMSCOM (10)
00549     SR415
00550     DEPT-EMPLOYEE
00551     IDBMSCOM (43).
00552   IF DB-END-OF-SET
00553   GO TO SET-WALK-EXIT
00554 ELSE
00555     PERFORM IDMS-STATUS.
00556     MOVE EMP-ID-0415 TO ID-OUT.
00557     MOVE EMP-LAST-NAME-0415 TO LAST-OUT.
00558     MOVE EMP-FIRST-NAME-0415 TO FIRST-OUT.
00559     MOVE START-YEAR-0415 TO SD-YY.
00560     MOVE START-MONTH-0415 TO SD-MM.
00561     MOVE START-DAY-0415 TO SD-DD.
00562   * IF EMP-EMPOSITION IS EMPTY       DMLC0011
00563     MOVE 11 TO DML-SEQUENCE
00564     CALL 'IDMS' USING SUBSCHEMA-CTRL
00565       IDBMSCOM (64)
00566       EMP-EMPOSITION;
00567     IF ERROR-STATUS EQUAL TO '0000'
00568       MOVE NO-JOB-MESSAGE TO TITLE-OUT
00569 ELSE
00570       * FIND FIRST WITHIN EMP-EMPOSITION DMLC0012
00571       MOVE 12 TO DML-SEQUENCE
00572       CALL 'IDMS' USING SUBSCHEMA-CTRL
00573       IDBMSCOM (20)
00574       EMP-EMPOSITION;
00575       PERFORM IDMS-STATUS
00576   * IF NOT JOB-EMPOSITION MEMBER DMLC0013
00577     MOVE 13 TO DML-SEQUENCE
00578     CALL 'IDMS' USING SUBSCHEMA-CTRL
00579     IDBMSCOM (62)
00580     JOB-EMPOSITION;
00581     IF ERROR-STATUS EQUAL TO '1601'
00582       MOVE NO-JOB-MESSAGE TO TITLE-OUT
00583 ELSE
00584       * OBTAIN OWNER WITHIN JOB-EMPOSITION DMLC0014
00585     MOVE 14 TO DML-SEQUENCE
00586     CALL 'IDMS' USING SUBSCHEMA-CTRL
00587     IDBMSCOM (31)
00588     JOB-EMPOSITION
00589     IDBMSCOM (43);
00590     PERFORM IDMS-STATUS
00591     MOVE TITLE-0440 TO TITLE-OUT.
00592   * IF OFFICE-EMPLOYEE IS EMPTY       DMLC0015
00593     MOVE 15 TO DML-SEQUENCE
00594     CALL 'IDMS' USING SUBSCHEMA-CTRL
00595     IDBMSCOM (64)
00596     OFFICE-EMPLOYEE;
00597     IF ERROR-STATUS EQUAL TO '0000'
00598       MOVE NO-OFFICE-MESSAGE TO STREET-OUT
00599     MOVE SPACES TO CITY-OUT
00600     MOVE SPACES TO STATE-OUT
00601     MOVE SPACES TO ZIP-OUT
00602 ELSE
00603       * OBTAIN OWNER WITHIN OFFICE-EMPLOYEE DMLC0016
00604     MOVE 16 TO DML-SEQUENCE
00605     CALL 'IDMS' USING SUBSCHEMA-CTRL
00606     IDBMSCOM (31)
00607     OFFICE-EMPLOYEE
00608     IDBMSCOM (43);
00609     PERFORM IDMS-STATUS
00610     MOVE OFFICE-STREET-0450 TO STREET-OUT
00611     MOVE OFFICE-CITY-0450 TO CITY-OUT
MOVE OFFICE-STATE-0450 TO STATE-OUT
MOVE OFFICE-ZIP-FIRST-FIVE-0450 TO ZIP-OUT
MOVE EMP-Detail-Line TO PRINT-LINE.
PERFORM U000-WRITE-LINE.
SET-WALK-EXIT.
EXIT.
END-PROCESSING.

MOVE 17 TO DML-SEQUENCE
CALL 'IDMS' USING SUBSCHEMA-CTRL
IDBMSCOM (2).
PERFORM IDMS-STATUS.
CLOSE DEPT-FILE-OUT.
CLOSE ERR-FILE-OUT.
CLOSE DEPT-FILE-IN.

EMPTY-SET.
MOVE SPACES TO ERR-LINE.
MOVE DEPT-ID-0410 TO ERR-ID-OUT.
MOVE NO-EMP-MESSAGE TO ERR-MESS-OUT.
MOVE ERR-DETAIL-LINE TO ERR-LINE.
PERFORM U200-WRITE-ERR-LINE.

NO-DEPT-PROCESSING.
MOVE DEPT-ID-IN TO ERR-ID-OUT.
MOVE NO-DEPT-MESSAGE TO ERR-MESS-OUT.
MOVE ERR-DETAIL-LINE TO ERR-LINE.
PERFORM U200-WRITE-ERR-LINE.

U000-WRITE-LINE.
WRITE DEPT-REC-OUT AFTER POSITIONING CC.
IF CC = '1' THEN MOVE 0 TO LINE-COUNT
ELSE IF CC = ' ' THEN ADD 1 TO LINE-COUNT
ELSE IF CC = '0' THEN ADD 2 TO LINE-COUNT.
IF LINE-COUNT > LINE-MAX
THEN PERFORM U010-NEW-PAGE-Routine.

U010-NEW-PAGE-Routine.
PERFORM U020-VALID-HEADER.
MOVE DEPT-DETAIL-LINE TO PRINT-LINE.
PERFORM U000-WRITE-LINE.
PERFORM U030-EMP-HEADERS.

U020-VALID-HEADER.
MOVE DEPT-HEADER TO PRINT-LINE.
MOVE '1' TO CC.
PERFORM U000-WRITE-LINE
MOVE ' ' TO CC.

U030-EMP-HEADERS.
MOVE '0' TO CC.
MOVE EMP-HEADER TO PRINT-LINE.
PERFORM U000-WRITE-LINE.
MOVE SPACES TO PRINT-LINE.
MOVE ' ' TO CC.
PERFORM U000-WRITE-LINE.

U200-WRITE-ERR-LINE.
WRITE ERR-REC-OUT AFTER POSITIONING ERR-CC.
IF ERR-CC = '1' THEN MOVE 0 TO ERR-LINE-COUNT
ELSE IF ERR-CC = ' ' THEN ADD 1 TO ERR-LINE-COUNT
ELSE IF ERR-CC = '0' THEN ADD 2 TO ERR-LINE-COUNT.
IF ERR-LINE-COUNT > LINE-MAX THEN
PERFORM U220-ERR-HEADER.

U220-ERR-HEADER.
MOVE ERR-HEADER-1 TO ERR-LINE.
MOVE '1' TO ERR-CC.
PERFORM U200-WRITE-ERR-LINE
MOVE '0' TO ERR-CC.
MOVE ERR-HEADER-2 TO ERR-LINE.
PERFORM U200-WRITE-ERR-LINE.
MOVE ERR-HEADER-3 TO ERR-LINE.
PERFORM U200-WRITE-ERR-LINE.
00683   MOVE SPACES TO ERR-LINE.
00684   MOVE ' ' TO ERR-CC.
00685   PERFORM U200-WRITE-ERR-LINE.
00686   IDMS-ABORT.
00687   EXIT.
00688   IDMS-ABORT-EXIT.
00689   * COPY IDMS IDMS-STATUS.
00690   ******************************************************************
00691   IDMS-STATUS SECTION.
00692   ******************************************************************
00693   IDMS-STATUS-PARAGRAPH.
00694      IF DB-STATUS-OK GO TO ISABEX.
00695      PERFORM IDMS-ABORT.
00696      DISPLAY '**************************'
00697         ' ABORTING - ' PROGRAM-NAME
00698         ', ' ERROR-STATUS
00699         ', ' ERROR-RECORD
00700         ' **** RECOVER IDMS ****'
00701      UPON CONSOLE.
00702      DISPLAY 'PROGRAM NAME ------ ' PROGRAM-NAME.
00703      DISPLAY 'ERROR STATUS ------ ' ERROR-STATUS.
00704      DISPLAY 'ERROR RECORD ------ ' ERROR-RECORD.
00705      DISPLAY 'ERROR SET ------ ' ERROR-SET.
00706      DISPLAY 'ERROR AREA ------ ' ERROR-AREA.
00707      DISPLAY 'LAST GOOD RECORD ------ ' RECORD-NAME.
00708      DISPLAY 'LAST GOOD AREA ---- ' AREA-NAME.
00709      MOVE 39 TO SSC-IN01-REQ-CODE.
00710      MOVE 0 TO SSC-IN01-REQ-RETURN.
00711      MOVE ' ' TO SSC-STATUS-LABEL.
00712      PERFORM IDMS-STATUS-LOOP
00713      UNTIL SSC-IN01-REQ-RETURN > 0.
00714      * ROLLBACK. DMLC0018
00715      MOVE 18 TO DML-SEQUENCE
00716      CALL 'IDMS' USING SUBSCHEMA-CTRL
00717          IDBMSCOM (67).
00718      CALL 'ABORT'.
00719      GO TO ISABEX.
00720      IDMS-STATUS-LOOP.
00721      CALL 'IDMSP1' USING IDBMSCOM(41)
00722      SSC-IN01-REQ-WK
00723      SUBSCHEMA-CTRL
00724      IDBMSCOM(1)
00725      DML-SEQUENCE
00726      SSC-STATUS-LINE.
00727      IF SSC-IN01-REQ-RETURN GREATER THAN 4
00728      DISPLAY 'DML SEQUENCE ------ ' DML-SEQUENCE
00729      ELSE
00730      DISPLAY SSC-STATUS-LABEL '--- ' SSC-STATUS-VALUE.
00731      ISABEX. EXIT.
Sample Online Program

This section contains a sample CA IDMS online application that illustrates the structure of CA IDMS programs that accept data from a terminal operator and retrieve information from the database. The application program highlights the following CA IDMS features:

- Mapping mode input and output
- Automatic editing and error handling
- Pseudo-conversational transactions
- LRF DML statements

The application's components, runtime requirements, and DML code are described below.

This article describes the following information:

- Application Components (see page 328)
- Application Runtime Requirements (see page 329)

Application Components

The application comprises a program, two tasks, a map, and a subschema:

- **Program** -- The EMPDISP program either performs a MAP OUT to start a session or performs a MAP IN, error checking, database access, and a MAP OUT.

- **Tasks** -- The task codes TSK01 and TSK02 affect the program flow of control:
  - **TSK01** causes the program to perform the INITIAL-MAPOUT portion of the program, mapping out the empty screen with an initial input message.
  - **TSK02** causes the program to perform the GET-EMP portion of the program, mapping in the data, checking the AID byte, performing the error checking and database access portion of the program, and mapping out either an error message or employee data.

- **Map** -- The application uses a map named EMPMAPLR to communicate with the terminal operator. The EMPMAPLR map is illustrated below. Its map definition specifies:
  - Eight literal fields including the title *** EMPLOYEE INFORMATION SCREEN ***.
  - Ten variable data fields, to contain: employee ID, last name, first name, job title, start date, department name, and office address (street, city, state, and zip code). All data is contained in the EMP-JOB-LR logical record.
  - Automatic editing for the employee ID field specifies that the field is in error if the ID entered by the terminal operator does not comply with the field's external picture (PIC 9(4)).
  - Messages are output in the $MESSAGE field.

- **Subschema** -- The application uses the EMPSS09 subschema, which specifies a usage mode of LR. The program uses LRF DML statements to retrieve the EMP-JOB-LR logical record.
Application Runtime Requirements

The following requirements must be met to execute the sample online application under CA IDMS:

- Define and generate the EMPMAPLR map.
- Compile and link edit the EMPDISP program into a load library that is identified to CA IDMS.
- Define the EMPDISP program to the CA IDMS system either by submitting PROGRAM statements to the system generation compiler or by using the DCMT VARY DYNAMIC PROGRAM command at runtime.
- Define the EMPMAPLR map and the EMPSS09 subschema to the CA IDMS system by submitting PROGRAM statements to the system generation compiler. Maps and subschemas are defined automatically at system startup if null program definition elements (PDEs) have been allocated for them at system generation.

Sample Online COBOL Program as Input to the DML Precompiler

```
*NO-ACTIVITY-LOG
*DMLIST
IDENTIFICATION DIVISION.

PROGRAM-ID. EMPDISP.
AUTHOR. COMPUTER ASSOCIATES INTERNATIONAL.
DATE-WRITTEN. APRIL 1995.
REMARKS. THIS PROGRAM DEMONSTRATES
CA IDMS PROGRAMMING USING
THE LOGICAL RECORD FACILITY.

********************************************************************
ENVIRONMENT DIVISION.
********************************************************************
```

*** EMPLOYEE INFORMATION SCREEN ***

Employee ID:
Last Name :
First Name:

Job Title:
Start Date:

Department Name:
Office:

** Clear to exit
IDMS-CONTROL SECTION.

PROTOCOL. MODE IS IDMS-DC DEBUG
IDMS-RECORDS MANUAL.
SKIP3
DATA DIVISION.

SCHEMA SECTION.

DB EMPSS09 WITHIN EMPSCHM.

MAP SECTION.
MAX FIELD LIST IS 5.
MAP EMPMAPLR VERSION 1 TYPE IS STANDARD.

WORKING-STOREAGE SECTION.
01 TASK-CODE PIC X(8).
01 TSK01 PIC X(8) VALUE 'TSK01'.
01 TSK02 PIC X(8) VALUE 'TSK02'.

01 MESSAGES.
05 INITIAL-MESSAGE PIC X(80) VALUE
  'ENTER AN EMP ID AND PRESS ENTER ** CLEAR TO EXIT'.
05 EDIT-ERROR-MESSAGE PIC X(80) VALUE
  'EMP-ID EITHER NOT ENTERED OR NOT NUMERIC'.
05 EMP-NOT-FOUND-MESSAGE PIC X(80) VALUE
  'SPECIFIED EMPLOYEE COULD NOT BE FOUND'.
05 DISPLAY-MESSAGE PIC X(80) VALUE
  'CLEAR TO EXIT ** NEW EMP-ID AND ENTER TO CONTINUE'.

01 COPY IDMS DC-AID-CONDITION-NAMES.
01 COPY IDMS EMP-DATE-WORK-REC.
01 COPY IDMS SUBSCHEMA-LR-CONTROL.
01 COPY IDMS SUBSCHEMA-LR-RECORDS.
03 SUBSCHEMA-LR-CTRL-END PIC X.

01 COPY IDMS MAP-CONTROLS.

EJECT

PROCEDURE DIVISION.

* *************************************************************
*  * PROEDURE DIVISION GENERAL STRATEGY:*
*  * RETRIEVE INFORMATION FOR A SPECIFIED EMPLOYEE. *
*  * DISPLAYED DATA INCLUDES EMPLOYEE, DEPARTMENT, *
*  * JOB, AND OFFICE INFORMATION. *
*  * ==> THIS PROGRAM USES THE EMP-JOB-LR LOGICAL RECORD<= *
*  * PROGRAM STRATEGY: *
*  *   ** CHECK FOR TASK CODE: TSK01= INITIAL MAPOUT *
*  *   ANYTHING ELSE = RETRIEVE LR *
*  *   ** CLEAR TO EXIT APPLICATION *
*  *   ** ENTER AND NEW EMP-ID TO CONTINUE *
*  *************************************************************

MAIN-LINE.

* THE BIND MAP STATEMENTS ADVISE IDMS-DC OF THE LOCATION OF *
* THE MRB AND THE MAP RECORDS. *

BIND MAP EMPMAPLR.
BIND MAP EMPMAPLR RECORD EMPLOYEE.
BIND MAP EMPMAPLR RECORD DEPARTMENT.
BIND MAP EMPMAPLR RECORD JOB.
BIND MAP EMPMAPLR RECORD OFFICE.
BIND MAP EMPMAPLR RECORD EMP-DATE-WORK-REC.
* ACCEPT TASK CODE INTO TASK-CODE.
  IF TASK-CODE = TSK01
    GO TO INITIAL-MAPOUT
  ELSE
    GO TO GET-EMP.
*THE INITIAL-MAPOUT PARAGRAPH IS PERFORMED IF THE CALLING TASK CODE IS TSK01.*
*THE MODIFY MAP STATEMENT Assigns THE PROTECTED Attribute TO ALL MAP FIELDS EXCEPT EMP-ID-0415.*
*THE MAP OUT STATEMENT Transmits THE EMPMAPLR MAP TO THE TERMINAL.*
*THE DC RETURN STATEMENT SPECIFIES THAT THE NEXT TASK THAT WILL BE INITIATED ON THE SAME TERMINAL WHEN THE OPERATOR PRESSES A CONTROL KEY WILL BE TSK02.*

INITIAL-MAPOUT.
MODIFY MAP EMPMAPLR TEMPORARY
FOR ALL EXCEPT EMP-ID-0415
ATTRIBUTES PROTECTED.
*MOVE ZERO TO EMP-ID-0415.
MAP OUT USING EMPMAPLR
OUTPUT DATA IS YES NEWPAGE
MESSAGE IS INITIAL-MESSAGE LENGTH 80.

DC RETURN
NEXT TASK CODE TSK02.
INITIAL-MAPOUT-EXIT.
EXIT.

*THE GET-EMP PARAGRAPH IS PERFORMED IF THE CALLING TASK CODE IS NOT TSK01.*
*THE MAP IN STATEMENT Transmits DATA FROM THE TERMINAL TO VARIABLE STORAGE DATA FIELDS.*
*THE FIRST INQUIRE MAP STATEMENT IS USED TO DETERMINE THE AID KEY Pressed.*
*THE SECOND INQUIRE MAP STATEMENT USES AUTOMATIC EDITING TO DETERMINE IF THE DATA ENTERED IS CONSISTENT WITH THE EXTERNAL PICTURE OF THE NAMED DATA ELEMENT.*
*THE MAP OUT STATEMENT Transmits DATA FROM THE EMP-JOB-LR LOGICAL RECORD IN VARIABLE STORAGE TO MAP FIELDS.*

GET-EMP.
MAP IN USING EMPMAPLR.
*INQUIRE MAP EMPMAPLR
MOVE AID TO DC-AID-IND-V.
IF CLEAR-HIT
DC RETURN.
*
INQUIRE MAP EMPMAPLR
IF DFLD EMP-ID-0415 EDIT IS ERROR
THEN GO TO EDIT-ERROR.
*
COPY IDMS SUBSCHEMA-BINDS.
READY USAGE-MODE IS RETRIEVAL.
*SINCE THE MAP FIELD IS ASSOCIATED WITH THE EMP-ID-0415 FIELD, THE PROGRAM USES THE "OF LR" RETRIEVAL. NOTE THAT AUTOSTATUS IMPLICITLY CHECKS FOR THE LR-ERROR PATH STATUS.*

OBTAIN EMP-JOB-LR
WHERE EMP-ID-0415 = EMP-ID-0415 OF LR
ON LR-NOT-FOUND
GO TO NOT-FOUND.
FINISH.
**************************************************************
* REFORMAT DATE TO MMDDYY; OUTPUT AS MM/DD/YY USING THE OLM *
* EXTERNAL PICTURE SPECIFICATION (XX/XX/XX). *
**************************************************************
MOVE START-YEAR-0415 TO WORK-YY.
MOVE START-MONTH-0415 TO WORK-MM.
MOVE START-DAY-0415 TO WORK-DD.

MAP OUT USING EMPMAPLR
OUTPUT DATA IS YES
MESSAGE IS DISPLAY-MESSAGE LENGTH 80.
*
DC RETURN NEXT TASK CODE TSK02.
GET-EMP-EXIT.
EXIT.
**************************************************************
* THE MODIFY MAP STATEMENT SPECIFIES THAT ALL MAP *
* FIELDS EXCEPT THE INCORRECT EMP-ID-0415 FIELD WILL BE *
* ERASED ON THE NEXT MAP OUT. *
**************************************************************
EDIT-ERROR.
MODIFY MAP EMPMAPLR TEMPORARY
FOR ALL EXCEPT DFLD EMP-ID-0415
OUTPUT DATA IS ERASE.
*
MAP OUT USING EMPMAPLR
MESSAGE IS EDIT-ERROR-MESSAGE LENGTH 80.
*
DC RETURN
NEXT TASK CODE TSK02.
EDIT-ERROR-EXIT.
EXIT.
**************************************************************
* THE FOLLOWING MODIFY MAP STATEMENT SPECIFIES THAT ALL *
* MAP FIELDS EXCEPT THE EMP-ID-0415 FIELD WILL BE ERASED *
* ON THE NEXT MAP OUT. *
**************************************************************
NOT-FOUND.
MODIFY MAP EMPMAPLR TEMPORARY
FOR ALL EXCEPT DFLD EMP-ID-0415
OUTPUT DATA IS ERASE.
*
MAP OUT USING EMPMAPLR
MESSAGE IS EMP-NOT-FOUND-MESSAGE LENGTH 80.
*
DC RETURN
NEXT TASK CODE TSK02.
NOT-FOUND-EXIT.
EXIT.
**************************************************************
IDMS-ABORT.
MOVE ERROR-STATUS TO SSC-ERRSTAT-SAVE.
MOVE DML-SEQUENCE TO SSC-DMLSEQ-SAVE.
SNAP FROM SUBSHEMA-LR-CTRL TO SUBSHEMA-LR-CTRL-END
ON ANY-STATUS NEXT SENTENCE.
MOVE SSC-ERRSTAT-SAVE TO ERROR-STATUS.
MOVE SSC-DMLSEQ-SAVE TO DML-SEQUENCE.
IDMS-ABORT-EXIT.
EXIT.
COPY IDMS IDMS-STATUS.
Sample Online COBOL Program as Output from the DML Precompiler

00002 *DMLIST
00003
00004 IDENTIFICATION DIVISION.
00005
00006 PROGRAM-ID. EMPDISP.
00007
00008 AUTHOR. COMPUTER ASSOCIATES INTERNATIONAL.
00009
00010 DATE-WRITTEN. APRIL 1995.
00011
00012 REMARKS. THIS PROGRAM DEMONSTRATES
00013 CA IDMS PROGRAMMING USING
00014 THE LOGICAL RECORD FACILITY.
00015
00016 **************************************************************
00017 ENVIRONMENT DIVISION.
00018 **************************************************************
DMLC 00019 IDMS-CONTROL SECTION.
DMLC 00020
DMLC 00021 PROTOCOL. MODE IS IDMS-DC DEBUG
DMLC 00022 IDMS-RECORDS MANUAL.
00023 SKIP3
DMLC 00024 DATA DIVISION.
00025
DMLC 00026 SCHEMA SECTION.
00027
DMLC 00028 DB EMPSS09 WITHIN EMPSCHM.
00029
DMLC 00030 MAP SECTION.
DMLC 00031 MAX FIELD LIST IS 5.
DMLC 00032 MAP EMMPMAPLR VERSION 1 TYPE IS STANDARD.
00033
00034
00035 WORKING-STORAGE SECTION.
00036
00037 01 TASK-CODE PIC X(8).
00038 01 TSK01 PIC X(8) VALUE 'TSK01'.
00039 01 TSK02 PIC X(8) VALUE 'TSK02'.
00040
00041 01 MESSAGES.
00042 05 INITIAL-MESSAGE PIC X(80) VALUE
00043 'ENTER AN EMP ID AND PRESS ENTER ** CLEAR TO EXIT'.
00044 05 EDIT-ERROR-MESSAGE PIC X(80) VALUE
00045 'EMP-ID EITHER NOT ENTERED OR NOT NUMERIC'.
00046 05 EMP-NOT-FOUND-MESSAGE PIC X(80) VALUE
00047 'SPECIFIED EMPLOYEE COULD NOT BE FOUND'.
00048 05 DISPLAY-MESSAGE PIC X(80) VALUE
00049 'CLEAR TO EXIT ** NEW EMP-ID AND ENTER TO CONTINUE'.
00050
DMLC 00051 01 COPY IDMS DC-AID-CONDITION-NAMES.
00052 01 DC-AID-CONDITION-NAMES.
00053 03 DC-AID-IND-V PIC X.
00054 88 ENTER-HIT VALUE QUOTE.
00055 88 CLEAR-HIT VALUE '.
00056 88 PF01-HIT VALUE '1'.
00057 88 PF02-HIT VALUE '2'.
00058 88 PF03-HIT VALUE '3'.
00059 88 PF04-HIT VALUE '4'.
00060 88 PF05-HIT VALUE '5'.
00061 88 PF06-HIT VALUE '6'.
00062 88 PF07-HIT VALUE '7'.
00063 88 PF08-HIT VALUE '8'.
00064 88 PF09-HIT VALUE '9'.

00065 88 PF10-HIT VALUE ':'.
00066 88 PF11-HIT VALUE '#'.
00067 88 PF12-HIT VALUE '@'.
00068 88 PF13-HIT VALUE 'A'.
00069 88 PF14-HIT VALUE 'B'.
00070 88 PF15-HIT VALUE 'C'.
00071 88 PF16-HIT VALUE 'D'.
00072 88 PF17-HIT VALUE 'E'.
00073 88 PF18-HIT VALUE 'F'.
00074 88 PF19-HIT VALUE 'G'.
00075 88 PF20-HIT VALUE 'H'.
00076 88 PF21-HIT VALUE 'I'.
00077 88 PF22-HIT VALUE '.'.
00078 88 PF23-HIT VALUE '_'.
00079 88 PF24-HIT VALUE '<'.
00080 88 PA01-HIT VALUE '%'.
00081 88 PA02-HIT VALUE '>'.
00082 88 PA03-HIT VALUE ','.
00083 88 PEN-ATTN-SPACE-NUL VALUE '='.
00084 88 PEN-ATTN VALUE QUOTE.
00085
DMLC 00086 01 COPY IDMS EMP-DATE-WORK-REC.
00087 01 EMP-DATE-WORK-REC.
00088 02 WORK-DATE.
00089 03 WORK-MM PIC 9(2).
00090 03 WORK-DD PIC 9(2).
00091 03 WORK-YY PIC 9(2).
00092
DMLC 00093 01 COPY IDMS SUBSCHEMA-LR-CONTROL.
00094 01 SUBSCHEMA-CTRL.
00095 03 PROGRAM-NAME PIC X(8) VALUE SPACES.
00096 03 ERROR-STATUS PIC X(4) VALUE '1400'.
00097 88 DB-STATUS-OK VALUE '0000'.
00098 88 ANY-STATUS.
00099 VALUE '0000' THRU '9999'.
00100 88 ANY-ERROR-STATUS.
00101 VALUE '0011' THRU '9999'.
00102 88 DB-END-OF-SET VALUE '0307'.
00103 88 DC-DEADLOCK VALUE '3101' '3201' '3401' '3901'.
00104 88 DC-NO-STORAGE VALUE '3202' '3203' '3402'.
00105 88 DC-AREA-ID-UNK VALUE '4303'.
00106 88 ANY-STATUS.
00107 VALUE '0000' THRU '9999'.
00108 88 DC-RESOURCE-NOT-AVAIL 00113.
00109 88 DC-NEW-STORAGE VALUE '3210' '3211' '3212'.
00110 88 DC-REC-REPLACED VALUE '4317'.
00111 88 DC-TRUNCATED-DATA 00120.
00112 VALUE '4319' '4419' '4519'.
00113 88 DC-ATTN-INT VALUE '4525'.
00114 88 DC-OPER-CANCEL VALUE '4743'.
00115 88 DC-FIRST-PAGE-SENT 0.
00116 88 DC-SECOND-STARTPAGE 0.
00117 88 DC-Detail-Not-Found 0.
00118 88 DC-ERROR-STATUS.
00119 VALUE '4676'.
00120 88 DC-REC-REPLACED VALUE '4676'.
00121 88 DC-ATTN-INT VALUE '4625'.
00122 88 DC-OPER-CANCEL VALUE '4743'.
00123 88 DC-ERROR-STATUS.
00124 VALUE '4676'.
00125 88 DC-REC-REPLACED VALUE '4676'.
00126 88 DB-END-OF-SET VALUE '0307'.
00127 88 DC-ATTN-INT VALUE '4625'.
00128 88 DC-OPER-CANCEL VALUE '4743'.
00129 88 DC-ERROR-STATUS.
00130 VALUE '4676'.
00131 88 DBEND-OF-SET VALUE '0307'.
00132 88 DC-ATTN-INT VALUE '4625'.
00133 88 DC-OPER-CANCEL VALUE '4743'.
00134 Usage Comp. 00135.
00136 03 DBEND-OF-SET PIC X.
00137 (16) VALUE SPACES. 00138 03 REDEFINES RECORD-NAME. 00139
00140 05 SSN-NAME PIC X(8). 00141 05 SSN-DUP PIC X(8).
00142 03 ERROR-SET PIC X.
00143 (16) VALUE SPACES. 00144 03 RECORD-NAME PIC X.
00145 03 ERROR-RECORD PIC X.
00146 03 ERROR-AREA PIC X.
00147 03 IDBMS-AREA PIC X.
00148 03 IDBMS-AREA PIC X.
00149 03 IDBMS-AREA PIC X.
00150 03 DBEND-OF-SET PIC X. 00151
00152 03 DBEND-OF-SET PIC X.
00153 03 DBEND-OF-SET PIC X.
00154 03 DBEND-OF-SET PIC X.
00155 03 DBEND-OF-SET PIC X.
00156 03 DBEND-OF-SET PIC X.
00157 03 DBEND-OF-SET PIC X.
00158 03 DBEND-OF-SET PIC X.
00159 03 DBEND-OF-SET PIC X.
00160 03 DBEND-OF-SET PIC X.
00161 03 DBEND-OF-SET PIC X.
00162 03 DBEND-OF-SET PIC X.
00163 03 DBEND-OF-SET PIC X.
CA IDMS Reference - 19.0

00285  03 PXEDSP16  REDEFINES PXE
00286      PIC X(16).
00287  03 PXEDSP8  REDEFINES PXE
00288      PIC X(8).
00289  03 PXECOMP-1  REDEFINES PXE
00290      USAGE COMP-1.
00291  03 PXECOMP-2  REDEFINES PXE
00292      USAGE COMP-2.
00293  03 PXECOMP-30  REDEFINES PXE
00294      PIC S9(18)
00295      USAGE COMP-3.
00296  03 PXECOMP-31  REDEFINES PXE
00297      PIC S9(17)V9(1)
00298      USAGE COMP-3.
00299  03 PXECOMP-32  REDEFINES PXE
00300      PIC S9(16)V9(2)
00301      USAGE COMP-3.
00302  03 PXECOMP-33  REDEFINES PXE
00303      PIC S9(15)V9(3)
00304      USAGE COMP-3.
00305  03 PXECOMP-34  REDEFINES PXE
00306      PIC S9(14)V9(4)
00307      USAGE COMP-3.
00308  03 PXECOMP-35  REDEFINES PXE
00309      PIC S9(13)V9(5)
00310      USAGE COMP-3.
00311  03 PXECOMP-36  REDEFINES PXE
00312      PIC S9(12)V9(6)
00313      USAGE COMP-3.
00314  03 PXECOMP-37  REDEFINES PXE
00315      PIC S9(11)V9(7)
00316      USAGE COMP-3.
00317  03 PXECOMP-38  REDEFINES PXE
00318      PIC S9(10)V9(8)
00319      USAGE COMP-3.
00320  03 PXECOMP-39  REDEFINES PXE
00321      PIC S9(9)V9(9)
00322      USAGE COMP-3.
00323  03 PXECOMP-310  REDEFINES PXE
00324      PIC S9(8)V9(10)
00325      USAGE COMP-3.
00326  03 PXECOMP-311  REDEFINES PXE
00327      PIC S9(7)V9(11)
00328      USAGE COMP-3.
00329  03 PXECOMP-312  REDEFINES PXE
00330      PIC S9(6)V9(12)
00331      USAGE COMP-3.
00332  03 PXECOMP-313  REDEFINES PXE
00333      PIC S9(5)V9(13)
00334      USAGE COMP-3.
00335  03 PXECOMP-314  REDEFINES PXE
00336      PIC S9(4)V9(14)
00337      USAGE COMP-3.
00338  03 PXECOMP-315  REDEFINES PXE
00339      PIC S9(3)V9(15)
00340      USAGE COMP-3.
00341  03 PXECOMP-316  REDEFINES PXE
00342      PIC S9(2)V9(16)
00343      USAGE COMP-3.
00344  03 PXECOMP-317  REDEFINES PXE
00345      PIC S9(1)V9(17)
00346      USAGE COMP-3.
00347  03 PXECOMP-318  REDEFINES PXE
00348      PIC SV9(18)
00349      USAGE COMP-3.
00350  03 PXECOMP20  REDEFINES PXE
00351      PIC S9(4)
00352      USAGE COMP.
00353  03 PXECOMP21  REDEFINES PXE
00354      PIC S9(3)V9(1)
00355      USAGE COMP.
00356 03 PXECOMP22    REDEFINES PXE
00357 PIC S9(2)V9(2)   USAGE COMP.
00358
00359 03 PXECOMP23    REDEFINES PXE
00360 PIC S9(1)V9(3)   USAGE COMP.
00361
00362 03 PXECOMP24    REDEFINES PXE
00363 PIC SV9(4)      USAGE COMP.
00364
00365 03 PXECOMP40    REDEFINES PXE
00366 PIC S9(9)       USAGE COMP.
00367
00368 03 PXECOMP41    REDEFINES PXE
00369 PIC S9(8)V9(1)  USAGE COMP.
00370
00371 03 PXECOMP42    REDEFINES PXE
00372 PIC S9(7)V9(2)  USAGE COMP.
00373
00374 03 PXECOMP43    REDEFINES PXE
00375 PIC S9(6)V9(3)  USAGE COMP.
00376
00377 03 PXECOMP44    REDEFINES PXE
00378 PIC S9(5)V9(4)  USAGE COMP.
00379
00380 03 PXECOMP45    REDEFINES PXE
00381 PIC S9(4)V9(5)  USAGE COMP.
00382
00383 03 PXECOMP46    REDEFINES PXE
00384 PIC S9(3)V9(6)  USAGE COMP.
00385
00386 03 PXECOMP47    REDEFINES PXE
00387 PIC S9(2)V9(7)  USAGE COMP.
00388
00389 03 PXECOMP48    REDEFINES PXE
00390 PIC S9(1)V9(8)  USAGE COMP.
00391
00392 03 PXECOMP49    REDEFINES PXE
00393 PIC SV9(9)      USAGE COMP.
00394
00395 03 PXECOMP80    REDEFINES PXE
00396 PIC S9(18)      USAGE COMP.
00397
00398 03 PXECOMP81    REDEFINES PXE
00399 PIC S9(17)V9(1) USAGE COMP.
00400
00401 03 PXECOMP82    REDEFINES PXE
00402 PIC S9(16)V9(2) USAGE COMP.
00403
00404 03 PXECOMP83    REDEFINES PXE
00405 PIC S9(15)V9(3) USAGE COMP.
00406
00407 03 PXECOMP84    REDEFINES PXE
00408 PIC S9(14)V9(4) USAGE COMP.
00409
00410 03 PXECOMP85    REDEFINES PXE
00411 PIC S9(13)V9(5) USAGE COMP.
00412
00413 03 PXECOMP86    REDEFINES PXE
00414 PIC S9(12)V9(6) USAGE COMP.
00415
00416 03 PXECOMP87    REDEFINES PXE
00417 PIC S9(11)V9(7) USAGE COMP.
00418
00419 03 PXECOMP88    REDEFINES PXE
00420 PIC S9(10)V9(8) USAGE COMP.
00421
00422 03 PXECOMP89    REDEFINES PXE
00423 PIC S9(9)V9(9)  USAGE COMP.
00424
00425 03 PXECOMP810   REDEFINES PXE
00426 PIC S9(8)V9(10)
00427 USAGE COMP.
00428 03 PXECOMP811 REDEFINES PXE
00429 PIC S9(7)V9(11)
00430 USAGE COMP.
00431 03 PXECOMP812 REDEFINES PXE
00432 PIC S9(6)V9(12)
00433 USAGE COMP.
00434 03 PXECOMP813 REDEFINES PXE
00435 PIC S9(5)V9(13)
00436 USAGE COMP.
00437 03 PXECOMP814 REDEFINES PXE
00438 PIC S9(4)V9(14)
00439 USAGE COMP.
00440 03 PXECOMP815 REDEFINES PXE
00441 PIC S9(3)V9(15)
00442 USAGE COMP.
00443 03 PXECOMP816 REDEFINES PXE
00444 PIC S9(2)V9(16)
00445 USAGE COMP.
00446 03 PXECOMP817 REDEFINES PXE
00447 PIC S9(1)V9(17)
00448 USAGE COMP.
00449 03 PXECOMP818 REDEFINES PXE
00450 PIC SV9(18)
00451 01 SUBSCHEMA-SSNAME PIC X(8) VALUE 'EMPSS09'.
00452 01 SUBSCHEMA-AREANAMES.
00453 03 EMP-DEMO-REGION PIC X(16)
00454 VALUE 'EMP-DEMO-REGION'.
00455 03 INS-DEMO-REGION PIC X(16)
00456 VALUE 'INS-DEMO-REGION'.
00457 03 ORG-DEMO-REGION PIC X(16)
00458 VALUE 'ORG-DEMO-REGION'.
00459 DMLC 00460 01 COPY IDMS SUBSCHEMA-LR-RECORDS.
00461 01 EMP-JOB-LR.
00462 02 EMPLOYEE.
00463 03 EMP-ID-0415 PIC 9(4).
00464 03 EMP-NAME-0415.
00465 03 EMP-NUMBER-0415.
00466 04 EMP-FIRST-NAME-0415 PIC X(10).
00467 04 EMP-LAST-NAME-0415 PIC X(15).
00468 03 STATUS-0415 PIC X(2).
00469 88 ACTIVE-0415 VALUE '01'.
00470 88 ST-DISABIL-0415 VALUE '02'.
00471 88 LT-DISABIL-0415 VALUE '03'.
00472 88 LEAVE-OF-ABSENCE-0415 VALUE '04'.
00473 88 TERMINATED-0415 VALUE '05'.
00474 03 SS-NUMBER-0415 PIC 9(9).
00475 03 SS-DATE-0415 PIC 9(2).
00476 03 START-DATE-0415 PIC 9(2).
00477 04 START-YEAR-0415 PIC 9(2).
00478 04 START-MONTH-0415 PIC 9(2).
00479 04 START-DAY-0415 PIC 9(2).
00480 03 FILLER PIC X(2).
00481 02 DEPARTMENT.
00482 03 DEPT-ID-0410 PIC 9(4).
00483 03 DEPT-NAME-0410 PIC X(45).
00484 03 DEPT-HEAD-ID-0410 PIC 9(4).
00485 03 FILLER PIC XXX.
00486 02 JOB.
00487 03 JOB-ID-0440 PIC 9(4).
00488 03 TITLE-0440 PIC X(20).
00489 02 OFFICE.
00490 03 OFFICE-CODE-0450 PIC X(3).
00491 03 OFFICE-ADDRESS-0450.
00492 04 OFFICE-STREET-0450 PIC X(20).
00493 04 OFFICE-CITY-0450 PIC X(15).
00494 04 OFFICE-STATE-0450 PIC X(2).
00495 04 OFFICE-ZIP-0450.
00496 05 OFFICE-ZIP-FIRST-FIVE-0450 PIC X(5).
00498 05 OFFICE-ZIP-LAST-FOUR-0450
00499    PIC X(4).
00500 03 OFFICE-PHONE-0450 PIC 9(7)
00501    OCCURS 3.
00502 03 OFFICE-AREA-CODE-0450 PIC X(3).
00503 03 SPEED-DIAL-0450 PIC X(3).
00504 03 FILLER PIC X(4).
00505 03 SUBSCHEMA-LR-CTRL-END PIC X.
00506
DMLC 00507 01 COPY IDMS MAP-CONTROLS.
00508 01 MRB-EMPMAPLR.
00509 03 MRB-EMPMAPLR-ID PIC X(8).
00510 03 MRB-EMPMAPLR-MCOMP-VER.
00511 05 MRB-EMPMAPLR-MCOMP-DATE
00512 PIC X(8).
00513 05 MRB-EMPMAPLR-MCOMP-TIME
00514 PIC X(6).
00515 05 MRB-EMPMAPLR-MCOMP-VERID
00516 PIC X(2).
00517 03 MRB-EMPMAPLR-SUBSCHEMA PIC X(8).
00518 03 MRB-EMPMAPLR-FLGS PIC X
00519 OCCURS 4.
00520 03 FILLER PIC X(6).
00521 03 MRB-EMPMAPLR-NFLDS PIC S9(4)
00522 USAGE COMP.
00523 03 MRB-EMPMAPLR-NRECS PIC S9(4)
00524
00525 03 MRB-EMPMAPLR-RECOF PIC S9(4)
00526 USAGE COMP.
00527 03 MRB-EMPMAPLR-FLGS PIC X.
00528 03 MRB-EMPMAPLR-TEMP-CURSOR
00529 PIC XX.
00530 03 MRB-EMPMAPLR-TEMP-CURSOR
00531    PIC XX.
00532 03 MRB-EMPMAPLR-TEMP-WCC PIC X.
00533 03 MRB-EMPMAPLR-TEMP-WCC PIC X.
00534 03 MRB-EMPMAPLR-FLGS PIC X.
00535 03 MRB-EMPMAPLR-FLGS PIC X.
00536 03 MRB-EMPMAPLR-FLGS PIC X.
00537 03 MRB-EMPMAPLR-SEGVIEW PIC X.
00538 03 FILLER PIC X.
00539 03 MRB-EMPMAPLR-MREO PIC S9(4)
00540 USAGE COMP.
00541 03 MRB-EMPMAPLR-ERR-CNT PIC S9(4)
00542 USAGE COMP.
00543 03 MRB-EMPMAPLR-ATTR-FLGS PIC X
00544 OCCURS 4.
00545 03 MRB-EMPMAPLR-ERR-FLGS PIC S9(4)
00546 USAGE COMP.
00547 03 MRB-EMPMAPLR-XTYP PIC X.
00548 03 MRB-EMPMAPLR-XTYP PIC X.
00549 03 MRB-EMPMAPLR-MRE-XLEN PIC S9(4)
00550 USAGE COMP.
00551 03 MRB-EMPMAPLR-MRE-XLEN PIC S9(4)
00552 USAGE COMP.
00553 03 MRB-EMPMAPLR-MRE OCCURS 11.
00554 05 MRB-EMPMAPLR-MRE-FLGS
00555 PIC X
00556 OCCURS 8.
00557 05 MRB-EMPMAPLR-MRE-INLEN
00558    PIC S9(4)
00559 USAGE COMP.
00560 05 MRB-EMPMAPLR-MRE-PAD-CHAR
00561 PIC X
00562 OCCURS 2.
00563 05 MRB-EMPMAPLR-MRE-FLG2
00564 PIC X
00565 OCCURS 2.
00566 03 MRB-EMPMAPLR-RECS PIC S9(8)
00567 OCCURS 5
00568 USAGE COMP
00569      SYNC.
00570      03 MRB-EMMPRLR-END PIC X.
00571      03 MRB-EMMPRLR-MRE-SUB PIC S9(4)
00572      USAGE COMP.
00573
00574      EJECT
00575      01 MRB-FDLST PIC S9(8)
00576
00577      OCCURS 6
00578      USAGE COMP.
00579
00580      PROCEDURE DIVISION.
00581
00582      * PROCEDURE DIVISION GENERAL STRATEGY: *
00583      * RETRIEVE INFORMATION FOR A SPECIFIED EMPLOYEE. *
00584      * DISPLAYED DATA INCLUDES EMPLOYEE, DEPARTMENT, *
00585      * JOB, AND OFFICE INFORMATION. *
00586      * THIS PROGRAM USES THE EMP-JOB-LR LOGICAL RECORD= *
00587      * PROGRAM STRATEGY: *
00588      * ** CHECK FOR TASK CODE: TSK01= INITIAL MAPOUT *
00589      * ANYTHING ELSE = RETRIEVE LR *
00590      * ** CLEAR TO EXIT APPLICATION *
00591      * ** ENTER AND NEW EMP-ID TO CONTINUE *
00592      * *********************************************************
00593
00594      MAIN-LINE.
00595      ***************************************************************
00596      * THE BIND MAP STATEMENTS ADVISE IDMS-DC OF THE LOCATION OF *
00597      * THE MBR AND THE MAP RECORDS. *
00598      ***************************************************************
00599      DMLC0001 00599      BIND MAP EMMPRLR.
00600      DMLC0002 00628      BIND MAP EMMPRLR RECORD EMPLOYEE.
00601      DMLC0003 00635      BIND MAP EMMPRLR RECORD DEPARTMENT.
00602      DMLC0004 00642      BIND MAP EMMPRLR RECORD JOB.
00603      DMLC0005 00649      BIND MAP EMMPRLR RECORD OFFICE.
00604      DMLC0006 00656      BIND MAP EMMPRLR RECORD EMP-DATE-WORK-REC.
00605      00663      *
00606      DMLC0007 00664      ACCEPT TASK CODE INTO TASK-CODE.
00607      IF TASK-CODE = TSK01
00608      GO TO INITIAL-MAPOUT
00609      ELSE
00610      GO TO GET-EMP.
00611      ***************************************************************
00612      ***************************************************************
00613      * THE INITIAL-MAPOUT PARAGRAPH IS PERFORMED IF THE CALLING *
00614      * TASK CODE IS TSK01. *
00615      ***************************************************************
00616      * THE MODIFY MAP STATEMENT Assigns THE PROTECTED *
00617      * ATTRIBUTE TO ALL MAP FIELDS EXCEPT EMP-ID-0415. *
00618      ***************************************************************
00619      * THE MAP OUT STATEMENT TRANSMITS THE EMMPRLR MAP *
00620      * TO THE TERMINAL. *
00621      ***************************************************************
00622      * THE DC RETURN STATEMENT SPECIFIES THAT THE NEXT *
00623      * TASK THAT WILL BE INITIATED ON THE SAME TERMINAL WHEN THE *
00624      * OPERATOR PRESSES A CONTROL KEY WILL BE TSK02. *
00625      ***************************************************************
00626      INITIAL-MAPOUT.
00627      DMLC0008 00691      MODIFY MAP EMMPRLR TEMPORARY
00628      DMLC0008 00692      FOR ALL EXCEPT EMP-ID-0415
00629      DMLC0008 00693      ATIBUTES PROTECTED.
00630      00707      *
00631      DMLC0009 00708      MOVE ZERO TO EMP-ID-0415.
00632      DMLC0009 00709      MAP OUT USING EMMPRLR
00633      DMLC0009 00710      OUTPUT DATA IS YES NEWPAGE
00634      DMLC0009 00711      MESSAGE IS INITIAL-MESSAGE LENGTH 80.
00635      00722
00636      DMLC0010 00723      DC RETURN
00637      DMLC0010 00724      NEXT TASK CODE TSK02.
00638      00731      INITIAL-MAPOUT-EXIT.
00639      00732      EXIT.
00733  **********************************************************************
00734  * THE GET-EMP PARAGRAPH IS PERFORMED IF THE CALLING TASK   *
00735  * CODE IS NOT TSK01.                     *
00736  **********************************************************************
00738  * THE MAP IN STATEMENT TRANSMITS DATA FROM THE TERMINAL TO   *
00739  * VARIABLE STORAGE DATA FIELDS. *
00740  **********************************************************************
00741  * THIS FIRST INQUIRE MAP STATEMENT IS USED TO DETERMINE   *
00742  * THE AID KEY PRESSED. *
00743  **********************************************************************
00744  * THIS SECOND INQUIRE MAP STATEMENT USES AUTOMATIC EDITING *
00745  * TO DETERMINE IF THE DATA ENTERED IS CONSISTENT WITH   *
00746  * THE EXTERNAL PICTURE OF THE NAMED DATA ELEMENT. *
00747  **********************************************************************
00748  * THE MAP OUT STATEMENT TRANSMITS DATA FROM THE   *
00749  * EMP-JOB-LR LOGICAL RECORD IN VARIABLE STORAGE TO MAP *
00750  **********************************************************************
00752  GET-EMP.
DMLC0011 00753      MAP IN USING EMMPAPLR.
       *
DMLC0012 00763      INQUIRE MAP EMMPAPLR
DMLC0013 00764      MOVE AID TO DC-AID-IND-V.
DMLC0014 00765      IF CLEAR-HIT
DMLC0015 00766      DC RETURN.
       *
DMLC0016 00767      OBTAIN EMP-JOB-LR
DMLC0017 00768      WHERE EMP-ID-0415 = EMP-ID-0415 OF LR
DMLC0018 00769      ON LR-NOT-FOUND
DMLC0019 00770      GO TO NOT-FOUND.
DMLC0020 00846      FINISH.

00757  **********************************************************************
00758  * SINCE THE MAP FIELD IS ASSOCIATED WITH THE EMP-ID-0415   *
00759  * FIELD, THE PROGRAM USES THE "OF LR" RETRIEVAL. NOTE THAT  *
00760  * AUTOSTATUS IMPLICITLY CHECKS FOR THE LR-ERROR PATH STATUS. *
00761  **********************************************************************
DMLC 00797      COPY IDMS SUBSCHEMA-BINDS.
00798 MOV 'EMPDISP ' TO PROGRAM-NAME
DMLC0015 00799 BIND RUN-UNIT.
DMLC0016 00810      READY USAGE-MODE IS RETRIEVAL.
DMLC0017 00811      * SINCE THE MAP FIELD IS ASSOCIATED WITH THE EMP-ID-0415 *
DMLC0018 00812      * FIELD, THE PROGRAM USES THE "OF LR" RETRIEVAL. NOTE THAT *
DMLC0019 00813      * AUTOSTATUS IMPLICITLY CHECKS FOR THE LR-ERROR PATH STATUS. *

00773  **********************************************************************
00774  * REFORMAT DATE TO MMDDYY; OUTPUT AS MM/DD/YY USING THE OLM *
00775  * EXTERNAL PICTURE SPECIFICATION (XX/XX/XX). *
00776  **********************************************************************
DMLC 00853      MOVE START-YEAR-0415 TO WORK-YY.
DMLC 00854      MOVE START-MONTH-0415 TO WORK-MM.
DMLC 00855      MOVE START-DAY-0415 TO WORK-DD.

DMLC 00856      MAP OUT USING EMMPAPLR
DMLC0019 00860      OUTPUT DATA IS YES
DMLC0019 00861      MESSAGE IS DISPLAY-MESSAGE LENGTH 80.
DMLC0020 00873      EXIT.

00783  **********************************************************************
00784  * REFORMAT DATE TO MMDDYY; OUTPUT AS MM/DD/YY USING THE OLM *
00785  * EXTERNAL PICTURE SPECIFICATION (XX/XX/XX). *
00786  **********************************************************************
DMLC0019 00853      MOVE START-YEAR-0415 TO WORK-YY.
DMLC0019 00854      MOVE START-MONTH-0415 TO WORK-MM.
DMLC0019 00855      MOVE START-DAY-0415 TO WORK-DD.

00787  **********************************************************************
00788  * THE MODIFY MAP STATEMENT SPECIFIES THAT ALL MAP *
00789  * FIELDS EXCEPT THE INCORRECT EMP-ID-0415 FIELD WILL BE *
00790  * ERASED ON THE NEXT MAP OUT. *
00791  **********************************************************************
DMLC0021 00880      MODIFY MAP EMMPAPLR TEMPORARY
DMLC0021 00881      FOR ALL EXCEPT DFLD EMP-ID-0415

            15-Jan-2018  341/415
Sample Online COBOL Program from the COBOL Compiler

00001 *NO-ACTIVITY-LOG
00002 *DMLIST
00003 00004 IDENTIFICATION DIVISION.
00005 00006 PROGRAM-ID. EMPDISP.
00007

00002 MESSAGES FOR PROGRAM EMPDISP

00001 *NO-ACTIVITY-LOG
00002 *DMLIST
00003 00004 IDENTIFICATION DIVISION.
00005 00006 PROGRAM-ID. EMPDISP.
00007

00002 MESSAGES FOR PROGRAM EMPDISP

00001 *NO-ACTIVITY-LOG
00002 *DMLIST
00003 00004 IDENTIFICATION DIVISION.
00005 00006 PROGRAM-ID. EMPDISP.
00007

00002 MESSAGES FOR PROGRAM EMPDISP

00001 *NO-ACTIVITY-LOG
00002 *DMLIST
00003 00004 IDENTIFICATION DIVISION.
00005 00006 PROGRAM-ID. EMPDISP.
00007

00002 MESSAGES FOR PROGRAM EMPDISP
00008  AUTHOR.     COMPUTER ASSOCIATES.
00009
00010  DATE-WRITTEN.  APRIL 1995.
00011
00012  REMARKS.  THIS PROGRAM DEMONSTRATES
00013  CA IDMS PROGRAMMING USING
00014  THE LOGICAL RECORD FACILITY.
00015
00016  ************************************************************************************
00017  ENVIRONMENT DIVISION.
00018  ************************************************************************************
00019  *IDMS-CONTROL SECTION.
00020
00021  *PROTOCOL.  MODE IS IDMS-DC DEBUG
00022  *                  IDMS-RECORDS MANUAL.
00023
00024  DATA DIVISION.
00025
00026  *SCHEMA SECTION.
00027
00028  *DB EMPSS09 WITHIN EMPSCHM.
00029
00030  *MAP SECTION.
00031  *MAX FIELD LIST IS 5.
00032  *MAP EMPMAPIR VERSION 1 TYPE IS STANDARD.
00033
00034
00035
00036  WORKING-STORAGE SECTION.
00037  01 TASK-CODE PIC X(8).
00038  01 TSK01 PIC X(8) VALUE 'TSK01'.
00039  01 TSK02 PIC X(8) VALUE 'TSK02'.
00040
00041  01 MESSAGES.
00042  05 INITIAL-MESSAGE PIC X(80) VALUE
00043      'ENTER AN EMP ID AND PRESS ENTER ** CLEAR TO EXIT'.
00044  05 EDIT-ERROR-MESSAGE PIC X(80) VALUE
00045      'EMP-ID EITHER NOT ENTERED OR NOT NUMERIC'.
00046  05 EMP-NOT-FOUND-MESSAGE PIC X(80) VALUE
00047      'SPECIFIED EMPLOYEE COULD NOT BE FOUND'.
00048  05 DISPLAY-MESSAGE PIC X(80) VALUE
00049      'CLEAR TO EXIT ** NEW EMP-ID AND ENTER TO CONTINUE'.
00050
00051  *01 COPY IDMS DC-AID-CONDITION-NAMES.
00052  01 DC-AID-CONDITION-NAMES.
00053  03 DC-AID-IND-V PIC X.
00054  88 ENTER-HIT VALUE QUOTE.
00055  88 CLEAR-HIT VALUE ' '.
00056  88 PF01-HIT VALUE '1'.
00057  88 PF02-HIT VALUE '2'.
00058  88 PF03-HIT VALUE '3'.
00059  88 PF04-HIT VALUE '4'.
00060  88 PF05-HIT VALUE '5'.
00061  88 PF06-HIT VALUE '6'.
00062  88 PF07-HIT VALUE '7'.
00063  88 PF08-HIT VALUE '8'.
00064  88 PF09-HIT VALUE '9'.
00065  88 PF10-HIT VALUE '>'.
00066  88 PF11-HIT VALUE '#'.
00067  88 PF12-HIT VALUE '@'.
00068  88 PF13-HIT VALUE 'A'.
00069  88 PF14-HIT VALUE 'B'.
00070  88 PF15-HIT VALUE 'C'.
00071  88 PF16-HIT VALUE 'D'.
00072  88 PF17-HIT VALUE 'E'.
00073  88 PF18-HIT VALUE 'F'.
00074  88 PF19-HIT VALUE 'G'.
00075  88 PF20-HIT VALUE 'H'.
00076  88 PF21-HIT VALUE 'I'.
00077  88 PF22-HIT VALUE 'J'.
00078  88 PF23-HIT VALUE '.'.
00079  88 PF24-HIT VALUE '<'.

15-Jan-2018  343/415
00080   88 PA01-HIT VALUE '(%).
00081   88 PA02-HIT VALUE '>."
00082   88 PA03-HIT VALUE ','.
00083   88 PEN-ATTN-SPACE-NULL VALUE '='.
00084   88 PEN-ATTN VALUE QUOTE.
00085
00086   *01 COPY IDMS EMP-DATE-WORK-REC.
00087   01 EMP-DATE-WORK-REC.
00088   02 WORK-DATE.
00089   03 WORK-MM PIC 9(2).
00090   03 WORK-DD PIC 9(2).
00091   03 WORK-YY PIC 9(2).
00092
00093   *01 COPY IDMS SUBSCHEMA-LR-CONTROL.
00094   01 SUBSCHEMA-CTRL.
00095   03 PROGRAM-NAME PIC X(8) VALUE SPACES.
00096   03 ERROR-STATUS PIC X(4) VALUE '1400'.
00097   88 DB-STATUS-OK VALUE '0000'.
00098   88 ANY-STATUS
00099   VALUE '0000' THRU '9999'.
00100   88 ANY-ERROR-STATUS
00101   VALUE '0001' THRU '9999'.
00102   88 DB-END-OF-SET VALUE '0307'.
00103   88 DB-REC-NOT-FOUND VALUE '0326'.
00104   88 DC-DEADLOCK VALUE '3101'
00105   '3201' '3401' '3901'.
00106   88 DC-NO-STORAGE VALUE '3202'
00107   '3402'.
00108   88 DC-AREA-ID-UNK VALUE '4303'.
00109   88 DC-QUEUE-ID-UNK VALUE '4404'.
00110   88 DC-REC-NOT-FOUND VALUE '4305'
00111   '4405'.
00112   88 DC-RESOURCE-NOT-AVAIL
00113   VALUE '3908'.
00114   88 DC-RESOURCE-AVAIL
00115   VALUE '3909'.
00116   88 DC-NEW-STORAGE VALUE '3210'.
00117   88 DC-MAX-TASKS VALUE '3711'.
00118   88 DC-REC-REPLACED VALUE '4317'.
00119   88 DC-TRUNCATED-DATA
00120   VALUE '4319' '4419'
00121   '4519' '4719'.
00122   88 DC-ATTN-INT VALUE '4525'
00123   '4625'.
00124   88 DC-OPER-CANCEL VALUE '4743'.
00125   88 DC-FIRST-PAGE-SENT
00126   VALUE '4676'.
00127   88 DC-SECOND-STARTPAGE
00128   VALUE '4604'.
00129   88 DC-DETAIL-NOT-FOUND
00130   VALUE '4664'.
00131   03 DBKEY PIC S9(8)
00132   USAGE COMP.
00133   03 RECORD-NAME PIC X(16) VALUE SPACES.
00134   03 RECORD-NAME REDEFINES RECORD-NAME.
00135   05 SSC-NODN PIC X(8).
00136   05 SSC-DBN PIC X(8).
00137   03 AREA-NAME PIC X(16) VALUE SPACES.
00138   03 ERROR-SET PIC X(16) VALUE SPACES.
00139   03 ERROR-RECORD PIC X(16) VALUE SPACES.
00140   03 ERROR-AREA PIC X(16) VALUE SPACES.
00141   03 IDBMSCOM-AREA PIC X(100) VALUE LOW-VALUE.
00142   03 IDBMSCOM REDEFINES IDBMSCOM-AREA
00143   PIC X
00144   OCCURS 100.
00145   03 RDBMSCOM REDEFINES IDBMSCOM-AREA.
00146   05 DB-SUB-ADDR PIC X(4).
00147   05 FILLER PIC X(0096).
00148   03 DIRECT-DBKEY PIC S9(8)
00149   USAGE COMP.
00150   03 DIRECT-DBK REDEFINES DIRECT-DBKEY
00151       PIC S9(8).
00152       USAGE COMP.
00153       03 DCBMSCOM-AREA PIC X(100) VALUE LOW-VALUE.
00154       03 DCBMSCOM REDEFINES DCBMSCOM-AREA
00155       PIC X
00156       OCCURS 100.
00157       03 R1DCBMSCOM REDEFINES DCBMSCOM-AREA.
00158       05 R2DCBMSCOM PIC S9(8)
00159       OCCURS 11
00160       USAGE COMP.
00161       05 DCTR1 PIC X(16).
00162       05 R3DCBMSCOM REDEFINES DCTR1.
00163       07 DCTR2 PIC X(8).
00164       07 R4DCBMSCOM REDEFINES DCTR2.
00165       09 DCTR4 PIC X(4).
00166       09 DCTR5 PIC X(4).
00167       07 DCTR3 PIC X(8).
00168       05 R5DCBMSCOM REDEFINES DCTR1.
00169       07 DCPNUM1 PIC S9(15)
00170       USAGE COMP-3.
00171       05 DCNM1 PIC S9(8)
00172       USAGE COMP.
00173       05 R6DCBMSCOM REDEFINES DCMN1.
00174       07 DCPNUM2 PIC S9(7)
00175       USAGE COMP-3.
00176       05 DCNM2 PIC S9(8)
00177       USAGE COMP.
00178       05 DCNM3 PIC S9(8)
00179       USAGE COMP.
00180       05 DCFLG1 PIC S9(4)
00181       USAGE COMP.
00182       05 DCFLG2 PIC S9(4)
00183       USAGE COMP.
00184       05 DCFLG3 PIC S9(4)
00185       USAGE COMP.
00186       05 DCFLG4 PIC S9(4)
00187       USAGE COMP.
00188       03 SSC-ERRSTAT-SAVE PIC X(4) VALUE SPACES.
00189       03 SSC-DMLSEQ-SAVE PIC S9(8)
00190       USAGE COMP.
00191       03 DML-SEQUENCE PIC S9(8)
00192       USAGE COMP.
00193       03 RECORD-OCCUR PIC S9(8)
00194       USAGE COMP.
00195       03 SUBSCHEMA-CTRL-END PIC X(4) VALUE SPACES.
00196       01 SUBSCHEMA-LR-CTRL.
00197       03 LRC-LRPIXELNG PIC S9(4)
00198       USAGE COMP.
00199       03 LRC-MAXVXP PIC S9(4)
00200       USAGE COMP.
00201       03 LRIDENT PIC X(4) VALUE 'LRC'.
00202       03 LRVERB PIC X(8).
00203       03 LRNAME PIC X(16).
00204       03 LR-STATUS PIC X(16).
00205       03 FILLER PIC X(16).
00206       03 LRPIXE PIC X
00207       OCCURS 0 TO 512 DEPENDING ON LRC-LRPIXELNG.
00208       03 PXE.
00209       05 PXENEXTPIC S9(8)
00210       USAGE COMP.
00211       05 PXETABO PIC S9(4)
00212       USAGE COMP.
00213       05 PXEDSPL PIC S9(4)
00214       USAGE COMP.
00215       05 PXEDYN PIC S9(4)
00216       USAGE COMP.
00217       05 PXEDLEN PIC S9(4)
00218       USAGE COMP.
00219       USAGE COMP.
00220       05 PXENDEC PIC X.
00221       05 PXEDTYP PIC X.
00222 05 PXEOTYP PIC X.
00223 05 PXEFLAG PIC X.
00224 05 FILLER PIC X(240).
00225 03 PXEDSP256 REDEFINES PXE
00226       PIC X(256).
00227 03 PXEDSP248 REDEFINES PXE
00228       PIC X(248).
00229 03 PXEDSP240 REDEFINES PXE
00230       PIC X(240).
00231 03 PXEDSP232 REDEFINES PXE
00232       PIC X(232).
00233 03 PXEDSP224 REDEFINES PXE
00234       PIC X(224).
00235 03 PXEDSP216 REDEFINES PXE
00236       PIC X(216).
00237 03 PXEDSP208 REDEFINES PXE
00238       PIC X(208).
00239 03 PXEDSP200 REDEFINES PXE
00240       PIC X(200).
00241 03 PXEDSP192 REDEFINES PXE
00242       PIC X(192).
00243 03 PXEDSP184 REDEFINES PXE
00244       PIC X(184).
00245 03 PXEDSP176 REDEFINES PXE
00246       PIC X(176).
00247 03 PXEDSP168 REDEFINES PXE
00248       PIC X(168).
00249 03 PXEDSP160 REDEFINES PXE
00250       PIC X(160).
00251 03 PXEDSP152 REDEFINES PXE
00252       PIC X(152).
00253 03 PXEDSP144 REDEFINES PXE
00254       PIC X(144).
00255 03 PXEDSP136 REDEFINES PXE
00256       PIC X(136).
00257 03 PXEDSP128 REDEFINES PXE
00258       PIC X(128).
00259 03 PXEDSP120 REDEFINES PXE
00260       PIC X(120).
00261 03 PXEDSP112 REDEFINES PXE
00262       PIC X(112).
00263 03 PXEDSP104 REDEFINES PXE
00264       PIC X(104).
00265 03 PXEDSP96 REDEFINES PXE
00266       PIC X(96).
00267 03 PXEDSP88 REDEFINES PXE
00268       PIC X(88).
00269 03 PXEDSP80 REDEFINES PXE
00270       PIC X(80).
00271 03 PXEDSP72 REDEFINES PXE
00272       PIC X(72).
00273 03 PXEDSP64 REDEFINES PXE
00274       PIC X(64).
00275 03 PXEDSP56 REDEFINES PXE
00276       PIC X(56).
00277 03 PXEDSP48 REDEFINES PXE
00278       PIC X(48).
00279 03 PXEDSP40 REDEFINES PXE
00280       PIC X(40).
00281 03 PXEDSP32 REDEFINES PXE
00282       PIC X(32).
00283 03 PXEDSP24 REDEFINES PXE
00284       PIC X(24).
00285 03 PXEDSP16 REDEFINES PXE
00286       PIC X(16).
00287 03 PXEDSP8 REDEFINES PXE
00288       PIC X(8).
00289 03 PXECOMP-1 REDEFINES PXE
00290       USAGE COMP-1.
00291 03 PXECOMP-2 REDEFINES PXE
00292       USAGE COMP-2.
00293        03 PXECOMP-30       REDEFINES PXE
00294            PIC S9(18)
00295            USAGE COMP-3.
00296        03 PXECOMP-31       REDEFINES PXE
00297            PIC S9(17)V9(1)
00298            USAGE COMP-3.
00299        03 PXECOMP-32       REDEFINES PXE
00300            PIC S9(16)V9(2)
00301            USAGE COMP-3.
00302        03 PXECOMP-33       REDEFINES PXE
00303            PIC S9(15)V9(3)
00304            USAGE COMP-3.
00305        03 PXECOMP-34       REDEFINES PXE
00306            PIC S9(14)V9(4)
00307            USAGE COMP-3.
00308        03 PXECOMP-35       REDEFINES PXE
00309            PIC S9(13)V9(5)
00310        03 PXECOMP-36       REDEFINES PXE
00311            PIC S9(12)V9(6)
00312            USAGE COMP-3.
00313        03 PXECOMP-37       REDEFINES PXE
00314            PIC S9(11)V9(7)
00315            USAGE COMP-3.
00316        03 PXECOMP-38       REDEFINES PXE
00317            PIC S9(10)V9(8)
00318            USAGE COMP-3.
00319        03 PXECOMP-39       REDEFINES PXE
00320            PIC S9(9)V9(9)
00321        03 PXECOMP-310      REDEFINES PXE
00322            PIC S9(8)V9(10)
00323        03 PXECOMP-311      REDEFINES PXE
00324            PIC S9(7)V9(11)
00325            USAGE COMP-3.
00326        03 PXECOMP-312      REDEFINES PXE
00327            PIC S9(6)V9(12)
00328            USAGE COMP-3.
00329        03 PXECOMP-313      REDEFINES PXE
00330            PIC S9(5)V9(13)
00331        03 PXECOMP-314      REDEFINES PXE
00332            PIC S9(4)V9(14)
00333        03 PXECOMP-315      REDEFINES PXE
00334            PIC S9(3)V9(15)
00335        03 PXECOMP-316      REDEFINES PXE
00336            PIC S9(2)V9(16)
00337        03 PXECOMP-317      REDEFINES PXE
00338            PIC S9(1)V9(17)
00339        03 PXECOMP-318      REDEFINES PXE
00340            PIC SV9(18)
00341        03 PXECOMP20       REDEFINES PXE
00342            PIC S9(4)
00343        03 PXECOMP21       REDEFINES PXE
00344            PIC S9(3)V9(1)
00345        03 PXECOMP22       REDEFINES PXE
00346            PIC S9(2)V9(2)
00347        03 PXECOMP23       REDEFINES PXE
00348            PIC S9(1)V9(3)
00349        03 PXECOMP24       REDEFINES PXE
00350            PIC SV9(4)
<table>
<thead>
<tr>
<th>Line</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>00364</td>
<td>03 PXECOMP40 USAGE COMP. REDEFINES PXE</td>
</tr>
<tr>
<td>00366</td>
<td>PIC S9(9)</td>
</tr>
<tr>
<td>00367</td>
<td>USAGE COMP.</td>
</tr>
<tr>
<td>00368</td>
<td>03 PXECOMP41 REDEFINES PXE</td>
</tr>
<tr>
<td>00369</td>
<td>PIC S9(8)V9(1) USAGE COMP.</td>
</tr>
<tr>
<td>00371</td>
<td>03 PXECOMP42 REDEFINES PXE</td>
</tr>
<tr>
<td>00372</td>
<td>PIC S9(7)V9(2) USAGE COMP.</td>
</tr>
<tr>
<td>00374</td>
<td>03 PXECOMP43 REDEFINES PXE</td>
</tr>
<tr>
<td>00375</td>
<td>PIC S9(6)V9(3) USAGE COMP.</td>
</tr>
<tr>
<td>00377</td>
<td>03 PXECOMP44 REDEFINES PXE</td>
</tr>
<tr>
<td>00378</td>
<td>PIC S9(5)V9(4) USAGE COMP.</td>
</tr>
<tr>
<td>00380</td>
<td>03 PXECOMP45 REDEFINES PXE</td>
</tr>
<tr>
<td>00381</td>
<td>PIC S9(4)V9(5) USAGE COMP.</td>
</tr>
<tr>
<td>00383</td>
<td>03 PXECOMP46 REDEFINES PXE</td>
</tr>
<tr>
<td>00384</td>
<td>PIC S9(3)V9(6) USAGE COMP.</td>
</tr>
<tr>
<td>00386</td>
<td>03 PXECOMP47 REDEFINES PXE</td>
</tr>
<tr>
<td>00387</td>
<td>PIC S9(2)V9(7) USAGE COMP.</td>
</tr>
<tr>
<td>00388</td>
<td>03 PXECOMP48 REDEFINES PXE</td>
</tr>
<tr>
<td>00390</td>
<td>PIC S9(1)V9(8) USAGE COMP.</td>
</tr>
<tr>
<td>00392</td>
<td>03 PXECOMP49 REDEFINES PXE</td>
</tr>
<tr>
<td>00393</td>
<td>PIC S9(9) USAGE COMP.</td>
</tr>
<tr>
<td>00394</td>
<td>03 PXECOMP80 REDEFINES PXE</td>
</tr>
<tr>
<td>00396</td>
<td>PIC S9(18) USAGE COMP.</td>
</tr>
<tr>
<td>00398</td>
<td>03 PXECOMP81 REDEFINES PXE</td>
</tr>
<tr>
<td>00399</td>
<td>PIC S9(17)V9(1) USAGE COMP.</td>
</tr>
<tr>
<td>00400</td>
<td>03 PXECOMP82 REDEFINES PXE</td>
</tr>
<tr>
<td>00401</td>
<td>PIC S9(16)V9(2) USAGE COMP.</td>
</tr>
<tr>
<td>00402</td>
<td>03 PXECOMP83 REDEFINES PXE</td>
</tr>
<tr>
<td>00403</td>
<td>PIC S9(15)V9(3) USAGE COMP.</td>
</tr>
<tr>
<td>00404</td>
<td>03 PXECOMP84 REDEFINES PXE</td>
</tr>
<tr>
<td>00405</td>
<td>PIC S9(14)V9(4) USAGE COMP.</td>
</tr>
<tr>
<td>00406</td>
<td>03 PXECOMP85 REDEFINES PXE</td>
</tr>
<tr>
<td>00408</td>
<td>PIC S9(13)V9(5) USAGE COMP.</td>
</tr>
<tr>
<td>00409</td>
<td>03 PXECOMP86 REDEFINES PXE</td>
</tr>
<tr>
<td>00410</td>
<td>PIC S9(12)V9(6) USAGE COMP.</td>
</tr>
<tr>
<td>00412</td>
<td>03 PXECOMP87 REDEFINES PXE</td>
</tr>
<tr>
<td>00414</td>
<td>PIC S9(11)V9(7) USAGE COMP.</td>
</tr>
<tr>
<td>00416</td>
<td>03 PXECOMP88 REDEFINES PXE</td>
</tr>
<tr>
<td>00418</td>
<td>PIC S9(10)V9(8) USAGE COMP.</td>
</tr>
<tr>
<td>00419</td>
<td>03 PXECOMP89 REDEFINES PXE</td>
</tr>
<tr>
<td>00420</td>
<td>PIC S9(9)V9(9) USAGE COMP.</td>
</tr>
<tr>
<td>00421</td>
<td>03 PXECOMP810 REDEFINES PXE</td>
</tr>
<tr>
<td>00422</td>
<td>PIC S9(8)V9(10) USAGE COMP.</td>
</tr>
<tr>
<td>00424</td>
<td>03 PXECOMP811 REDEFINES PXE</td>
</tr>
<tr>
<td>00426</td>
<td>PIC S9(7)V9(11) USAGE COMP.</td>
</tr>
<tr>
<td>00427</td>
<td>03 PXECOMP812 REDEFINES PXE</td>
</tr>
<tr>
<td>00429</td>
<td>PIC S9(6)V9(12) USAGE COMP.</td>
</tr>
<tr>
<td>00431</td>
<td>03 PXECOMP813 REDEFINES PXE</td>
</tr>
</tbody>
</table>
00435  PIC  S9(5)V9(13)
00436  USAGE COMP.
00437 03 PXECOMP814  REDEFINES PXE
00438  PIC  S9(4)V9(14)
00439  USAGE COMP.
00440 03 PXECOMP815  REDEFINES PXE
00441  PIC  S9(3)V9(15)
00442  USAGE COMP.
00443 03 PXECOMP816  REDEFINES PXE
00444  PIC  S9(2)V9(16)
00445  USAGE COMP.
00446 03 PXECOMP817  REDEFINES PXE
00447  PIC  S9(1)V9(17)
00448  USAGE COMP.
00449 03 PXECOMP818  REDEFINES PXE
00450  PIC  SV9(18)
00451  USAGE COMP.
00452 01 SUBSCHEMA-SSNAME  PIC  X(8)  VALUE  'EMPSS09 '.
00453 01 SUBSCHEMA-AREANAMES.
00454 03 EMP-DEMO-REGION  PIC  X(16)
00455  VALUE 'EMP-DEMO-REGION '
00456 03 INS-DEMO-REGION  PIC  X(16)
00457  VALUE 'INS-DEMO-REGION '
00458 03 ORG-DEMO-REGION  PIC  X(16)
00459  VALUE 'ORG-DEMO-REGION '
00460 00461  *01 COPY IDMS SUBSCHEMA-LR-RECORDS.
00462 01 EMP-JOB-LR.
00463 02 EMPLOYEE.
00464 03 EMP-ID-0415  PIC  9(4).
00465 03 EMP-NAME-0415.
00466 04 EMP-FIRST-NAME-0415  PIC  X(10).
00467 04 EMP-LAST-NAME-0415  PIC  X(15).
00468 03 STATUS-0415  PIC  X(2).
00469 88 ACTIVE-0415  VALUE '01'.
00470 88 ST-DISABIL-0415  VALUE '02'.
00471 88 LT-DISABIL-0415  VALUE '03'.
00472 88 LEAVE-OF-ABSENCE-0415  VALUE '04'.
00473 88 TERMINATED-0415  VALUE '05'.
00474 03 SS-NUMBER-0415  PIC  9(9).
00475 03 START-DATE-0415.
00476 04 START-YEAR-0415  PIC  9(2).
00477 04 START-MONTH-0415  PIC  9(2).
00478 04 START-DAY-0415  PIC  9(2).
00479 03 FILLER  PIC  X(2).
00480 02 DEPARTMENT.
00481 03 DEPT-ID-0410  PIC  9(4).
00482 03 DEPT-NAME-0410  PIC  X(45).
00483 03 DEPT-HEAD-ID-0410  PIC  9(4).
00484 03 OFFICE-ID-0410  PIC  9(4).
00485 03 FILLER  PIC  XXX.
00486 02 JOB.
00487 03 JOB-ID-0440  PIC  9(4).
00488 03 TITLE-0440  PIC  X(20).
00489 02 OFFICE.
00490 03 OFFICE-CODE-0450  PIC  X(3).
00491 03 OFFICE-ADDRESS-0450.
00492 04 OFFICE-ADDRESS-0450  PIC  X(20).
00493 04 OFFICE-CITY-0450  PIC  X(15).
00494 04 OFFICE-STATE-0450  PIC  X(2).
00495 04 OFFICE-ZIP-0450.
00496 05 OFFICE-ZIP-FIRST-FIVE-0450  PIC  X(5).
00497 05 OFFICE-ZIP-LAST-FOUR-0450  PIC  X(4).
00498 03 FILLER  PIC  X(4).
00499 03 OFFICE-PHONE-0450  PIC  9(7).
00500 03 OFFICE-AREA-CODE-0450  PIC  X(3).
00501 03 OCCURS 3.
00502 03 OFFICE-AREA-CODE-0450  PIC  X(3).
00503 03 OFFICE-DIAL-0450  PIC  X(3).
00504 03 FILLER  PIC  X(4).
00505 03 SUBSCHEMA-LR-CTRL-END  PIC  X.
00508 01 MRB-EMMPMLR.
00509 03 MRB-EMMPMLR-ID PIC X(8).
00510 03 MRB-EMMPMLR-MCOMP-VER.
00511 05 MRB-EMMPMLR-MCOMP-DATE
00512 PIC X(8).
00513 05 MRB-EMMPMLR-MCOMP-TIME
00514 PIC X(6).
00515 05 MRB-EMMPMLR-MCOMP-VERID
00516 PIC X(2).
00517 03 MRB-EMMPMLR-SUBSCHEMA PIC X(8).
00518 03 MRB-EMMPMLR-FLGS PIC X
00519 OCCURS 4.
00520 03 FILLER PIC X(6).
00521 03 MRB-EMMPMLR-NFLDS PIC S9(4)
00522 USAGE COMP.
00523 03 MRB-EMMPMLR-NRECS PIC S9(4)
00524 USAGE COMP.
00525 03 MRB-EMMPMLR-RECOF PIC S9(4)
00526 USAGE COMP.
00527 03 MRB-EMMPMLR-RECS PIC S9(8)
00528 03 MRB-EMMPMLR-TEMP-CURSOR
00529 03 MRB-EMMPMLR-TEMP-CURSOR
00530 PIC XX.
00531 03 MRB-EMMPMLR-PERM-WCC PIC X.
00532 03 MRB-EMMPMLR-TEMP-WCC PIC X.
00533 03 MRB-EMMPMLR-Cursor PIC XX.
00534 03 MRB-EMMPMLR-AID PIC X.
00535 03 MRB-EMMPMLR-INPUT-FLGS
00536 PIC X.
00537 03 MRB-EMMPMLR-SEGVIEW PIC X.
00538 03 FILLER PIC X.
00539 03 MRB-EMMPMLR-MREO PIC S9(4)
00540 USAGE COMP.
00541 03 MRB-EMMPMLR-ERR-CNT PIC S9(4)
00542 USAGE COMP.
00543 03 MRB-EMMPMLR-ATTR-FLGS PIC X
00544 OCCURS 4.
00545 03 MRB-EMMPMLR-CURR-MFLD PIC S9(4)
00546 USAGE COMP.
00547 03 MRB-EMMPMLR-XTYP PIC X.
00548 03 MRB-EMMPMLR-FILLER PIC X.
00549 03 MRB-EMMPMLR-MRE-XLEN PIC S9(4)
00550 USAGE COMP.
00551 03 MRB-EMMPMLR-MRE-XLEN PIC S9(4)
00552 USAGE COMP.
00553 03 MRB-EMMPMLR-MRE OCCURS 11.
00554 05 MRB-EMMPMLR-MRE-FLGS
00555 PIC X
00556 OCCURS 8.
00557 05 MRB-EMMPMLR-MRE-INLEN
00558 PIC S9(4)
00559 USAGE COMP.
00560 05 MRB-EMMPMLR-MRE-PAD-CHAR
00561 PIC X
00562 OCCURS 2.
00563 05 MRB-EMMPMLR-MRE-FLG2
00564 PIC X
00565 OCCURS 2.
00566 03 MRB-EMMPMLR-RECS PIC S9(8)
00567 OCCURS 5
00568 USAGE COMP
00569 SYNC.
00570 03 MRB-EMMPMLR-END PIC X.
00571 03 MRB-EMMPMLR-MRE-SUB PIC S9(4)
00572 USAGE COMP.
00573
00574 01 MRB-FLDLST.
00576 02 FLDLST PIC S9(8)
PROCEDURE DIVISION.

* *********************************************************
* * PROCEDURE DIVISION GENERAL STRATEGY:               *
* * DISPLAYED DATA INCLUDES EMPLOYEE, DEPARTMENT, *
* * JOB, AND OFFICE INFORMATION.                      *
* * ==> THIS PROGRAM USES THE EMP-JOB-LR LOGICAL RECORD<= *
* * PROGRAM STRATEGY:                                 *
* *   ** CHECK FOR TASK CODE: TSK01= INITIAL MAPOUT *
* *   ANYTHING ELSE = RETRIEVE LR                      *
* *   ** CLEAR TO EXIT APPLICATION                    *
* *   ** ENTER AND NEW EMP-ID TO CONTINUE             *
* *********************************************************

MAIN-LINE.

***************************************************************
* THE BIND MAP STATEMENTS ADVISE IDMS-DC OF THE LOCATION OF *
* THE MRB AND THE MAP RECORDS.                             *
***************************************************************

* BIND MAP EMPMAPLR.
00600 MOVE 0001 TO DML-SEQUENCE DMLC0001
00601 CALL 'IDMSCOBI' USING SUBSCHEMA-CTRL
00602 DCMSCOM (90)
00603 MRB-EMPMAPLR
00604 MRB-EMPMAPLR-END
00605 MOVE '08/12/51121414R2'
00606 TO MRB-EMPMAPLR-MCOMP-VER
00607 MOVE 'EMPS009 '
00608 TO MRB-EMPMAPLR-SUBSCHEMA
00609 MOVE 'EMPMAPLR'
00610 TO MRB-EMPMAPLR-ID
00611 MOVE 11

TO MRB-EMPMAPLR-NFLDS
00613 MOVE 5
00614 TO MRB-EMPMAPLR-NRECS
00615 MOVE 156
00616 TO MRB-EMPMAPLR-RECOF
00617 MOVE 76
00618 TO MRB-EMPMAPLR-MREO
00619 MOVE '0'
00620 TO MRB-EMPMAPLR-XTYP
00621 MOVE 0
00622 TO MRB-EMPMAPLR-MRE-XLEN
00623 MOVE 0
00624 TO MRB-EMPMAPLR-MRB-XLEN
00625 MOVE 'Y'
00626 TO MRB-EMPMAPLR-SEGVIEW
00627 PERFORM IDMS-STATUS.

* BIND MAP EMPMAPLR RECORD EMPLOYEE.
00629 MOVE 0002 TO DML-SEQUENCE DMLC0002
00630 CALL 'IDMSCOBI' USING SUBSCHEMA-CTRL
00631 DCMSCOM (91)
00632 MRB-EMPMAPLR-RECS (1)
00633 EMPLOYEE
00634 PERFORM IDMS-STATUS.

* BIND MAP EMPMAPLR RECORD DEPARTMENT.
00635 MOVE 0003 TO DML-SEQUENCE DMLC0003
00636 CALL 'IDMSCOBI' USING SUBSCHEMA-CTRL
00637 DCMSCOM (91)
00638 MRB-EMPMAPLR-RECS (2)
00639 DEPARTMENT
00640 PERFORM IDMS-STATUS.

* BIND MAP EMPMAPLR RECORD JOB.
00641 MOVE 0004 TO DML-SEQUENCE DMLC0004
00642 CALL 'IDMSCOBI' USING SUBSCHEMA-CTRL
00643 DCMSCOM (91)
00644 MRB-EMPMAPLR-RECS (3)
00645 JOB
00648 PERFORM IDMS-STATUS.
00649   * BIND MAP EMPPA MPLR RECORD OFFICE.
00650 MOVE 0005 TO DML-SEQUENCE DMLC0005
00651 CALL 'IDMSCOBI' USING SUBSCHEMA-CTRL
00652 DCBMS COMM (91)
00653 MRB-EMPPA MPLR-RECS (4)
00654 OFFICE
00655 PERFORM IDMS-STATUS.
00656   * BIND MAP EMPPA MPLR RECORD EMP-DATE-WORK-REC.
00657 MOVE 0006 TO DML-SEQUENCE DMLC0006
00658 CALL 'IDMSCOBI' USING SUBSCHEMA-CTRL
00659 DCBMS COMM (91)
00660 MRB-EMPPA MPLR-RECS (5)
00661 EMP-DATE-WORK-REC
00662 PERFORM IDMS-STATUS.
00663   * ACCEPT TASK CODE INTO TASK-CODE.
00664 MOVE 0007 TO DML-SEQUENCE DMLC0007
00665 MOVE 1 TO DCNUM1
00666 CALL 'IDMSCOBI' USING SUBSCHEMA-CTRL
00667 DCBMS COMM (2)
00668 TASK-CODE
00669 PERFORM IDMS-STATUS.
00670 IF TASK-CODE = TSK01
00671   GO TO INITIAL-MAPOUT
00672 ELSE
00673   GO TO GET-EMP.
00674 ***************************************************************
00675 * THE INITIAL-MAPOUT PARAGRAPH IS PERFORMED IF THE CALLING *
00676 * TASK CODE IS TSK01.                                      *
00677 ***************************************************************
00678   * THE INITIAL-MAPOUT PARAGRAPH IS PERFORMED IF THE CALLING *
00679 ***************************************************************
00680 * THE MODIFY MAP STATEMENT ASSIGNS THE PROTECTED            *
00681 * ATTRIBUTE TO ALL MAP FIELDS EXCEPT EMP-ID-0415.         *
00682 ***************************************************************
00683 * THE MAP OUT STATEMENT TRANSMITS THE EMPPA MPLR MAP         *
00684 * TO THE TERMINAL.                                         *
00685 ***************************************************************
00686 * THE DC RETURN STATEMENT SPECIFIES THAT THE NEXT            *
00687 * TASK THAT WILL BE INITIATED ON THE SAME TERMINAL WHEN THE *
00688 * OPERATOR PRESSES A CONTROL KEY WILL BE TSK02.           *
00689 ***************************************************************
00690 INITIAL-MAPOUT.
00691   * MODIFY MAP EMPPA MPLR TEMPORARY                       *
00692   * FOR ALL EXCEPT EMP-ID-0415                            *
00693   * ATTRIBUTES PROTECTED.                                 *
00694 MOVE 0008 TO DML-SEQUENCE DMLC0008
00695 MOVE 8 TO DCNUM1
00696 MOVE 2561 TO DCFLG1
00697 MOVE 0 TO DCFLG2
00698 MOVE 0 TO DCFLG3
00699 MOVE 0 TO DCFLG4
00700 MOVE 1 TO FLDLST (2)
00701 MOVE 1 TO FLDLST (1)
00702 CALL 'IDMSCOBI' USING SUBSCHEMA-CTRL
00703 DCBMS COMM (93)
00704 MRB-EMPPA MPLR
00705 MRB-FLDLST
00706   PERFORM IDMS-STATUS.
00707   * MOVE ZERO TO EMP-ID-0415.                           *
00708   * MAP OUT USING EMPPA MPLR                            *
00709   * OUTPUT DATA IS YES NEWPAGE                         *
00710   * MESSAGE IS INITIAL-MESSAGE LENGTH 80.              *
00711   * MOVE 0009 TO DML-SEQUENCE DMLC0009                 *
00712   MOVE 5 TO DCFLG1                                    *
00713   MOVE 16 TO DCFLG2                                   *
00714   MOVE 1 TO DCFLG3                                    *
00715   MOVE 4 TO DCFLG4                                   *
00716 CALL 'IDMSCOBI' USING SUBSCHEMA-CTRL
00717 DCBMS COMM (34)
Initial-MESSAGE DCBMSCOM (80)

PERFORM IDMS-STATUS.

* DC RETURN

NEXT TASK CODE TSK02.

MOVE 0010 TO DML-SEQUENCE DMLC0010

MOVE TSK02 TO DCSTR2

CALL 'IDMSCOBI' USING SUBSCHEMA-CTRL

DCBMSCOM (19)

PERFORM IDMS-STATUS.

INITIAL-MESSAGE DCBMSCOM (80)

PERFORM IDMS-STATUS.

INITIAL-MAPOUT-EXIT.

EXIT.

* THE GET-EMP PARAGRAPH IS PERFORMED IF THE CALLING TASK CODE IS NOT TSK01.

* THE MAP IN STATEMENT TRANSMITS DATA FROM THE TERMINAL TO VARIABLE STORAGE DATA FIELDS.

* THIS FIRST INQUIRE MAP STATEMENT IS USED TO DETERMINE THE AID KEY PRESSED.

* THIS SECOND INQUIRE MAP STATEMENT USES AUTOMATIC EDITING TO DETERMINE IF THE DATA ENTERED IS CONSISTENT WITH THE EXTERNAL PICTURE OF THE NAMED DATA ELEMENT.

* THE MAP OUT STATEMENT TRANSMITS DATA FROM THE EMP-JOB-LR LOGICAL RECORD IN VARIABLE STORAGE TO MAP FIELDS.

GET-EMP.

* MAP IN USING EMPMaplr.

:edisplay.

MOVE 0011 TO DML-SEQUENCE DMLC0011

MOVE 6 TO DCFLG1

MOVE 0 TO DCFLG2

MOVE 0 TO DCFLG3

MOVE 0 TO DCFLG4

CALL 'IDMSCOBI' USING SUBSCHEMA-CTRL

DCBMSCOM (34)

MRB-EMPMAPLR

PERFORM IDMS-STATUS.

* INQUIRE MAP EMPMaplr

MOVE AID TO DC-AID-IND-V.

MOVE 0012 TO DML-SEQUENCE DMLC0012

MOVE 7 TO DCNUM1

CALL 'IDMSCOBI' USING SUBSCHEMA-CTRL

DCBMSCOM (92)

MRB-EMPMAPLR

PERFORM IDMS-STATUS.

IF CLEAR-HIT

* DC RETURN.

MOVE 0013 TO DML-SEQUENCE DMLC0013

MOVE 0 TO DCFLG1

CALL 'IDMSCOBI' USING SUBSCHEMA-CTRL

DCBMSCOM (19)

PERFORM IDMS-STATUS.

* INQUIRE MAP EMPMaplr

IF DFLD EMP-ID-0415 EDIT IS ERROR

MOVE 0014 TO DML-SEQUENCE DMLC0014

MOVE 17 TO DCNUM1

MOVE 5 TO DCNUM2

MOVE 2048 TO DCFLG1

MOVE 1 TO FDLST (2)
MOVE 1 TO FLDLST (1)
CALL 'IDMSCOBI' USING SUBSCHEMA-CTRL
DCBMSCOM (92)
MRB-EMPMAPLR
MRB-FLDLST;
IF ERROR-STATUS EQUAL TO '4641'
THEN GO TO EDIT-ERROR.
*
COPY IDMS SUBSCHEMA-BINDS.
MOVE 'EMPDISP ' TO PROGRAM-NAME
BIND RUN-UNIT.
MOVE 0015 TO DML-SEQUENCE DMLC0015
MOVE 576 TO LRC-LRPXELNG
MOVE 6 TO LRC-MAXVXP
MOVE 'LRF-BIND' TO LR-STATUS
CALL 'IDMSCOBI' USING SUBSCHEMA-CTRL
IDBMSCOM (59)
SUBSCHEMA-CTRL
SUBSCHEMA-SSNAME
SUBSCHEMA-LR-CTRL
PERFORM IDMS-STATUS.
*
READY USAGE-MODE IS RETRIEVAL.
MOVE 0016 TO DML-SEQUENCE DMLC0016
CALL 'IDMSCOBI' USING SUBSCHEMA-CTRL
IDBMSCOM (37)
PERFORM IDMS-STATUS.
**********************************************************************
* SINCE THE MAP FIELD IS ASSOCIATED WITH THE EMP-ID-0415 FIELD, THE PROGRAM USES THE "OF LR" RETRIEVAL. NOTE THAT AUTOSTATUS IMPLICITLY CHECKS FOR THE LR-ERROR PATH STATUS.
**********************************************************************
OBTAIN EMP-JOB-LR
WHERE EMP-ID-0415 = EMP-ID-0415 OF LR
ON LR-NOT-FOUND
MOVE 0017 TO DML-SEQUENCE DMLC0017
MOVE 0 TO LRC-LRPXELNG
MOVE 0036 TO LRC-MAXVXP
MOVE 'LR-ERROR' TO LR-STATUS
MOVE 'OBTAIN N' TO LRVERB
MOVE 'EMP-JOB-LR' TO LRNAME
MOVE START-YEAR-0415 TO WORK-YY.
MOVE START-MONTH-0415 TO WORK-MM.
MOVE START-DAY-0415 TO WORK-DD.
MAP OUT USING EMPMAPLR
OUTPUT DATA IS YES
MESSAGE IS DISPLAY-MESSAGE LENGTH 80.
MOVE 0019 TO DML-SEQUENCE DMLC0019
MOVE 5 TO DCFLG1
MOVE 16 TO DCFLG2
MOVE 0 TO DCFLG3
MOVE 4 TO DCFLG4
CALL 'IDMSCOBI' USING SUBSCHEMA-CTRL
DCBMSCOM (34)
MRB-EMPMAPLR
DISPLAY-MESSAGE DCBMSCOM (80)
PERFORM IDMS-STATUS.
*
DC RETURN NEXT TASK CODE TSK02.
MOVE 0020 TO DML-SEQUENCE DMLC0020
MOVE TSK02 TO DCSTR2
MOVE 128 TO DCFLG1
CALL 'IDMSCOBI' USING SUBSCHEMA-CTRL
DCBMSCOM (19)
PERFORM IDMS-STATUS.
GET-EMP-EXIT.
EXIT.
**********************************************************************
* THE MODIFY MAP STATEMENT SPECIFIES THAT ALL MAP FIELDS EXCEPT THE INCORRECT EMP-ID-0415 FIELD WILL BE *
**********************************************************************
00887  * ERASED ON THE NEXT MAP OUT.        *
00888  ***************************************************************
00889  EDIT-ERROR.
00890  * MODIFY MAP EMPMAPLR TEMPORARY
00891  * FOR ALL EXCEPT DFLD EMP-ID-0415
00892  * OUTPUT DATA IS ERASE.
00893  MOVE 0021 TO DML-SEQUENCE DMLC0021
00894  MOVE 0 TO DCNUM1
00895  MOVE 2561 TO DCFLG1
00896  MOVE 16 TO DCFLG2
00897  MOVE 0 TO DCFLG3
00898  MOVE 0 TO DCFLG4
00899  MOVE 1 TO FLDLST (2)
00900  MOVE 1 TO FLDLST (1)
00901  CALL 'IDMSCOBI' USING SUBSCHEMA-CTRL
00902  DCBMSCOM (93)
00903  MRB-EMPMAPLR
00904  MRB-FLDLST
00905  PERFORM IDMS-STATUS.
00906  *
00907  * MAP OUT USING EMPMAPLR
00908  * MESSAGE IS EDIT-ERROR-MESSAGE LENGTH 80.
00909  MOVE 0022 TO DML-SEQUENCE DMLC0022
00910  MOVE 5 TO DCFLG1
00911  MOVE 0 TO DCFLG2
00912  MOVE 0 TO DCFLG3
00913  MOVE 4 TO DCFLG4
00914  CALL 'IDMSCOBI' USING SUBSCHEMA-CTRL
00915  DCBMSCOM (34)
00916  MRB-EMPMAPLR
00917  EDIT-ERROR-MESSAGE DCBMSCOM (80)
00918  PERFORM IDMS-STATUS.
00919  *
00920  * DC RETURN
00921  * NEXT TASK CODE TSK02.
00922  MOVE 0023 TO DML-SEQUENCE DMLC0023
00923  MOVE TSK02 TO DCSTR2
00924  MOVE 128 TO DCFLG1
00925  CALL 'IDMSCOBI' USING SUBSCHEMA-CTRL
00926  DCBMSCOM (19)
00927  PERFORM IDMS-STATUS.
00928  *
00929  EXIT.
00930  ***************************************************************
00931  ***************************************************************
00932  * THE FOLLOWING MODIFY MAP STATEMENT SPECIFIES THAT ALL   *
00933  * MAP FIELDS EXCEPT THE EMP-ID-0415 FIELD WILL BE ERASED   *
00934  * ON THE NEXT MAP OUT.        *
00935  ***************************************************************
00936  NOT-FOUND.
00937  * MODIFY MAP EMPMAPLR TEMPORARY
00938  * FOR ALL EXCEPT DFLD EMP-ID-0415
00939  * OUTPUT DATA IS ERASE.
00940  MOVE 0024 TO DML-SEQUENCE DMLC0024
00941  MOVE 0 TO DCNUM1
00942  MOVE 2561 TO DCFLG1
00943  MOVE 16 TO DCFLG2
00944  MOVE 0 TO DCFLG3
00945  MOVE 0 TO DCFLG4
00946  MOVE 1 TO FLDLST (2)
00947  MOVE 1 TO FLDLST (1)
00948  CALL 'IDMSCOBI' USING SUBSCHEMA-CTRL
00949  DCBMSCOM (93)
00950  MRB-EMPMAPLR
00951  MRB-FLDLST
00952  PERFORM IDMS-STATUS.
00953  *
00954  * MAP OUT USING EMPMAPLR
00955  * MESSAGE IS EMP-NOT-FOUND-MESSAGE LENGTH 80.
00956  MOVE 0025 TO DML-SEQUENCE DMLC0025
00957  MOVE 5 TO DCFLG1
00958 MOVE 0 TO DCFLG2
00959 MOVE 0 TO DCFLG3
00960 MOVE 4 TO DCFLG4
00961 CALL 'IDMSCOBI' USING SUBSCHEMA-CTRL
00962 DCBMSCOM (34)
00963 MRB-EMMPAPLR
00964 EMP-NOT-FOUND-MESSAGE DCBMSCOM (80)
00965 PERFORM IDMS-STATUS.
00966 * 00967  * DC RETURN
00968  * NEXT TASK CODE TSK02.
00969 MOVE 0026 TO DML-SEQUENCE DMLC0026
00970 MOVE TSK02 TO DCSTR2
00971 MOVE 128 TO DCFLG1
00972 CALL 'IDMSCOBI' USING SUBSCHEMA-CTRL
00973 DCBMSCOM (19)
00974 PERFORM IDMS-STATUS.
00975 NOT-FOUND-EXIT.
00976 EXIT.
00977 ***************************************************************
00978 IDMS-ABORT.
00979 MOVE ERROR-STATUS TO SSC-ERRSTAT-SAVE.
00980 MOVE DML-SEQUENCE TO SSC-DMLSEQ-SAVE.
00981  * SNAP FROM SUBSCHEMA-LR-CTRL TO SUBSCHEMA-LR-CTRL-END
00982  * ON ANY-STATUS
00983 MOVE 0027 TO DML-SEQUENCE DMLC0027
00984 MOVE 0 TO DCFLG1
00985 CALL 'IDMSCOBI' USING SUBSCHEMA-CTRL
00986 DCBMSCOM (22)
00987 DCSTR1
00988 DCSTR1
00989 DCSTR1
00990 SUBSCHEMA-LR-CTRL SUBSCHEMA-LR-CTRL-END DCBMSCOM (1)
00991 IF NOT ANY-STATUS PERFORM IDMS-STATUS;
00992 ELSE
00993 NEXT SENTENCE.
00994 MOVE SSC-ERRSTAT-SAVE TO ERROR-STATUS.
00995 MOVE SSC-DMLSEQ-SAVE TO DML-SEQUENCE.
00996 IDMS-ABORT-EXIT.
00997 EXIT.
00998 * COPY IDMS IDMS-STATUS.
00999 ***************************************************************01617000
01000 IDMS-STATUS SECTION.01618000
01001 ******************* IDMS-STATUS FOR IDMS-DC *******************01619000
01002 IF DB-STATUS-OK GO TO ISABEX. 01620000
01003 PERFORM IDMS-ABORT. 01621000
01004 MOVE ERROR-STATUS TO SSC-ERRSTAT-SAVE 01622000
01005 MOVE DML-SEQUENCE TO SSC-DMLSEQ-SAVE 01623000
01006  * SNAP FROM SUBSCHEMA-CTRL TO SUBSCHEMA-CTRL-END 01624000
01007  * ON ANY-STATUS 01625000
01008 MOVE 0028 TO DML-SEQUENCE DMLC0028
01009 MOVE 0 TO DCFLG1
01010 CALL 'IDMSCOBI' USING SUBSCHEMA-CTRL
01011 DCBMSCOM (22)
01012 DCSTR1
01013 DCSTR1
01014 DCSTR1
01015 SUBSCHEMA-CTRL SUBSCHEMA-CTRL-END DCBMSCOM (1)
01016 IF NOT ANY-STATUS PERFORM IDMS-STATUS;
01017 ELSE
01018 NEXT SENTENCE.
01019  * ABEND CODE SSC-ERRSTAT-SAVE 01626000
01020  * ON ANY-STATUS 01627000
01021 MOVE 0029 TO DML-SEQUENCE DMLC0029
01022 MOVE SSC-ERRSTAT-SAVE TO DCSTR4
01023 MOVE 2 TO DCFLG1
01024 CALL 'IDMSCOBI' USING SUBSCHEMA-CTRL
01025 DCBMSCOM (1)
01026 IF NOT ANY-STATUS PERFORM IDMS-STATUS;
01027 ELSE
01028 NEXT SENTENCE.
01029 ISABEX. EXIT. 01628000
Call Formats (COBOL)

This section contains the call formats used by CA IDMS to execute DML commands. Each DML function can be coded using standard CALL statements.

The tables in this section are grouped by DB expansions and DC expansions. These tables present the function codes and arguments that are passed to CA IDMS for execution of a DML command.

The following example shows the expanded call format for a BIND RECORD statement (BIND EMPLOYEE):

```
CALL 'IDMS' USING SUBSCHEMA-CTRL
    IDBMSCOM (48)
    SR415
    EMPLOYEE.
```

DB Call Formats (COBOL)

This page lists the following statements:

- Control Statements (see page 359)
- Modification Statements (see page 362)
- Retrieval Statements (see page 363)
- ACCEPT Statements (see page 368)
- LRF DML Statements (see page 369)

Arguments marked with asterisks have default values.
Control Statements

<table>
<thead>
<tr>
<th>Major Function Code</th>
<th>Database Service (in COBOL DML)</th>
<th>(1) IDBMSCOM (nn)</th>
<th>(2)</th>
<th>(3)</th>
<th>(4)</th>
<th>(5)</th>
</tr>
</thead>
<tbody>
<tr>
<td>14</td>
<td>BIND RUN-UNIT</td>
<td>59</td>
<td>IDMS Communications Block*</td>
<td>subschema-name*</td>
<td>subschema-name*</td>
<td>subschema-name*</td>
</tr>
<tr>
<td></td>
<td>BIND RUN-UNIT FOR subschema-name</td>
<td>59</td>
<td>IDMS Communications Block*</td>
<td>subschema-name*</td>
<td>subschema-name*</td>
<td>subschema-name*</td>
</tr>
<tr>
<td></td>
<td>BIND RUN-UNIT NODENAME node-name</td>
<td>59</td>
<td>IDMS Communications Block*</td>
<td>subschema-name*</td>
<td>subschema-name*</td>
<td>subschema-name*</td>
</tr>
<tr>
<td></td>
<td>BIND RUN-UNIT FOR subschema-name NODENAME node-name</td>
<td>59</td>
<td>IDMS Communications Block*</td>
<td>subschema-name*</td>
<td>subschema-name*</td>
<td>subschema-name*</td>
</tr>
<tr>
<td></td>
<td>BIND RUN-UNIT FOR subschema-name DBNAME database-name</td>
<td>59</td>
<td>IDMS Communications Block*</td>
<td>subschema-name*</td>
<td>subschema-name*</td>
<td>subschema-name*</td>
</tr>
<tr>
<td></td>
<td>BIND RUN-UNIT NODENAME node-name DBNAME database-name</td>
<td>59</td>
<td>IDMS Communications Block*</td>
<td>subschema-name*</td>
<td>subschema-name*</td>
<td>subschema-name*</td>
</tr>
<tr>
<td>Operation</td>
<td>Code</td>
<td>Description</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-----------</td>
<td>------</td>
<td>-------------</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BIND RUN-UNIT FOR subschema-name NODENAME node-name</td>
<td>59</td>
<td>IDMS Communications Block* OR subschema-lr-ctrl*</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BIND record-name</td>
<td>48</td>
<td>record-id record-location*</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BIND record-name TO record-name</td>
<td>48</td>
<td>record-id record-location</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BIND record-location WITH record-name</td>
<td>48</td>
<td>record-id record-location</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BIND PROCEDURE FOR procedure-name TO procedure-control-location</td>
<td>73</td>
<td>procedure-name procedure-control-location</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>READY area-name</td>
<td>37</td>
<td>area-name</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>READY area-name USAGE-MODE IS RETRIEVAL</td>
<td>37</td>
<td>area-name</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>READY area-name USAGE-MODE IS PROTECTED RETRIEVAL</td>
<td>39</td>
<td>area-name</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>READY area-name USAGE-MODE IS EXCLUSIVE RETRIEVAL</td>
<td>40</td>
<td>area-name</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>READY area-name USAGE-MODE IS UPDATE</td>
<td>36</td>
<td>area-name</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>READY area-name USAGE-MODE IS PROTECTED UPDATE</td>
<td>38</td>
<td>area-name</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>READY area-name USAGE-MODE IS EXCLUSIVE UPDATE</td>
<td>41</td>
<td>area-name</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>READY USAGE-MODE IS ...</td>
<td></td>
<td>Choose function code from 36-41 as shown above.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>01 FINISH</td>
<td>02</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FINISH TASK</td>
<td>113</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18 COMMIT</td>
<td>66</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>COMMIT ALL</td>
<td>95</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Command</td>
<td>Page</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>------</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>COMMIT TASK</td>
<td>114</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>COMMIT TASK ALL</td>
<td>115</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ROLLBACK</td>
<td>67</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ROLLBACK ALL</td>
<td>96</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ROLLBACK TASK</td>
<td>116</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ROLLBACK TASK CONTINUE</td>
<td>117</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>KEEP CURRENT</td>
<td>87</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>KEEP EXCLUSIVE CURRENT</td>
<td>90</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>KEEP CURRENT <code>record-name</code></td>
<td>89</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>KEEP EXCLUSIVE CURRENT <code>record-name</code></td>
<td>90</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>KEEP CURRENT WITHIN <code>set-name</code></td>
<td>91</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>KEEP EXCLUSIVE CURRENT WITHIN <code>set-name</code></td>
<td>92</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>KEEP CURRENT WITHIN <code>area-name</code></td>
<td>93</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>KEEP EXCLUSIVE CURRENT WITHIN <code>area-name</code></td>
<td>92</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

16 IF `set-name` IS EMPTY ... 64 `set-name`

16 IF `set-name` IS NOT EMPTY ... 65 `set-name`

(Upon return to user run-unit, the status indicator=' 0000' if set is empty, and ' 1601' if not empty.)

16 IF `set-name` MEMBER ... 60 `set-name`

16 IF NOT `set-name` MEMBER ... 62 `set-name`

(Upon return to user run-unit, the status indicator = ' 0000' if the record (current of run unit) is linked into the specified set, and ' 1601' if it is not a member.)
## Modification Statements

<table>
<thead>
<tr>
<th>Major Function Code</th>
<th>Database Service (in COBOL DML)</th>
<th>Calling Arguments (argument 0 contains SUBSCHEMA-CTRL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>STORE record-name</td>
<td>42 record-name</td>
</tr>
<tr>
<td>07</td>
<td>CONNECT record-name TO set-name</td>
<td>44 record-name set-name</td>
</tr>
<tr>
<td>08</td>
<td>MODIFY record-name</td>
<td>35 record-name</td>
</tr>
<tr>
<td>11</td>
<td>DISCONNECT record-name FROM set-name</td>
<td>46 record-name set-name</td>
</tr>
<tr>
<td>02</td>
<td>ERASE record-name MEMBERS</td>
<td>52 record-name</td>
</tr>
<tr>
<td></td>
<td>ERASE record-name PERMANENT MEMBERS</td>
<td>03 record-name</td>
</tr>
<tr>
<td></td>
<td>ERASE record-name SELECTIVE MEMBERS</td>
<td>53 record-name</td>
</tr>
<tr>
<td></td>
<td>ERASE record-name ALL MEMBERS</td>
<td>4 record-name</td>
</tr>
</tbody>
</table>
## Retrieval Statements

<table>
<thead>
<tr>
<th>Major Function Code</th>
<th>Database Service (in COBOL DML)</th>
<th>Calling Arguments (argument 0 contains SUBSCHEMA-CTRL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>03</td>
<td>FIND DB-KEY db-key</td>
<td>(1) IDBMSCOM (nn) 75 db-key</td>
</tr>
<tr>
<td></td>
<td>FIND record-name DB-KEY IS db-key</td>
<td>(2) record-name db-key</td>
</tr>
<tr>
<td></td>
<td>FIND CURRENT</td>
<td>(3) record-name</td>
</tr>
<tr>
<td></td>
<td>FIND CURRENT record-name</td>
<td>(4) record-name</td>
</tr>
<tr>
<td></td>
<td>FIND CURRENT WITHIN set-name</td>
<td>(5) set-name</td>
</tr>
<tr>
<td></td>
<td>FIND CURRENT WITHIN area-name</td>
<td>(6) area-name</td>
</tr>
<tr>
<td></td>
<td>FIND NEXT WITHIN set-name</td>
<td>(7) set-name</td>
</tr>
<tr>
<td></td>
<td>FIND NEXT record-name WITHIN set-name</td>
<td>(8) record-name set-name</td>
</tr>
<tr>
<td></td>
<td>FIND PRIOR WITHIN set-name</td>
<td>(9) set-name</td>
</tr>
<tr>
<td></td>
<td>FIND PRIOR record-name WITHIN set-name</td>
<td>(10) record-name set-name</td>
</tr>
<tr>
<td></td>
<td>FIND FIRST WITHIN set-name</td>
<td>(11) set-name</td>
</tr>
<tr>
<td></td>
<td>FIND FIRST record-name WITHIN set-name</td>
<td>(12) record-name set-name</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(13) set-name</td>
</tr>
<tr>
<td>FIND LAST WITHIN set-name</td>
<td>24</td>
<td>reco set-name</td>
</tr>
<tr>
<td>----------------------------</td>
<td>----</td>
<td>---------------</td>
</tr>
<tr>
<td>FIND LAST record-name WITHIN set-name</td>
<td>22</td>
<td>reco set-name</td>
</tr>
<tr>
<td>FIND number WITHIN set-name</td>
<td>78</td>
<td>set-number</td>
</tr>
<tr>
<td>FIND number record-name WITHIN set-name</td>
<td>76</td>
<td>reco set-number</td>
</tr>
<tr>
<td>FIND NEXT WITHIN area-name</td>
<td>15</td>
<td>area-name</td>
</tr>
<tr>
<td>FIND NEXT record-name WITHIN area-name</td>
<td>11</td>
<td>reco area-name</td>
</tr>
<tr>
<td>FIND PRIOR WITHIN area-name</td>
<td>17</td>
<td>area-name</td>
</tr>
<tr>
<td>FIND PRIOR record-name WITHIN area-name</td>
<td>13</td>
<td>reco area-name</td>
</tr>
<tr>
<td>FIND FIRST WITHIN area-name</td>
<td>21</td>
<td>area-name</td>
</tr>
<tr>
<td>FIND FIRST record-name WITHIN area-name</td>
<td>19</td>
<td>reco area-name</td>
</tr>
<tr>
<td>FIND LAST WITHIN area-name</td>
<td>25</td>
<td>area-name</td>
</tr>
<tr>
<td>FIND LAST record-name WITHIN area-name</td>
<td>23</td>
<td>reco set-name</td>
</tr>
<tr>
<td>Record Name</td>
<td>Argument Count</td>
<td></td>
</tr>
<tr>
<td>-------------</td>
<td>----------------</td>
<td></td>
</tr>
<tr>
<td>FIND number WITHIN area-name</td>
<td>79</td>
<td></td>
</tr>
<tr>
<td>FIND number record-name WITHIN area-name</td>
<td>77</td>
<td></td>
</tr>
<tr>
<td>FIND OWNER WITHIN set-name</td>
<td>31</td>
<td></td>
</tr>
<tr>
<td>FIND CALC (ANY) record-name</td>
<td>32</td>
<td></td>
</tr>
<tr>
<td>FIND DUPLICATE record-name</td>
<td>50</td>
<td></td>
</tr>
<tr>
<td>FIND record-name WITHIN set-name USING sort-key</td>
<td>33</td>
<td></td>
</tr>
<tr>
<td>FIND record-name WITHIN set-name CURRENT USING sort-key</td>
<td>51</td>
<td></td>
</tr>
</tbody>
</table>

**OBTAIN**

Can use any of the above FIND record selection expressions.

Call generated consists of arguments described above for the FIND in question plus an additional argument of IDBMSCOM (43) function.

For example:

<table>
<thead>
<tr>
<th>Record Name</th>
<th>Argument Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>OBTAIN CALC record-name</td>
<td>32</td>
</tr>
<tr>
<td>OBTAIN PRIOR record-name WITHIN set-name</td>
<td>12</td>
</tr>
</tbody>
</table>
KEEP

KEEP EXCLUSIVE

Can use any of the above FIND/OBTAIN record selection expressions.

Call generated consists of arguments described above for the FIND/OBTAIN in question plus one of the following additional IDBMSCOM functions:

KEEP.....................IDBMSCOM(87)
KEEP EXCLUSIVE.........IDBMSCOM(**)

For example:

<table>
<thead>
<tr>
<th>OBTAINT KEEP CALC record-name</th>
<th>32</th>
<th>IDB MSC OM (43) (87)</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIND KEEP EXCLUSIVE CURRENT</td>
<td>30</td>
<td>IDB MSC OM (88)</td>
</tr>
<tr>
<td>GET record-name</td>
<td>34</td>
<td>inde db-key symbol -key</td>
</tr>
<tr>
<td>17 RETURN db-key FROM index-set-name CURRENCY KEY INTO symbolic-key</td>
<td>81</td>
<td>inde db-key symbol-key</td>
</tr>
<tr>
<td>82 RETURN db-key FROM index-set-name FIRST KEY INTO symbolic-key</td>
<td>82</td>
<td>inde db-key symbol-key</td>
</tr>
<tr>
<td>83</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>


RETURN db-key FROM index-set-name LAST KEY INTO symbolic-key

RETURN db-key FROM index-set-name NEXT KEY INTO symbolic-key 84

RETURN db-key FROM index-set-name PRIOR KEY INTO symbolic-key 85

RETURN db-key FROM index-set-name USING index-key-value KEY INTO symbolic-key 86
# ACCEPT Statements

<table>
<thead>
<tr>
<th>Major Function Code</th>
<th>Database Service (in COBOL DML)</th>
<th>(1) IDBMSCOM (nn)</th>
<th>(2) (3) (4) (5)</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>ACCEPT db-key FROM CURRENCY</td>
<td>54</td>
<td>db-key</td>
</tr>
<tr>
<td></td>
<td>ACCEPT db-key FROM record-name CURRENCY</td>
<td>55</td>
<td>record-name db-key</td>
</tr>
<tr>
<td></td>
<td>ACCEPT db-key FROM set-name CURRENCY</td>
<td>57</td>
<td>set-name db-key</td>
</tr>
<tr>
<td></td>
<td>ACCEPT db-key FROM area-name CURRENCY</td>
<td>56</td>
<td>area-name db-key</td>
</tr>
<tr>
<td></td>
<td>ACCEPT db-key FROM set-name NEXT CURRENCY</td>
<td>68</td>
<td>set-name db-key</td>
</tr>
<tr>
<td></td>
<td>ACCEPT db-key FROM set-name PRIOR CURRENCY</td>
<td>69</td>
<td>set-name db-key</td>
</tr>
<tr>
<td></td>
<td>ACCEPT db-key FROM set-name OWNER CURRENCY</td>
<td>70</td>
<td>set-name db-key</td>
</tr>
<tr>
<td></td>
<td>ACCEPT db-statistics FROM IDMS STATISTICS</td>
<td>71</td>
<td>db-statistics</td>
</tr>
<tr>
<td></td>
<td>ACCEPT bind-address FROM record-name BIND</td>
<td>72</td>
<td>record-name bind-address</td>
</tr>
<tr>
<td></td>
<td>ACCEPT procedure-control-location FROM procedure-name PROCEDURE</td>
<td>74</td>
<td>procedure-control-location</td>
</tr>
</tbody>
</table>
# LRF DML Statements

<table>
<thead>
<tr>
<th>Major Function Code</th>
<th>Database Service (in COBOL DML)</th>
<th>(1) IDBMSCOM (nn)</th>
<th>(2)</th>
<th>(3)</th>
<th>(4)</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>OBTAIN FIRST logical-record-name</td>
<td>99</td>
<td>subschema logical-record-loc*</td>
<td>subschema logical-record-loc*</td>
<td>subschema logical-record-loc*</td>
</tr>
<tr>
<td></td>
<td>OBTAIN FIRST logical-record-name INTO alt-logical-record-location</td>
<td>99</td>
<td>subschema logical-record-loc*</td>
<td>subschema logical-record-loc*</td>
<td>subschema logical-record-loc*</td>
</tr>
<tr>
<td></td>
<td>OBTAIN NEXT logical-record-name</td>
<td>99</td>
<td>subschema logical-record-loc*</td>
<td>subschema logical-record-loc*</td>
<td>subschema logical-record-loc*</td>
</tr>
<tr>
<td></td>
<td>OBTAIN NEXT logical-record-name INTO alt-logical-record-location</td>
<td>99</td>
<td>subschema logical-record-loc*</td>
<td>subschema logical-record-loc*</td>
<td>subschema logical-record-loc*</td>
</tr>
<tr>
<td></td>
<td>MODIFY logical-record-name</td>
<td>99</td>
<td>subschema logical-record-loc*</td>
<td>subschema logical-record-loc*</td>
<td>subschema logical-record-loc*</td>
</tr>
<tr>
<td></td>
<td>MODIFY logical-record-name FROM alt-logical-record-location</td>
<td>99</td>
<td>subschema logical-record-loc*</td>
<td>subschema logical-record-loc*</td>
<td>subschema logical-record-loc*</td>
</tr>
<tr>
<td></td>
<td>STORE logical-record-name</td>
<td>99</td>
<td>subschema logical-record-loc*</td>
<td>subschema logical-record-loc*</td>
<td>subschema logical-record-loc*</td>
</tr>
<tr>
<td></td>
<td>STORE logical-record-name FROM alt-logical-record-location</td>
<td>99</td>
<td>subschema logical-record-loc*</td>
<td>subschema logical-record-loc*</td>
<td>subschema logical-record-loc*</td>
</tr>
<tr>
<td></td>
<td>ERASE logical-record-name</td>
<td>99</td>
<td>subschema logical-record-loc*</td>
<td>subschema logical-record-loc*</td>
<td>subschema logical-record-loc*</td>
</tr>
<tr>
<td></td>
<td>ERASE logical-record-name FROM alt-logical-record-location</td>
<td>99</td>
<td>subschema logical-record-loc*</td>
<td>subschema logical-record-loc*</td>
<td>subschema logical-record-loc*</td>
</tr>
</tbody>
</table>

To differentiate between the LRF DML statements, the DML compiler places the name of the verb issued into the LRC Block (subschema-lr-ctrl).

---

## DC Call Formats (COBOL)

This page contains the following statements:

- Program Management Statements (see page 370)
- Storage Management Statements (see page 370)
- Task Management Statements (see page 371)
- Time Management Statements (see page 371)
- Scratch Management Statements (see page 372)
Program Management Statements

<table>
<thead>
<tr>
<th>Major Function Code</th>
<th>DC System Service (in COBOL DML)</th>
<th>Calling Arguments (argument 0 contains SUBSCHEMA-CTRL)</th>
<th>(1) DCBMSCOM (nn)</th>
<th>(2)</th>
<th>(3)</th>
<th>(4)</th>
<th>(5)</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>TRANSFER CONTROL</td>
<td>DCFLG1 DCSTR2 param</td>
<td>23</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>DC RETURN</td>
<td></td>
<td>19</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>34</td>
<td>LOAD TABLE</td>
<td>01-level-program-location</td>
<td>15</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>34</td>
<td>DELETE TABLE</td>
<td>01-level-program-location</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>33</td>
<td>SET ABEND EXIT (STATE)</td>
<td></td>
<td>20</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>33</td>
<td>ABEND</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Storage Management Statements

<table>
<thead>
<tr>
<th>Major Function Code</th>
<th>DC System Service (in COBOL DML)</th>
<th>Calling Arguments (argument 0 contains SUBSCHEMA-CTRL)</th>
<th>(1) DCBMSCOM (nn)</th>
<th>(2)</th>
<th>(3)</th>
<th>(4)</th>
<th>(5)</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>GET STORAGE</td>
<td>01-level-storage-data-location end-storage-data-location</td>
<td>13</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>FREE STORAGE</td>
<td>01-level-storage-data-location start-free-storage-location</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Task Management Statements

<table>
<thead>
<tr>
<th>Major Function Code</th>
<th>DC System Service (in COBOL DML)</th>
<th>Calling Arguments (argument 0 contains SUBSCHEMA-CTRL)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>(1) DCBMSCOM ((nn)) (2) (3) (4) (5)</td>
</tr>
<tr>
<td>37</td>
<td>ATTACH</td>
<td>3 DCFL DCSTR2 G1 parameter</td>
</tr>
<tr>
<td>37</td>
<td>CHANGE PRIORITY</td>
<td>4</td>
</tr>
<tr>
<td>39</td>
<td>ENQUEUE</td>
<td>9 DCFL DCBMSCOM (1 resource-mode DCBMSCOM(l resource-length resource-id..) G1)</td>
</tr>
<tr>
<td>39</td>
<td>DEQUEUE</td>
<td>8 DCFL DCBMSCOM (1 resource-id.. G1 length)</td>
</tr>
<tr>
<td>31</td>
<td>WAIT</td>
<td>24 ecb</td>
</tr>
<tr>
<td>31</td>
<td>POST</td>
<td>16 ecb</td>
</tr>
</tbody>
</table>

### Time Management Statements

<table>
<thead>
<tr>
<th>Major Function Code</th>
<th>DC System Service (in COBOL DML)</th>
<th>Calling Arguments (argument 0 contains SUBSCHEMA-CTRL)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>(1) DCBMSCOM (nn) (2) (3) (4) (5)</td>
</tr>
<tr>
<td>35</td>
<td>GET TIME</td>
<td>14 return-time return-date</td>
</tr>
<tr>
<td>35</td>
<td>SET TIMER</td>
<td>21 task-data-location end-task-data-location-location</td>
</tr>
<tr>
<td>35</td>
<td>SET TIMER (post)</td>
<td>21 post-ecb</td>
</tr>
</tbody>
</table>
## Scratch Management Statements

<table>
<thead>
<tr>
<th>Major Function Code</th>
<th>DC System Service (in COBOL DML)</th>
<th>Calling Arguments (argument 0 contains SUBSCHEMA-CTRL)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>(1) DCBMSCOM (nn)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(2) scratch-data-location end-scratch-data-location</td>
</tr>
<tr>
<td>43</td>
<td>PUT SCRATCH</td>
<td>18</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>43</td>
<td>GET SCRATCH</td>
<td>12 return-scratch-data-location end-scratch-data-location</td>
</tr>
<tr>
<td>43</td>
<td>DELETE SCRATCH</td>
<td>6 post-ecb</td>
</tr>
</tbody>
</table>

## Queue Management Statements

<table>
<thead>
<tr>
<th>Major Function Code</th>
<th>DC System Service (in COBOL DML)</th>
<th>Calling Arguments (argument 0 contains SUBSCHEMA-CTRL)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>(1) DCBMSCOM (nn)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(2) queue-data-location end-queue-data-location</td>
</tr>
<tr>
<td>44</td>
<td>PUT QUEUE</td>
<td>17</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>44</td>
<td>GET QUEUE</td>
<td>11 return-queue-data-location end-queue-data-location</td>
</tr>
<tr>
<td>44</td>
<td>DELETE QUEUE</td>
<td>6</td>
</tr>
</tbody>
</table>
### Terminal Management Statements

<table>
<thead>
<tr>
<th>Major Function Code</th>
<th>DC System Service (in COBOL DML)</th>
<th>Calling Arguments (argument 0 contains SUBSCHEMA-CTRL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>45</td>
<td>READ TERMINAL</td>
<td>(1) DCBMSCOM (nn) input-data-location end-input-data-location</td>
</tr>
<tr>
<td>45</td>
<td>WRITE TERMINAL</td>
<td>(1) DCBMSCOM (nn) output-data-location end-output-data-location</td>
</tr>
<tr>
<td>45</td>
<td>WRITE THEN READ TERMINAL</td>
<td>(1) DCBMSCOM (nn) output-data-location end-output-data-location input-data-location end-input-data-location</td>
</tr>
<tr>
<td>45</td>
<td>CHECK TERMINAL</td>
<td>(1) DCBMSCOM (nn) input-data-location end-input-data-location</td>
</tr>
<tr>
<td>47</td>
<td>READ LINE FROM TERMINAL</td>
<td>(1) DCBMSCOM (nn) input-data-location end-input-data-location</td>
</tr>
<tr>
<td>47</td>
<td>WRITE LINE TO TERMINAL</td>
<td>(1) DCBMSCOM (nn) output-data-location end-output-data-location</td>
</tr>
<tr>
<td>47</td>
<td>END LINE TERMINAL SESSION</td>
<td>(1) DCBMSCOM (nn) message-end-message-location</td>
</tr>
<tr>
<td>48</td>
<td>WRITE PRINTER</td>
<td>(1) DCBMSCOM (nn) message-end-message-location</td>
</tr>
<tr>
<td>46</td>
<td>MAP IN (IO)</td>
<td>(1) DCBMSCOM (nn) MRB-mapname</td>
</tr>
<tr>
<td>46</td>
<td>MAP IN (NOIO)</td>
<td>(1) DCBMSCOM (nn) MRB-mapname mapped-data-location end-mapped-data-location</td>
</tr>
<tr>
<td>46</td>
<td>MAP IN (paging) (a)</td>
<td>(1) DCBMSCOM (nn) MRB-mapname data-field-name sequence-field-page-number</td>
</tr>
<tr>
<td>46</td>
<td>MAP IN (paging) (b)</td>
<td>(1) DCBMSCOM (nn) MRB-mapname key page-number</td>
</tr>
<tr>
<td>46</td>
<td>MAP OUT (IO)</td>
<td>(1) DCBMSCOM (nn) MRB-mapname message-text end-message-data-location OR DCBMSCOM (length) mapped-data-location end-mapped-data-location</td>
</tr>
<tr>
<td>46</td>
<td>MAP OUT (NOIO)</td>
<td>(1) DCBMSCOM (nn) MRB-mapname mapped-data-location end-mapped-data-location</td>
</tr>
<tr>
<td>Page</td>
<td>Command</td>
<td>Length</td>
</tr>
<tr>
<td>------</td>
<td>---------</td>
<td>--------</td>
</tr>
<tr>
<td>46</td>
<td>MAP OUT (paging)</td>
<td>34</td>
</tr>
<tr>
<td>46</td>
<td>MAP OUTIN</td>
<td>34</td>
</tr>
<tr>
<td>46</td>
<td>MODIFY MAP</td>
<td>93</td>
</tr>
<tr>
<td>46</td>
<td>INQUIRE MAP (a)</td>
<td>92</td>
</tr>
<tr>
<td>46</td>
<td>INQUIRE MAP (b)</td>
<td>92</td>
</tr>
<tr>
<td>46</td>
<td>INQUIRE MAP (c)</td>
<td>92</td>
</tr>
<tr>
<td>46</td>
<td>INQUIRE MAP (d)</td>
<td>92</td>
</tr>
<tr>
<td>46</td>
<td>STARTPAGE</td>
<td>40</td>
</tr>
<tr>
<td>46</td>
<td>ENDPAGE</td>
<td>41</td>
</tr>
</tbody>
</table>
## Utility Statements

<table>
<thead>
<tr>
<th>Major Function Code</th>
<th>DC System Service (in COBOL DML)</th>
<th>(1) DCBMSCOM (nn)</th>
<th>(2)</th>
<th>(3)</th>
<th>(4)</th>
<th>(5)</th>
</tr>
</thead>
<tbody>
<tr>
<td>48</td>
<td>ACCEPT</td>
<td>2</td>
<td>return-location</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>SNAP</td>
<td>22</td>
<td>DCSTR1 (6) begin-dump-location</td>
<td>DCSTR1 title</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>end-dump-location</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DCBMSCOM</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>49</td>
<td>SEND MESSAGE</td>
<td>38</td>
<td>user-id message-location</td>
<td>end-message-location</td>
<td></td>
<td></td>
</tr>
<tr>
<td>38</td>
<td>BIND TRANSACTION STATISTICS</td>
<td>28</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>38</td>
<td>ACCEPT TRANSACTION STATISTICS</td>
<td>28</td>
<td>return-stat-data-location</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>38</td>
<td>END TRANSACTION STATISTICS</td>
<td>28</td>
<td>return-stat-data-location</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>51</td>
<td>KEEP LONGTERM</td>
<td>29</td>
<td>record-name set-name area-name</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>36</td>
<td>WRITE LOG</td>
<td>25</td>
<td>text-return-location end-text-return-location reply-location end-reply-location</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(6)</td>
<td>(7)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>parameter-end-parameter-location</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Recovery Statements

<table>
<thead>
<tr>
<th>Major Function Code</th>
<th>DC System Service (in COBOL DML)</th>
<th>Calling Arguments (argument 0 contains SUBSCHEMA-CTRL)</th>
<th>(1) DCBMSCOM ( (2) nn)</th>
<th>(3)</th>
<th>(4) (5)</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>COMMIT</td>
<td></td>
<td>66</td>
<td></td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>COMMIT TASK</td>
<td></td>
<td>27</td>
<td></td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>FINISH</td>
<td></td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>FINISH TASK</td>
<td></td>
<td>27</td>
<td></td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>ROLLBACK</td>
<td></td>
<td>67</td>
<td></td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>ROLLBACK TASK</td>
<td></td>
<td>27</td>
<td></td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>WRITE JOURNAL</td>
<td></td>
<td>26 record-location</td>
<td></td>
<td>end-record-location</td>
</tr>
</tbody>
</table>

### DC-BATCH Statements

<table>
<thead>
<tr>
<th>Major Function Code</th>
<th>DC System Service (in COBOL DML)</th>
<th>Calling Arguments (argument 0 contains SUBSCHEMA-CTRL)</th>
<th>(1) DCBMSCOM (nn)</th>
<th>(2)</th>
<th>(3)</th>
<th>(4)</th>
<th>(5)</th>
</tr>
</thead>
<tbody>
<tr>
<td>14</td>
<td>BIND-TASK</td>
<td></td>
<td>28</td>
<td>DCSTR2</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
CA IDMS Keywords

This section contains a list of keywords recognized by the DML precompiler, including words applicable in the online environment only. All keywords marked with an asterisk are also reserved words. Reserved words cannot be used for user-defined element, record, set, paragraph, or area variable names.

⚠️ Note: The method of parsing used by the IDMSDMLC preprocessor is significantly different in CA IDMS Release 12.0 and later releases from that used in prior releases. The current parsing method looks at individual words in the source code. If it encounters a keyword, it assumes that the keyword should be expanded and tries to do so. Invalid use of reserved words can thus result in either coding errors or syntax errors. For example, if you use FIND as a variable, the parser will try to handle it as the DML verb FIND.

List of Keywords

| *ABEND | INTERNAL | *REMARKS | REPLACE |
| *ABORT | INTERVAL | REPORT | |
| *ACCEPT | INTO | REQUIRED | |
| AID | INVOKED | REREAD | |
| ALARM | IO | REREDD | |
| ALL | IS | RESETKBD | |
| ALPHAMERIC | JOURNAL | RESETMDT | |
| ALWAYS | JUSTIFY | RESUME | |
| ANY | *KEEP | RETENTION | |
| AREA | KEY | RETRY | |
| ASSIGN | LAST | RETURN | |
| AT | LEAVE | RETRIEVAL | |
| *ATTACH | LEFT | RETRY | |
| ATTRIBUTES | LENGTH | *RETURN | |
| BACKPAGE | LEVELS | REVERSE-VIDEO | |
| BACKSCAN | LINE | REVERSED | |
| *BIND | LINK | REWIND | |
| BLINK | *LINKAGE | RIGHT | |
| BLUE | LIST | *ROLLBACK | |
| BRIGHT | LITERALS | RUN | |
| BROWSE | *LOAD | RUN-UNIT | |
| BUFFER | LOCK | *SCHEMA | |
| BUT | LOG | SCRATCH | |
| BY | LONG | SCREEN | |
| CALC | LONGTERM | SCREENSIZE | |
| *CALL | LR | SECONDS | |
| CANCEL | LSSC-NODN | *SECTION | |
| *CHANGE | LTERM | *SELECT | |
| CHANGED | MANUAL | SELECTIVE | |
| *CHECK | *MAP | SEND | |
| CLASS | MAP-BINDS | SEQUENCE | |
| CLEAR | MAP-CONTROL | SEQUENCE-NUMBER | |
| CODE | MAP-CONTROLS | SESSION | |
| *COMMIT | MAP-RECORDS | *SET | |
| COMP | MAPS | *SHARE | |
| COMP-3 | MAX | SHARED | |
| *CONNECT | MDT | SHORT | |
| CONTENTS | MEMBER | SKIP | |
IDMS-RECORDS PTERM XCTL
IDMS-STATISTICS *PUT YELLOW
*IF QUEUE YES
IGNORED *READ 40CR
IN *READY 64CR
INCREMENTED RECORD 80CR
INPUT RED
*INQUIRE REDISPATCH
INTENT RELEASE
Notes to Teleprocessing Monitor Users

Contents

- TP Monitor Coding Guidelines (see page 380)
- TP monitor Coding Requirements (see page 381)

This section describes special considerations relating to application programs running under teleprocessing (TP) monitors supported by CA IDMS (that is, CICS, INTERCOMM, SHADOW, TASK /MASTER, UTM, and WESTI).

TP Monitor Coding Guidelines

While there are no special coding requirements for TP monitor transactions, the following section should be adhered to:

- DML statements should be coded such that all database requests (for example, BIND, READY, OBTAIN, FINISH) are executed together whenever possible to achieve maximum efficiency and ease of recovery.

- For each TP monitor, you should check with the DBA to determine the operating mode (protocol) installed. The proper mode must then be specified in the IDMS-CONTROL SECTION of the ENVIRONMENT DIVISION.

- For CICS, INTERCOMM, SHADOW, UTM, and WESTI applications, the mode as installed may require the inclusion of additional statements in the IDMS-CONTROL SECTION, WORKING-STORAGE SECTION, and LINKAGE SECTION of each program. These requirements and the applicable modes are outlined in the following table.

  Note: The same rules apply to the COPY IDMS statements used to insert logical-record source code components into the program: IDMS-RECORDS MANUAL should be coded in the ENVIRONMENT DIVISION; SUBSCHEMA-LR-NAMES should be copied into the WORKING-STORAGE SECTION; and SUBSCHEMA-CTRL, SUBSCHEMA-LR-CTRL, and SUBSCHEMA-LR-RECORDS should be copied into the LINKAGE-SECTION (except under CICS-EXEC or CICS-EXEC-AUTO, when all required components should be copied into the WORKING-STORAGE SECTION).

- The DML compiler should be executed before the teleprocessing monitor preprocessor.
# TP monitor Coding Requirements

<table>
<thead>
<tr>
<th>TP MONITOR IS</th>
<th>IF MODE</th>
<th>IDMS-CONTROL SECTION</th>
<th>WORKING-STORAGE SECTION</th>
<th>LINKAGE SECTION</th>
<th>PROCEDURE DIVISION</th>
</tr>
</thead>
<tbody>
<tr>
<td>CICS</td>
<td>CICS</td>
<td>IDMS-RECORDS MANUAL.</td>
<td>COPY IDMS SUBSCHEMA- NAMES</td>
<td>*01 TWA 03 FILLER PIC S9 (8) COMP SYNC. 03 COPY IDMS SUBSCHEMA-CTRL. 03 COPY IDMS SUBSCHEMA-RECORDS. OR **COPY IDMS SUBSCHEMA-CTRL COPY IDMS SUBSCHEMA-RECORDS. (A CICS GETMAIN must be issued for the SUBSCHEMA-CTRL and for each record being copied.)</td>
<td>COPY IDMS IDMS-WAIT.</td>
</tr>
<tr>
<td>CICS</td>
<td>EXEC</td>
<td>IDMS-RECORDS MANUAL.</td>
<td>COPY IDMS SUBSCHEMA- CTRL.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CICS</td>
<td>EXEC-AUTO</td>
<td>IDMS-RECORDS MANUAL.</td>
<td>COPY IDMS SUBSCHEMA- NAMES.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CICS</td>
<td>EXEC-AUTO</td>
<td>IDMS-RECORDS MANUAL.</td>
<td>COPY IDMS SUBSCHEMA- NAMES.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>INTERCO MM</td>
<td>INTERCO MM</td>
<td>IDMS-RECORDS MANUAL.</td>
<td>COPY IDMS SUBSCHEMA- NAMES.</td>
<td>COPY IDMS SUBSCHEMA-CTRL. COPY IDMS SUBSCHEMA-RECORDS.</td>
<td>COPY IDMS SUBSCHEMA-RECORDS.</td>
</tr>
<tr>
<td>SHADOW</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TP MONITOR IS</td>
<td>IF MODE CONTROL SECTION</td>
<td>IDMS-RECORDS MANUAL. TUS</td>
<td>WORKING-STORAGE SECTION</td>
<td>Linkage Section</td>
<td>PROCEDURE DIVISION</td>
</tr>
<tr>
<td>---------------</td>
<td>--------------------------</td>
<td>---------------------------</td>
<td>--------------------------</td>
<td>-----------------</td>
<td>-------------------</td>
</tr>
<tr>
<td>SHADOW</td>
<td>COPY IDMS SUBSCHEMA-RECORDS</td>
<td>COPY IDMS SUBSCHEMA-NAMES.</td>
<td>COPY IDMS SUBSCHEMA-RECORDS</td>
<td>MOVE LOW-VALUES to SUBSCHEMA_CTRL before each BIND RUN-UNIT.</td>
<td></td>
</tr>
<tr>
<td>UTM</td>
<td>COPY KCKBC. 05 X PIC S9 (8) COMP SYNC. 05 COPY IDMS SUBSCHEMA-CTRL. 05 COPY IDMS SUBSCHEMA-RECORDS. COPY KCPAC.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WESTI</td>
<td>COPY IDMS SUBSCHEMA-RECORDS MANUAL.</td>
<td>COPY IDMS SUBSCHEMA-NAMES.</td>
<td>COPY IDMS SUBSCHEMA-RECORDS.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*If SUBSCHEMA-CTRL, SUBSCHEMA_RECORDS, and additional data does not exceed 4,096 bytes.*

**If SUBSCHEMA-CTRL, SUBSCHEMA_RECORDS, and additional data exceeds 4,096 bytes.*
EMPLOYEE Database Definition

This section contains the IDMSRPTS utility and the data structure diagram for the EMPLOYEE
database from which most of the examples in this manual are taken. Both of the sample programs
listed earlier in this manual access this database.

**Note:** For more information about the IDMSRPTS utility, see Utilities ([https://docops.ca.com/display
/IDMSCU19/Utilities](https://docops.ca.com/display/IDMSCU19/Utilities)).

The following topics are discussed on this page:

- IDMSRPTS Utility Report Listings (see page 383)
- EMPLOYEE Database Structure Diagram (see page 390)

### IDMSRPTS Utility Report Listings

```
IDMSRPTS nn.
n SCHEMA RECORD DESCRIPTION LISTING DATE TIME PAGE
/yy hhmms 1

SCHEMA EMPSCHM VERSION 100

RECORD NAME........ COVERAGE RLGTH= 36
RECORD VERSION..... 0100 DLGTH= 20
RECORD ID........... 0400 KLGTH= 16
RECORD LENGTH...... FIXED DSTRT= 16
LOCATION MODE...... VIA SET EMP-COVERAGE DISPLACEMENT 0000 PAGES
WITHIN............. INS-DEMO-REGION OFFSET 5 PGS FOR 20 PGS
DBKEY POSITIONS.... SET............... TYPE........ NEXT PRIOR OWNER
EMP-COVERAGE MEMBER 1 2 3
COVERAGE-CLAIMS OWNER 4
DATA ITEM.......... REDEFINES... USAGE....... VALUE..... PICTURE. STRT LGTH
02 SELECTION-DATE-0400 DISPLAY 1 8
03 SELECTION-YEAR-0400 DISPLAY 9(4) 1 4
03 SELECTION-MONTH-0400 DISPLAY 9(2) 5 2
03 SELECTION-DAY-0400 DISPLAY 9(2) 7 2
02 TERMINATION-DATE-0400 DISPLAY 9 8
03 TERMINATION-YEAR-0400 DISPLAY 9(4) 9 4
03 TERMINATION-MONTH-0400 DISPLAY 9(2) 13 2
03 TERMINATION-DAY-0400 DISPLAY 9(2) 15 2
02 TYPE-0400 DISPLAY X 17 1
88 MASTER-0400 COND 'M' 17
88 FAMILY-0400 COND 'F' 17
88 DEPENDENT-0400 COND 'D' 17
02 INS-PLAN-CODE-0400 DISPLAY X(3) 18 3
88 GROUP-LIFE-0400 COND '001' 18
88 HMO-0400 COND '002' 18
88 GROUP-HEALTH-0400 COND '003' 18
88 GROUP-DENTAL-0400 COND '004' 18
```

### EMPLOYEE Database Structure Diagram

```
IDMSRPTS nn.
n SCHEMA RECORD DESCRIPTION LISTING DATE TIME PAGE
/yy hhmms 3

SCHEMA EMPSCHM VERSION 100

RECORD NAME........ DENTAL-CLAIM RLGTH= 944
```
RECORD VERSION..... 0100
RECORD ID........ 0405
RECORD LENGTH..... VARIABLE
MINIMUM ROOT....... 132 CHARACTERS
MINIMUM FRAGMENT... 932 CHARACTERS
LOCATION MODE..... VIA SET COVERAGE-CLAIMS
WITHIN............. INS-DEMO-REGION
DBKEY POSITIONS.... SET

DATA ITEM........ REDEFINES... USAGE....... VALUE..... PICTURE. STRT LGTH
02 CLAIM-DATE-0405 DISPLAY 1 8
03 CLAIM-YEAR-0405 DISPLAY 9(4) 1 4
03 CLAIM-MONTH-0405 DISPLAY 9(2) 5 2
03 CLAIM-DAY-0405 DISPLAY 9(2) 7 2
02 PATIENT-NAME-0405 DISPLAY 9 25
03 PATIENT-FIRST-NAME-0405 DISPLAY X(10) 9 10
03 PATIENT-LAST-NAME-0405 DISPLAY X(15) 19 15
02 PATIENT-BIRTH-DATE-0405 DISPLAY 34 8
03 PATIENT-BIRTH-YEAR-0405 DISPLAY 9(4) 34 4
03 PATIENT-BIRTH-MONTH-0405 DISPLAY 9(2) 38 2
03 PATIENT-BIRTH-DAY-0405 DISPLAY 9(2) 40 2
02 PATIENT-SEX-0405 DISPLAY X 42 1
02 RELATION-TO-EMPLOYEE-0405 DISPLAY X(10) 43 10
02 DENTIST-NAME-0405 DISPLAY 53 25
03 DENTIST-FIRST-NAME-0405 DISPLAY X(10) 53 10
03 DENTIST-LAST-NAME-0405 DISPLAY X(15) 63 15
02 DENTIST-ADDRESS-0405 DISPLAY 78 46
03 DENTIST-STREET-0405 DISPLAY X(20) 78 20
03 DENTIST-CITY-0405 DISPLAY X(15) 98 15
03 DENTIST-STATE-0405 DISPLAY X(2) 113 2
03 DENTIST-ZIP-0405 DISPLAY 115 9
04 DENTIST-ZIP-FIRST-FIVE-0405 DISPLAY X(5) 115 5
04 DENTIST-ZIP-LAST-FOUR-0405 DISPLAY X(4) 120 4
02 DENTIST-LICENSE-NUMBER-0405 DISPLAY 124 6
02 NUMBER-OF-PROCEDURES-0405 DISPLAY 130 2
02 FILLER DISPLAY X 132 1
02 DENTIST-CHARGES-0405 DISPLAY OCCURS 0 TO 10 133 800
03 TOOTH-NUMBER-0405 DISPLAY 9(2) 1 2
03 TOOTH-DATE-0405 DISPLAY 9(2) 3 2
03 SERVICE-YEAR-0405 DISPLAY 9(4) 3 4
04 SERVICE-MONTH-0405 DISPLAY 9(2) 7 2
04 SERVICE-DAY-0405 DISPLAY 9(2) 9 2
03 PROCEDURE-CODE-0405 DISPLAY 9(4) 11 4
03 DESCRIPTION-OF-SERVICE-0405 DISPLAY X(60) 15 60
03 FEE-0405 COMP-3 9(7)V99 75 5
03 FILLER DISPLAY X 80 1

RECORD NAME........ DEPARTMENT
RECORD VERSION..... 0100
RECORD ID........ 0410
RECORD LENGTH..... FIXED
LOCATION MODE..... CALC USING DEPT-ID-0410
WITHIN............. ORG-DEMO-REGION
DBKEY POSITIONS.... SET

DATA ITEM........ REDEFINES... USAGE....... VALUE..... PICTURE. STRT LGTH
02 DEPT-ID-0410 DISPLAY 9(4) 1 4
02 DEPT-NAME-0410 DISPLAY X(45) 5 45
02 DEPT-HEAD-ID-0410 DISPLAY 9(4) 50 4
02 FILLER DISPLAY XXX 54 3

IDMSRPTS nn.

RECORDS

SCHEMA RECORD DESCRIPTION LISTING DATE TIME PAGE
RECDES DICTIONARY APPLDICT OF NODE DEFAULT mm/dd

/yy hhmms 6

SCHEMA EMPSCHM VERSION 100

IDMSRPTS nn.
RECD   DICTIONARY APPLDICT OF NODE DEFAULT

SCHEMA EMPSCHM VERSION 100

RECORD NAME........ EMPLOYEE            RLGTH= 192
RECORD VERSION..... 0100              DLGTH= 120
RECORD ID.......... 0415              KLGTH=  72
RECORD LENGTH...... FIXED              DSTRT=  72
LOCATION MODE...... CALC USING EMP-ID-0415  Duplicates not allowed
WITHIN............. EMP-DEMO-REGION OFFSET 5 PGs FOR 45 PGs
DBKEY POSITIONS.... SET.................. TYPE....... NEXT PRIOR OWNER
           CALC  MEMBER 1  2
           DEPT-EMPLOYEE INDEX MEMBER 3  4
           EMP-NAME-NDX INDEX MEMBER 5
           EMP-SSN-NDX INDEX MEMBER 6
           OFFICE-EMPLOYEE INDEX MEMBER 7  8
           EMP-COVERAGE OWNER 9 10
           EMP-EMPOSITION OWNER 11 12
           EMP-EXPERTISE OWNER 13 14
           MANAGES OWNER 15 16
           REPORTS-TO OWNER 17 18
DATA ITEM......... REDEFINES... USAGE....... VALUE..... PICTURE. STRT LGTH
 02 EMP-ID-0415 DISPLAY 9(4) 1  4
 02 EMP-NAME-0415 DISPLAY 5  25
 03 EMP-FIRST-NAME-0415 DISPLAY X(10) 5  10
 03 EMP-LAST-NAME-0415 DISPLAY X(15) 15  15
 02 EMP-ADDRESS-0415 DISPLAY 30  46
 03 EMP-STREET-0415 DISPLAY X(20) 30  20
 03 EMP-CITY-0415 DISPLAY X(15) 50  15
 03 EMP-STATE-0415 DISPLAY X(2) 65  02
 03 EMP-ZIP-0415 DISPLAY 67  9
 04 EMP-ZIP-FIRST-FIVE-0415 DISPLAY X(5) 67  5
 04 EMP-ZIP-LAST-FOUR-0415 DISPLAY X(4) 72  4
 02 EMP-PHONE-0415 DISPLAY 9(10) 76 10
 02 STATUS-0415 DISPLAY X(2) 86  2
 88 ACTIVE-0415   COND '01'  86
 88 ST-DISABIL-0415   COND '02'  86
 88 LT-DISABIL-0415   COND '03'  86
 88 LEAVE-OF-ABSENCE-0415   COND '04'  86
 88 TERMINATED-0415   COND '05'  86
 02 SS-NUMBER-0415 DISPLAY 9(9) 88  9
 02 START-DATE-0415 DISPLAY 97  8
 02 START-YEAR-0415 DISPLAY 9(4) 97  4
 03 START-MONTH-0415 DISPLAY 9(2) 101  2
 03 START-DAY-0415 DISPLAY 9(2) 103  2
 02 TERMINATION-DATE-0415 DISPLAY 105  8
 03 TERMINATION-YEAR-0415 DISPLAY 9(4) 105  4
 03 TERMINATION-MONTH-0415 DISPLAY 9(2) 109  2
 03 TERMINATION-DAY-0415 DISPLAY 9(2) 111  2
 02 BIRTH-DATE-0415 DISPLAY 113  8
 03 BIRTH-YEAR-0415 DISPLAY 9(4) 113  4
 03 BIRTH-MONTH-0415 DISPLAY 9(2) 117  2
 03 BIRTH-DAY-0415 DISPLAY 9(2) 119  2

IDMSRPTS nn.

SCHEMA RECORD DESCRIPTION LISTING

RECD   DICTIONARY APPLDICT OF NODE DEFAULT

SCHEMA EMPSCHM VERSION 100

RECORD NAME........ EMPOSITION            RLGTH=  56
RECORD VERSION..... 0100              DLGTH=  32
RECORD ID.......... 0420              KLGTH=  24
RECORD LENGTH...... FIXED              DSTRT=  24
LOCATION MODE...... VIA SET EMP-EMPOSITION DISPLACEMENT 0000 PAGES
WITHIN............. EMP-DEMO-REGION OFFSET 5 PGs FOR 45 PGs
DBKEY POSITIONS.... SET.................. TYPE....... NEXT PRIOR OWNER
           EMP-EMPOSITION MEMBER 1  2  3
           JOB-EMPOSITION MEMBER 4  5  6
DATA ITEM......... REDEFINES... USAGE....... VALUE..... PICTURE. STRT LGTH
02 START-DATE-0420 DISPLAY 1 8
03 START-YEAR-0420 DISPLAY 9(4) 1 4
03 START-MONTH-0420 DISPLAY 9(2) 5 2
03 START-DAY-0420 DISPLAY 9(2) 7 2
02 FINISH-DATE-0420 DISPLAY 9 8
03 FINISH-YEAR-0420 DISPLAY 9(4) 9 4
03 FINISH-MONTH-0420 DISPLAY 9(2) 13 2
03 FINISH-DAY-0420 DISPLAY 9(2) 15 2
02 SALARY-GRADE-0420 DISPLAY 9(2) 17 2
02 SALARY-AMOUNT-0420 COMP-3 S9(7)V99 19 5
02 BONUS-PERCENT-0420 COMP-3 SV999 24 2
02 COMMISSION-PERCENT-0420 COMP-3 SV999 26 2
02 OVERTIME-RATE-0420 COMP-3 S9V99 28 2
02 FILLER DISPLAY XXX 30 3

02 START-DATE-0420 DISPLAY 1 8
03 START-YEAR-0420 DISPLAY 9(4) 1 4
03 START-MONTH-0420 DISPLAY 9(2) 5 2
03 START-DAY-0420 DISPLAY 9(2) 7 2
02 FINISH-DATE-0420 DISPLAY 9 8
03 FINISH-YEAR-0420 DISPLAY 9(4) 9 4
03 FINISH-MONTH-0420 DISPLAY 9(2) 13 2
03 FINISH-DAY-0420 DISPLAY 9(2) 15 2
02 SALARY-GRADE-0420 DISPLAY 9(2) 17 2
02 SALARY-AMOUNT-0420 COMP-3 S9(7)V99 19 5
02 BONUS-PERCENT-0420 COMP-3 SV999 24 2
02 COMMISSION-PERCENT-0420 COMP-3 SV999 26 2
02 OVERTIME-RATE-0420 COMP-3 S9V99 28 2
02 FILLER DISPLAY XXX 30 3

IDMSRPTS nn.

RECODES DICTIONARY APPLDICT OF NODE DEFAULT

SCHEMA EMPSCHM VERSION 100

RECORD NAME........ EXPERTISE RLGTH= 32
RECORD VERSION..... 0100 DLGH= 12
RECORD ID.......... 0425 KLGTH= 20
RECORD LENGTH..... FIXED DSTRT= 20
LOCATION MODE..... VIA SET EMP-EXPERTIS DISPLACEMENT 0000 PAGES
WITHIN............. EMP-DEMO-REGION OFFSET 5 PGS FOR 45 PGS
DBKEY POSITIONS.... SET............... TYPE......... NEXT PRIOR OWNER
EMP-EXPERTISE MEMBER 1 2 3
SKILL-EXPERTISE INDEX MEMBER 4 5
DATA ITEM........ REDEFINES..... USAGE....... VALUE..... PICTURE. STRT LGTH
02 SKILL-LEVEL-0425 DISPLAY XX 1 2
88 EXPERT-0425 COND '04' 1
88 PROFICIENT-0425 COND '03' 1
88 COMPETENT-0425 COND '02' 1
88 ELEMENTARY-0425 COND '01' 1
03 EXPERTISE-YEAR-0425 DISPLAY 9(4) 3 4
03 EXPERTISE-MONTH-0425 DISPLAY 9(2) 7 2
03 EXPERTISE-DAY-0425 DISPLAY 9(2) 9 2
02 FILLER DISPLAY XX 11 2

IDMSRPTS nn.

RECODES DICTIONARY APPLDICT OF NODE DEFAULT

SCHEMA EMPSCHM VERSION 100

RECORD NAME........ HOSPITAL-CLAIM RLGTH= 304
RECORD VERSION..... 0100 DLGH= 300
RECORD ID.......... 0430 KLGTH= 4
RECORD LENGTH..... FIXED DSTRT= 4
LOCATION MODE..... VIA SET COVERAGE-CLAIMS DISPLACEMENT 0000 PAGES
WITHIN............. INS-DEMO-REGION OFFSET 5 PGS FOR 20 PGS
DBKEY POSITIONS.... SET............... TYPE......... NEXT PRIOR OWNER
COVERAGE-CLAIMS MEMBER 1
DATA ITEM........ REDEFINES..... USAGE....... VALUE..... PICTURE. STRT LGTH
02 CLAIM-DATE-0430 DISPLAY 1 8
03 CLAIM-YEAR-0430 DISPLAY 9(4) 1 4
03 CLAIM-MONTH-0430 DISPLAY 9(2) 5 2
03 CLAIM-DAY-0430 DISPLAY 9(2) 7 2
02 PATIENT-NAME-0430 DISPLAY 9 25
03 PATIENT-FIRST-NAME-0430 DISPLAY X(10) 9 10
03 PATIENT-LAST-NAME-0430 DISPLAY X(15) 19 15
02 PATIENT-BIRTH-DATE-0430 DISPLAY 9(4) 34 8
02 PATIENT-BIRTH-YEAR-0430 DISPLAY 9(4) 34 4
03 PATIENT-BIRTH-MONTH-0430 DISPLAY 9(2) 38 2
03 PATIENT-BIRTH-DAY-0430 DISPLAY 9(2) 40 2
02 PATIENT-SEX-0430 DISPLAY X 42 1
02 RELATION-TO-EMPLOYEE-0430 DISPLAY X(10) 43 10
02 HOSPITAL-NAME-0430 DISPLAY X(25) 53 25
02 HOSP-ADDRESS-0430 DISPLAY 78 46
03 HOSP-STREET-0430 DISPLAY X(20) 78 20
03 HOSP-CITY-0430 DISPLAY X(15) 98 15
03 HOSP-STATE-0430 DISPLAY X(2) 113 2
03 HOSP-ZIP-0430 DISPLAY 115 9
04 HOSP-ZIP-FIRST-FIVE-0430 DISPLAY X(5) 115 5
04 HOSP-ZIP-LAST-FOUR-0430 DISPLAY X(4) 120 4
02 ADMIT-DATE-0430 DISPLAY 124 8
03 ADMIT-YEAR-0430 DISPLAY 9(4) 124 4
03 ADMIT-MONTH-0430 DISPLAY 9(2) 128 2
03 ADMIT-DAY-0430 DISPLAY 9(2) 130 2
02 DISCHARGE-DATE-0430 DISPLAY 132 8
03 DISCHARGE-YEAR-0430 DISPLAY 9(4) 132 4
03 DISCHARGE-MONTH-0430 DISPLAY 9(2) 136 2
03 DISCHARGE-DAY-0430 DISPLAY 9(2) 138 2
02 DIAGNOSIS-0430 DISPLAY OCCURS 2 X(60) 140 120
03 ROOM-AND-BOARD-0430 DISPLAY 260 26
04 WARD-0430 DISPLAY 260 13
05 WARD-DAYS-0430 COMP-3 S9(5) 260 3
05 WARD-RATE-0430 COMP-3 S9(7)V99 273 5
05 WARD-TOTAL-0430 COMP-3 S9(7)V99 281 5
04 SEMI-PRIVATE-0430 DISPLAY 273 13
05 SEMI-DAYS-0430 COMP-3 S9(5) 273 3
05 SEMI-RATE-0430 COMP-3 S9(7)V99 276 5
05 SEMI-TOTAL-0430 COMP-3 S9(7)V99 281 5
03 OTHER-ChARGES-0430 DISPLAY 286 15
04 DELIVERY-COST-0430 COMP-3 S9(7)V99 286 5
04 ANESTHESIA-COST-0430 COMP-3 S9(7)V99 291 5
04 LAB-COST-0430 COMP-3 S9(7)V99 296 5

IDMSRPTS nn.

n SCHEMA RECORD DESCRIPTION LISTING DATE TIME PAGE
RECDES DICTIONARY APPLDICT OF NODE DEFAULT mm/dd
/yy hhmms 12

SCHEMA EMPSCHM VERSION 100

RECORD NAME........ INSURANCE-PLAN RLGTH= 140
RECORD VERSION..... 0100 DLGTH= 132
RECORD ID.......... 0435 KLGTH= 8
RECORD LENGTH...... FIXED DSTRT= 8
LOCATION MODE..... CALC USING INS-PLAN-CODE-0435 DUPLICATES NOT ALLOWED
WITHIN................ INS-DEMO-REGION OFFSET 1 PGS FOR 4 PGS
DBKEY POSITIONS.... SET........... TYPE........ NEXT PRIOR OWNER
CALC MEMBER 1 2
DATA ITEM......... REDEFINES... USAGE...... VALUE..... PICTURE. STRT LGTH
02 INS-PLAN-CODE-0435 DISPLAY X(3) 1 3
88 GROUP-LIFE-0435 COND '001' 1
88 HMO-0435 COND '002' 1
88 GROUP-HEALTH-0435 COND '003' 1
88 GROUP-DENTAL-0435 COND '004' 1
02 INS-NAME-0435 DISPLAY X(45) 4 45
02 INS-ADDRESS-0435 DISPLAY 49 46
03 INS-PLACE-0435 DISPLAY X(20) 49 20
03 INS-CITY-0435 DISPLAY X(15) 60 15
03 INS-STATE-0435 DISPLAY X(2) 84 2
03 INS-ZIP-0435 DISPLAY 86 9
04 INS-CHIEF-0435 DISPLAY X(5) 86 5
04 INS-CHIEF-LAST-FOUR-0435 DISPLAY X(4) 91 4
02 INS-PHONE-0435 DISPLAY 9(10) 95 10
02 GROUP-NUMBER-0435 DISPLAY 9(6) 105 6
02 PLAN-DESCRIPTION-0435 DISPLAY 111 20
03 DEDUCT-0435 COMP-3 S9(7)V99 111 5
03 MAXIMUM-LIFE-COST-0435 COMP-3 S9(7)V99 116 5
03 FAMILY-COST-0435 COMP-3 S9(7)V99 121 5
03 DEP-COST-0435 COMP-3 S9(7)V99 126 5
02 FILLER DISPLAY XX 131 2

IDMSRPTS nn.

n SCHEMA RECORD DESCRIPTION LISTING DATE TIME PAGE
RECDES DICTIONARY APPLDICT OF NODE DEFAULT mm/dd

15-Jan-2018 387/415
/yy hhmss 14

SCHEMA EMPSCHM VERSION 100

RECORD NAME........ JOB
RLGTH= 324
RECORD VERSION..... 0100
DLGTH= 300
RECORD ID.......... 0440
KLGH= 24
RECORD LENGTH...... FIXED (INTERNALLY VARIABLE) DSTRT= 28
MINIMUM ROOT....... 24 CHARACTERS
MINIMUM FRAGMENT... 296 CHARACTERS
LOCATION MODE...... CALC USING JOB-ID-0440 Duplicates not allowed
WITHIN............. ORG-DEMO-REGION OFFSET 5 PGS FOR 20 PGS
CALL PROCEDURES.... NAME WHEN FUNCTION
IDMSCOMP BEFORE STORE
IDMSCOMP BEFORE MODIFY
IDMSDCOM AFTER GET
DBKEY POSITIONS.... SET............ TYPE......... NEXT PRIOR OWNER
CALC MEMBER 1 2
JOB-TITLE-NDX INDEX MEMBER 3
JOB-EMPOSITION OWNER 4 5
(FRAGMENT CHAIN) INTRNL 6
DATA ITEM.......... REDEFINES... USAGE....... VALUE..... PICTURE. STRT LGTH
02 JOB-ID-0440 DISPLAY 9(4) 1 4
02 TITLE-0440 DISPLAY X(20) 5 20
03 DESCRIPTION-LINE-0440 DISPLAY OCCURS 2 X(60) 25 120
02 REQUIREMENTS-0440 DISPLAY 145 120
03 REQUIREMENT-LINE-0440 DISPLAY OCCURS 2 X(60) 145 120
02 MINIMUM-SALARY-0440 DISPLAY S9(6)V99 265 8
02 MAXIMUM-SALARY-0440 DISPLAY S9(6)V99 273 8
02 SALARY-GRADES-0440 DISPLAY OCCURS 4 X(2) 281 8
02 NUMBER-OF-POSITIONS-0440 DISPLAY 9(3) 289 3
02 NUMBER-OPEN-0440 DISPLAY 9(3) 292 3
02 FILLER DISPLAY XX 205 2

IDMSRPTS n.

SCHEMA RECORD DESCRIPTION LISTING DATE TIME PAGE
RECDES DICTIONARY APPLDICT OF NODE DEFAULT mm/dd

/yy hhmss 15

SCHEMA EMPSCHM VERSION 100

RECORD NAME....... NON-HOSP-CLAIM
RLGTH= 1064
RECORD VERSION..... 0100
DLGTH= 1056
RECORD ID.......... 0445
KLGH= 8
RECORD LENGTH...... VARIABLE DSTRT= 12
MINIMUM ROOT....... 248 CHARACTERS
MINIMUM FRAGMENT... 1052 CHARACTERS
LOCATION MODE...... VIA SET COVERAGE-CLAIMS DISPLACEMENT 0000 PAGES
WITHIN............. INS-DEMO-REGION OFFSET 5 PGS FOR 20 PGS
DBKEY POSITIONS.... SET............ TYPE......... NEXT PRIOR OWNER
COVERAGE-CLAIMS MEMBER 1

DATA ITEM.......... REDEFINES... USAGE....... VALUE..... PICTURE. STRT LGTH
02 CLAIM-DATE-0445 DISPLAY 1 8
03 CLAIM-YEAR-0445 DISPLAY 9(4) 1 4
03 CLAIM-MONTH-0445 DISPLAY 9(2) 5 2
03 CLAIM-DAY-0445 DISPLAY 9(2) 7 2
02 PATIENT-NAME-0445 DISPLAY 9 25
03 PATIENT-FIRST-NAME-0445 DISPLAY X(10) 9 10
03 PATIENT-LAST-NAME-0445 DISPLAY X(15) 19 15
02 PATIENT-BIRTH-DATE-0445 DISPLAY 34 8
03 PATIENT-BIRTH-YEAR-0445 DISPLAY 9(4) 34 4
03 PATIENT-BIRTH-MONTH-0445 DISPLAY 9(2) 38 2
03 PATIENT-BIRTH-DAY-0445 DISPLAY 9(2) 40 2
02 PATIENT-SEX-0445 DISPLAY X 42 1
02 RELATION-TO-EMPLOYEE-0445 DISPLAY X(10) 43 10
02 PHYSICIAN-NAME-0445 DISPLAY 53 25
03 PHYSICIAN-FIRST-NAME-0445 DISPLAY X(10) 53 10
03 PHYSICIAN-LAST-NAME-0445 DISPLAY X(15) 63 15
02 PHYSICIAN-ADDRESS-0445 DISPLAY 78 46
03 PHYSICIAN-STREET-0445 DISPLAY X(20) 78 20
03 PHYSICIAN-CITY-0445 DISPLAY X(15) 98 15
03 PHYSICIAN-STATE-0445 DISPLAY X(2) 113 2
03 PHYSICIAN-ZIP-0445 DISPLAY 115 9
04 PHYSICIAN-ZIP-FIRST-FIVE-0445 DISPLAY X(5) 115 5
04 PHYSICIAN-ZIP-LAST-FOUR-0445 DISPLAY X(4) 120 4
02 PHYSICIAN-ID-0445 DISPLAY 9(6) 124 6
02 DIAGNOSIS-0445 DISPLAY OCCURS 2 X(60) 130 120
02 NUMBER-OF-PROCEDURES-0445 COMP 9(2) 250 2
02 FILLER DISPLAY X 252 1
02 PHYSICIAN-CHARGES-0445 DISPLAY OCCURS 0 TO 10 253 800
DEPENDING ON -- NUMBER-OF-PROCEDURES-0445
03 SERVICE-DATE-0445 DISPLAY 1 8
04 SERVICE-YEAR-0445 DISPLAY 9(4) 1 4
04 SERVICE-MONTH-0445 DISPLAY 9(2) 5 2
04 SERVICE-DAY-0445 DISPLAY 9(2) 7 2
03 PROCEDURE-CODE-0445 DISPLAY 9(4) 9 4
03 DESCRIPTION-OF-SERVICE-0445 DISPLAY X(60) 13 60
03 FEE-0445 COMP-3 S9(7)V99 73 5
03 FILLER DISPLAY XXX 78 3
IDMSRPTS nn.

n SCHEMA RECORD DESCRIPTION LISTING DATE TIME PAGE
RECODES DICTIONARY APPLDICT OF NODE DEFAULT mm/dd
/yy hhmms 18
SCHEMA EMPSCHM VERSION 100

IDMSRPTS nn.

n SCHEMA RECORD DESCRIPTION LISTING DATE TIME PAGE
RECODES DICTIONARY APPLDICT OF NODE DEFAULT mm/dd
/yy hhmms 20
SCHEMA EMPSCHM VERSION 100

IDMSRPTS nn.

n SCHEMA RECORD DESCRIPTION LISTING DATE TIME PAGE
RECODES DICTIONARY APPLDICT OF NODE DEFAULT mm/dd
/yy hhmms 22
SCHEMA EMPSCHM VERSION 100
EMPLOYEE Database Structure Diagram

The data structure diagram illustrates record relationships in the EMPLOYEE database.
VS COBOL II Support

This section discusses CA IDMS support for programs compiled under the VS COBOL II compiler. It is divided into two parts:

- Features Supported by CA IDMS (see page 391)
- Features Not Supported by CA IDMS (see page 394)

**Note:** This section applies only to programs run in the online DC/UCF system. Except where specifically noted, it does not apply to programs run in another region (such as batch or CICS Transaction Server) even if the programs contain CA IDMS DML commands. All the provisions of this section also apply to programs compiled under an LE-compliant compiler, unless otherwise noted. For more information about IBM Language Environment and LE-compliant compilers, see Considerations for IBM Language Environment (see page 395).

Programs compiled under VS COBOL II can be run under the IBM runtime Language Environment subject to the requirements documented by IBM and the CA IDMS restrictions documented below and in Considerations for IBM Language Environment (see page 395). IBM no longer supports programs running under the VS COBOL II runtime environment.

Features Supported by CA IDMS

The following COBOL II features are supported by CA IDMS:

- **Reentrancy** -- Fully reentrant and non-reentrant programs are supported. The RENT compiler option must be specified if the program is reentrant. NORENT must be specified if the program is non-reentrant.

  **Note:** Quasi-reentrancy is not supported for VS COBOL II programs. It is strongly recommended that all COBOL II programs be compiled with the RENT option. A separate copy of each NORENT COBOL II program will be loaded for each concurrent task. CPU and storage utilization will be extremely high.

- **Residency** -- Resident and nonresident programs are supported. The NORES compiler option causes all necessary VS COBOL II runtime support modules to be linked with the program. The program can then be executed without having to load any support modules. The RES option causes the runtime support modules to be brought in as needed during execution. The following combinations of RENT and RES options are supported:
  - RENT RES
  - NORENT NORES
**NORENT RES**

*Note:* 31-bit programs require the RENT and RES options. This combination is recommended for most efficient processing. The RES option is not relevant to LE-compliant compilers, which always use this option.

The RENT/NORES combination is not allowed by the VS COBOL II compiler. Do not confuse the RES compiler option with the CA IDMS RESIDENT parameter (assigned at SYSGEN or by using a DCMT command). The CA IDMS RESIDENT parameter causes the user program to be loaded into the resident pool during startup, and remains there for the duration of system execution.

**XA support** -- Full 31-bit support is provided. COBOL II programs can reside above or below the 16-meg line, and can execute in 24-bit or 31-bit mode. User data areas can reside below the 16-meg line (compiler option DATA(24)) or anywhere in the region (DATA(31)). The following table shows the default attributes assigned by the COBOL II compiler based on the combination of RES and RENT compiler options.

<table>
<thead>
<tr>
<th>Compiler options</th>
<th>Default RMODE/AMODE</th>
</tr>
</thead>
<tbody>
<tr>
<td>RES/RENT</td>
<td>RMODE(ANY) AMODE(ANY)</td>
</tr>
<tr>
<td>RES/NORENT</td>
<td>RMODE(24) AMODE(ANY)</td>
</tr>
<tr>
<td>NORES/NORENT</td>
<td>RMODE(24) AMODE(24)</td>
</tr>
</tbody>
</table>

To run a task in 31-bit mode, it must be defined with a LOCATION of ANY (at SYSGEN or at runtime using a DCMT VARY DYNAMIC PROGRAM command).

**Static and dynamic calls** -- CA IDMS supports the following types of calls provided by VS COBOL II:

- **CALL literal with NODYNAM**, static (more storage, less CPU)

- **CALL identifier**, dynamic (less storage, more CPU)

To call a program dynamically you must use the call identifier format. A VS COBOL II program can use the COBOL CALL verb to invoke an assembler or COBOL II subprogram. The CA IDMS TRANSFER CONTROL (LINK or XCTL) must be used for invoking VS COBOL subprograms. The subprogram must be defined to the system either at SYSGEN or by using a DCMT VARY DYNAMIC PROGRAM command. The correct language must be specified, and the NONOVERLAYABLE attribute must also be specified.

A COBOL II program and all the COBOL II subprograms that it calls dynamically must be compiled with the same RES/NORES compiler option.

A dynamic call is often a more efficient way for one online VS COBOL II program to call another than the use of a TRANSFER CONTROL DML command. Note, however, that when a dynamic call is made, the DC/UCF system is not aware that the application is running in a new program. Therefore, error messages and program statistics will not reflect the call.

There are also restrictions on using static or dynamic calls when invoking an assembler subprogram. If the assembler program is not fully reentrant or if the assembler program
issues any operating system SVC instructions, the program must be invoked with a DC
TRANSFER control statement. Note that use of SVC instructions in an online program presents
security and performance concerns. Such instructions should be avoided unless they are
absolutely necessary. In most cases, DC/UCF system functions can be used instead.

- **Optimizer** -- The COBOL II optimizer is fully supported. Service reloads do not have to be explicitly
coded in the program, as is required for VS COBOL.

- **STRING/UNSTRING/INSPECT** -- COBOL II verbs that require GETMAIN services are supported; this
includes STRING, UNSTRING, and INSPECT.

> **Note:** Exercise caution with STRING, UNSTRING, and INSPECT. Use of these may increase
SRB time. Commands in a VS/COBOL II environment may cause additional screening of
supervisor calls resulting in some performance degradation. This concern does not apply
when using VS/COBOL II in an IBM runtime Language Environment provided that
RHDCLEFE is defined in the IDMS/DC Sysgen.

- **Compiler options** -- The following compiler options that affect object code execution are
supported:

  - DATA
  - OPTIMIZE
  - PFDSGN
  - RENT
  - RESIDENT
  - SSRANGE
  - TRUNC

> **Note:** See the discussion of the TRUNC option in the section *Executing Programs* (see
page 26).

- ZWB

- **Execution time options** -- COBOL II has an options module (IGZEOPT) that can be assembled and
link-edited to control options at execution time. The module needs to be linked with each online
VS COBOL II application program. Valid macro values for parameters that affect CA IDMS
performance are shown below:

```
IGZOPT SYSTYPE=OS,
   DEBUG=N0,
   STAE=N0,
   AIXBLD=N0,
   SSRANGE=YES/N0,
   SPOUT=YES/N0
```
Features Not Supported by CA IDMS

The following COBOL II features are not supported by CA IDMS:

- ENVIRONMENT and DATA DIVISION entries normally associated with file management (for example, INPUT-OUTPUT SECTION, FILE SECTION)

- I/O statements, including ACCEPT, CLOSE, DELETE, DISPLAY, OPEN, READ, REWRITE, and WRITE

⚠️ **Note:** DATE/TIME related ACCEPT statements are supported in release 14.1 and later for COBOL II and LE-compliant compilers.

- The debugging features FDUMP and TEST

- The sorting features SORT and MERGE

- The compiler options ADV, DYNAM, FASTSRT, GRAPHIC, and OUTDD
Considerations for IBM Language Environment

This section applies only to runtime support for COBOL programs that run in an online DC/UCF region. It does not apply to batch or CICS programs that access CA IDMS. It also does not apply to online COBOL programs compiled with the "old" VS COBOL compiler, prior to VS COBOL II. Online VS COBOL programs must comply with the compile and linkage restrictions described in Compiling and Executing CA IDMS Programs (https://docops.ca.com/display/IDMSCU19/Introduction+to+CA+IDMS+Data+Manipulation+Language-Compiling+and+Executing+CA+IDMS+Programs). If these restrictions are observed, the LE runtime environment will not be accessed by VS COBOL programs. This section does apply to programs compiled under VS COBOL II when run in online LE runtime environment.

- What is IBM Language Environment (LE)? (see page 395)
- How Can You Use LE with CA IDMS/DC? (see page 396)
- Considerations About LE Runtime (see page 396)
- Running LE-Compliant Compiler Programs Under CA IDMS/DC (see page 397)
- Supported LE Functions (see page 400)
- Unsupported LE Functions (see page 401)
- Performance Improvements with RHDCLEFE (see page 401)
- Multiple-Program Enclave (see page 402)

What is IBM Language Environment (LE)?

LE is a runtime environment that replaces the language-specific runtime environments that existed previously. For example, VS COBOL had its own runtime environment; VS COBOL II had another. CA IDMS can execute programs that are designed to use the LE runtime environment. It can also execute programs compiled with pre-LE compilers that use the LE runtime environment subject to IBM’s documented restrictions.

Language Environment has had several names for different operating systems and release levels. The term "LE" will be used in this document to refer to the IBM runtime Language Environment for any of the following operating systems:

- z/VSE
  - z/OS
  - z/VM

⚠️ Note: This section applies only to runtime support in CA IDMS/DC. It does not apply to batch or CICS programs that access CA IDMS.
How Can You Use LE with CA IDMS/DC?

To execute online programs using the LE runtime libraries, follow these steps to bring up your CA IDMS environment:

1. Ensure that the CA IDMS system has been generated with a 24-bit reentrant pool (or program pool, if no reentrant pool is generated) that is large enough to contain the IBM-supplied LE application program interface module CEEPIPI. The size of this module is approximately 100K.

2. Ensure that the CA IDMS system has been generated with an XA reentrant pool that is large enough to maintain residence for several IBM-supplied LE support modules. Allow 5 megabytes for these programs.

Include the LE runtime load libraries in the CDMSLIB loadlib concatenation before any other IBM language loadlibs you are using. For example, before COBOL II.

Considerations About LE Runtime

Running Pre-LE Programs

There are restrictions that apply when you run pre-LE programs in an LE runtime environment within CA IDMS/DC. Pre-LE programs are programs that were compiled with a non-LE compliant compiler, such as COBOL II.

Some of these restrictions are already documented in Compiling and Executing CA IDMS Programs (https://docops.ca.com/display/IDMSCU19/Introduction+to+CA+IDMS+Data+Manipulation+Language#IntroductiontoCAIDMSDataManipulationLanguage-CompilingandExecutingCAIDMSPrograms) and VS COBOL II Support (see page 391). Additional restrictions for LE are:

- VS COBOL II programs have to run without storage protection unless RHDCLEFE (see "Performance Improvements with RHDCLEFE" below) is in use.

- VS COBOL II programs must be linked with an IGZEOPT module that specifies STAE=NO (see "Execution Time Options" in Appendix H: (see page 391), for more information on the use of IGZEOPT). If this restriction is not observed, a program check in a COBOL program will result in immediate termination of the program with no indication of an error. Certain other abnormal abend conditions may also go unreported. This restriction does not apply if one of the following conditions is true:
  - RHDCLEFE is in use. See "Performance Improvements with RHDCLEFE" later in this section for more information.
  - A special CEEDOPT or CEEROPT is in use as described later in this section under Runtime Options, and either or both of the following options is specified:
    ABTERMENC=((ABEND,...
    TRAP=((OFF,...
The IBM LE support module CEEPIPI must be loaded once before any VS COBOL II program is run. This is most easily done by defining CEEPIPI as RESIDENT in the CA IDMS/DC sysgen using the following syntax.

ADD PROGRAM CEEPIPI CONCURRENT ENABLED LANGUAGE ASSEMBLER NONOVERLAYABLE PROGRAM PROTECT REENTRANT RESIDENT REUSABLE.

Restrictions mentioned in the IBM documentation (for example, the IBM COBOL/370 Migration Guide) apply.

⚠️ **Note:** Running pre-LE programs with LE runtime can degrade performance in some circumstances. If you notice poor performance, you should consider recompiling the programs with the newer compiler or running with RHDCLEFE (see "Performance Improvements with RHDCLEFE" below). The use of RHDCLEFE also removes the necessity of forcing the load of CEEPIPI before running any VS COBOL II programs.

### Running LE Programs

**LE programs** are programs that were compiled with an LE-compliant compiler. CA IDMS/DC supports all LE-compliant compilers supported by IBM including:

- IBM COBOL for VM
- IBM Enterprise COBOL for z/OS
- COBOL for z/VSE

For convenience, programs compiled with an LE-compliant compiler are referred to as "LE COBOL" programs below.

### Running LE-Compliant Compiler Programs Under CA IDMS/DC

This section discusses Language Environment runtime options relevant to the online CA IDMS/DC environment.

⚠️ **Note:** Also see [Compiling and Executing CA IDMS Programs](https://docops.ca.com/display/IDMSCU19/Introduction+to+CA+IDMS+Data+Manipulation+Language#IntroductiontoCAIDMSDataManipulationLanguage-CompilingandExecutingCAIDMSPrograms) and Appendix H: (see page 391). The restrictions on VS COBOL and VS COBOL II compile and runtime options also apply to programs compiled with an LE-compliant COBOL compiler unless specifically noted below.
See Appendix A: (see page 280) for sample compile and link JCL for both batch and online programs which use CA IDMS DML statements.

Runtime Options

The IBM Language Environment provides numerous options that control how programs operate at runtime. The default values are designed to be suitable in a batch environment. Therefore, it is necessary to modify some values for applications that are to run in a DC/UCF online system.

Note: As stated in the introduction to this section, this section does not apply to programs that run in a CICS or other region, even if they access CA IDMS using DML or SQL commands. It does apply to programs that run a DC/UCF online system, which are invoked from another front-end using CA IDMS UCF (such as an ADS/O application that is accessed using UCFCICS from a CICS front-end).

The IBM Language Environment provides a number of ways to specify runtime options. Four methods are supported for CA IDMS/DC online programs:

1. Modify, assemble, and link the IBM-supplied CEEUOPT module. Link the resulting module with each application program. Product Documentation Change LI18624 contains a sample version of the CEEUOPT with values that are appropriate for most online CA IDMS applications. Also consult the section "Creating an Application-Specific Runtime Options Module" in IBM's LE Installation and Customization Manual.

2. Assemble and link a CEEUOPT module as described above. Link the resulting module with RHDCLEFE. Make sure that RHDCLEFE is defined in the DC/UCF Sysgen (as described under "Performance Improvements Using RHDCLEFE" below). This option affects only COBOL programs. This is the recommended option for all online COBOL applications.

3. Assemble and link a specialized CEEDOPT module.

Note: This method is not available for z/OS Version 1.10 and higher. Use method 1 or method 4 for non-COBOL applications on z/OS Versin 1.10 and higher.

If this method is chosen, special copies of the IBM modules CEEBINIT and CEPIPI must be maintained for use with online DC/UCF systems only. Due to maintenance considerations, this method is not recommended for COBOL applications. It is needed for PL/I programs compiled with a non-LE-compliant compiler. For more information on using this method, see Product Documentation Change LI23664.

4. Assemble and link a specialized CEEROPT module.
Note: This method is not available for z/OS Version 1.9 and lower or for VSE. Use method 1 or 3 for those operating systems.

If this method is chosen, a CEEROPT load module can be created to override desired options. Like CEEUOPT, and unlike CEEDOPT, you only need to specify those options which are to be different from the installation default LE run-time operations. The resultant load module must be included in a load library in the CDMSLIB concatenation ahead of the default SCEERUN load library.

Note: The CEEROPT will be loaded in a CA IDMS region only if your CEEPRMxx member specified CEEROPT(ALL). For more information on using this module, refer to IBM documentation.

Except as discussed below, the IBM-supplied default runtime options can be used with any site-specific desired modifications. Note that the MSGFILE parameter is ignored and messages are sent to the CA IDMS log file.

Recommended settings for certain parameters are as shown below. For more details on these parameters see the IBM Language Environment Customization manual.

- ABTERMENC=(RETCODE) or ABTERMENC=(ABEND): This parameter affects the action taken when an LE enclave ends with an unhandled condition of severity 2 or higher. If RETCODE code is specified, the DC task will abend with message DC128004. If ABEND is specified, the DC task will abend with a Uxxx where xxx corresponds to the hexadecimal value of the user abend code set by LE. For example, an LE user abend 4093 would result in a DC task abend with code UFFD.

- ALL31=(ON): This parameter will minimize the amount of below-the-line storage that will be allocated by LE. This parameter requires that all COBOL programs are linked with AMODE(31). It is strongly recommended that any non-conforming programs be relinked so that ALL31=(ON) can be specified.

- DEBUG=(OFF): The DEBUG runtime option cannot be used in a DC environment.

- INTERRUPT=(OFF): Attention interrupts are handled by the CA IDMS/DC system and not by LE runtime support. Application COBOL programs can test for attention interrupts using the DC-ATTN-INT condition name under LE just as with earlier COBOL runtime environments.

- POSIX=(OFF): POSIX is not supported under DC/UCF.

- RPTSTG=(OFF) or RPTSTG=(ON): Normally OFF should be specified. OFF must be specified for systems prior to release 14.1. The purpose of RPTSTG is to determine the storage utilization for a particular application. The report is produced at the end of a COBOL task thread and is written to the CA IDMS log file. For efficiency reasons, the termination phase of COBOL processing is normally not executed in an online DC environment. If it is necessary to obtain storage information for a particular application, optional bit 196 can be set (See "COBOL II and LE COBOL Task Management" in Optional Online COBOL Functionality (see page 404)). Note that this option adversely affects performance. Storage reports are therefore normally produced only in a test or development system.

- TERMTHDACT=(QUIET) or TERMTHDACT=(TRACE): This option controls the extent of LE runtime information that will be supplied when an application terminates. All messages will be written to the DC log file.
TRAP=(ON) or TRAP=(OFF): If ON is specified, program checks in an LE application will result in IBM LE error-handling being put into effect. COBOL-specific and LE messages will be written to the log. After these messages are written and the COBOL thread ends abnormally, the DC task will abend with message DC128004 and a task snap will be taken. If OFF is specified, program checks in an LE application will result in an immediate task snap. This is similar to the result in a VS COBOL or VS COBOL II runtime environment. No LE messages related to the program check will be written. Furthermore, if any PL/I applications are included in the online system, any ON ERROR clauses will not be handled properly.

In addition to the parameters above, we strongly recommend that you use smaller values than the default ones for the various heap (ANYHEAP, BELOWHEAP, and HEAP) parameters and stack (LIBSTACK and STACK) parameters because these are allocated on a task thread basis. Storage allocation is most efficient if relatively large values are specified as sixteen bytes less than a multiple of 4096. Smaller values than 4096 should be set for some parameters to avoid wasting storage. The values shown below have been found to be suitable for most DC/UCF systems.

Even when the smallest possible storage values are chosen, the IBM Language Environment requests a substantial amount of below-the-line storage for each program invoked in an online task--particularly with older releases of LE. This storage is used for functions which are not supported in an online DC/UCF system. For this reason, DC/UCF provides optional functionality which forces all LE storage to be allocated above the 16M line for tasks which are defined as LOCATION ANY. You can enable this functionality by specifying #DEFOPT OPT00227 when compiling module RHDCOPTF.

ANYHEAP=(2032, 4080, ANYWHERE, FREE)
BELOWHEAP=(496, 496, FREE)
HEAP=(2032, 4080, ANYWHERE, KEEP, 2032, 2032)
LIBSTACK=(496, 496, FREE)
STACK=(2032, 8176, ANY, KEEP)
STORAGE=(NONE, NONE, NONE, 0)
THREADHEAP=(0100, 0100, ANYWHERE, KEEP)

Supported LE Functions

CA IDMS/DC supports these LE functions:

- Math services
- National language support services
- Date and time services
- XML parsing

CA IDMS/DC also supports storage management services, but for performance reasons, they are not recommended. The storage management services are:

- CEECRHP: Create heap segment
- CEECZST: Re-allocate (change size of) heap storage
- CEEDSHP: Discard heap segment
- CEEFRST: Free heap storage
- CEEGTST: Get heap storage

Unsupported LE Functions

CA IDMS/DC does not support the following LE functions:

- CEE3PRM: Get exec parms
- CEETDLI: Call IMS
- CEETEST: Invoke debugging environment

Performance Improvements with RHDCLEFE

Beginning with Release 14.1, CA IDMS supports a more efficient method of running online VS COBOL II and LE COBOL programs under LE runtime. In order to realize this performance improvement, link RHDCLEFE and define it in the CA IDMS sysgen with the following values:

```
ADD PROGRAM RHDCLEFE
CONCURRENT
DYNAMIC
ENABLED
LANGUAGE IS ASSEMBLER
NEW COPY IS ENABLED
NONOVERLAYABLE
PROGRAM
NPROTECT
REENTRANT
RESIDENT
REUSABLE.
```

The advantages of using defining RHDCLEFE in an LE runtime environment are as follows.

- COBOL II programs can run with Storage Protect.
- If RHDCLEFE is in use, it is not necessary to link CEEUOPT with each application program.
- If a VS COBOL II or an LE COBOL program is invoked multiple times in the same task using an CA IDMS DML call (#LINK from Assembler, DC TRANSFER from COBOL or PL/I, or LINK from ADS/O), then only one LE enclave and one LE environment will be established. The use of RHDCLEFE can reduce the CPU usage for TRANSFER CONTROL to another COBOL program, particularly a VS COBOL II program. Without RHDCLEFE, each such invocation of a VS COBOL II program will result in the establishment and termination of both the environment and the enclave. Each such invocation of a LE COBOL program will result in the establishment and termination of the enclave.

⚠️ Note: RHDCLEFE is linked with a CEEUOPT with ALL31=(ON). As a consequence, all LE COBOL and VS COBOL II programs must be linked with AMODE(31) or AMODE(any).
Multiple-Program Enclave

This feature became available on release 15.0 service pack 3.

You can improve the performance of certain online applications that use COBOL programs under the IBM Language Environment (LE) by enabling a new optional feature which allows the use of a single LE enclave for multiple programs. The following explains the conditions under which performance can be improved and some restrictions on the programs that can utilize this new feature:

- Because of restrictions on the applications that can use the new functionality, this feature is not in effect unless MULTIPLE ENCLAVE IS ON is specified on the SYSTEM statement in the DC System Generation. In addition, module RHDCLEFE must be in use as described in "Performance Improvements with RHDCLEFE." In release 15.0, this feature is available only for z/OS operating systems.

- When MULTIPLE ENCLAVE IS OFF, each new LE program invoked within a DC online task causes the initialization of a new LE process and enclave, provided the program was invoked as a result of one of the following:
  - The DC task definition specified INVOKES PROGRAM...
  - The program was invoked using a TRANSFER CONTROL.
  - After an LE program is invoked in a given task, the same process and enclave can be reused if one of the following occurs:
    - The same program is invoked subsequently in the same task.
    - A different program is invoked from an LE COBOL program using a static CALL (CALL 'literal') or a dynamic CALL (CALL IDENTIFIER).

  When MULTIPLE ENCLAVE IS ON, a new LE process and enclave are created the first time an LE COBOL program is invoked in a task. Subsequent invocations of any COBOL program in the same task utilizes the same process and enclave even if it was invoked using TRANSFER CONTROL LINK or TRANSFER CONTROL RETURN.

  Starting an LE process and/or enclave involves considerable overhead of both storage and CPU utilization. Therefore, MULTIPLE ENCLAVE IS ON can provide significant improvement for tasks that invoke many programs using TRANSFER CONTROL RETURN or TRANSFER CONTROL LINK.

Restrictions on Using Multiple-Program Enclaves

The following restrictions apply to COBOL programs that participate in a multiple-program enclave:

- Enabled programs cannot perform a DC RETURN DML call and then be reentered using a subsequent TRANSFER. This restriction does not apply to programs that contain a DC RETURN with no subparameters because the DML compiler generates a GOBACK for this type of statement. This restriction does apply if the DC RETURN statement does have subparameters. For example, you cannot execute a "DC RETURN NEXT TASKCODE ..." statement and then reenter the same program in the same task.
Enabled programs cannot issue a TRANSFER CONTROL NORETURN or a TRANSFER CONTROL XCTL.

Optional bit 196 is ignored for programs that participate in a multiple-program enclave. Therefore, if MULTIPLE ENCLAVE IS ON at the system level, any program that depends on bit 196 must be exempted as described in "Exempting Programs from Multiple-Program Enclave."

Exempting Programs from Multiple-Program Enclave

You can enable multiple-program enclaves at the system level even if some programs are not eligible. An ineligible program can be exempted in one of two ways:

- Use the MULTIPLE ENCLAVE IS OFF clause of the PROGRAM statement in the DC System Generation.
- Use the MULTIPLE ENCLAVE OFF clause on the DCMT VARY PROGRAM statement or the DCMT VARY DYNAMIC PROGRAM statement.

Exempted programs can participate in the same task with eligible programs. All eligible programs share one process/enclave. Each exempted program uses its own process/enclave.
Optional Online COBOL Functionality

Several APARs have been written that affect the performance and/or functionality of COBOL programs in the online CA IDMS/DC system. This article discusses the effects of the various APARs.

- COBOL II and LE COBOL Task Management (see page 404)
- PSW Program Mask Settings (see page 407)
- Loading VS COBOL Programs into XA Storage (see page 408)

⚠️ **Note:** This discussion applies only to online programs running in a DC/UCF region. It does not apply to batch programs or to programs running in a CICS region or under control of another TP monitor, even if such programs access an IDMS database via LOCAL or CV mode.

### COBOL II and LE COBOL Task Management

Several optional APARs have to do with the management of a COBOL II task thread or a LE COBOL process(environment) and enclave. To better understand the concept of a COBOL task thread, first consider a batch COBOL job in which IDMS is involved. When a COBOL II program is first invoked, the COBOL support code causes the load of a small program called IGZCTCO. As the COBOL II runtime system is built, control information is placed in the copy of IGZCTCO that has been loaded into the address space. If the top level program (call it program A) issues a CALL IDENTIFIER to a second COBOL program (call it program B), the COBOL II support code finds the existing copy of IGZCTCO. Program B is entered using the same COBOL II environment. The first time program B is entered, its WORKING STORAGE is initialized according to any VALUE clauses coded.

If program B does a GOBACK to program A and then program A issues a second CALL IDENTIFIER to program B, program B is normally entered with the same WORKING STORAGE values left from the previous invocation. The VALUE clauses are not reinitialized.

In Language Environment for z/OS, the concepts of the LE process and enclave are roughly analogous to the COBOL II task thread. See IBM documentation for a more complete discussion of these concepts.

Now let us return to the discussion of COBOL II in an online DC/UCF system. When COBOL II support was first introduced for DC/UCF, every invocation of a COBOL program via a TRANSFER RETURN from another COBOL program caused a new IGZCTCO to be loaded. The COBOL II task thread was recreated. Thus if Task A invokes COBOL II program X, which does a TRANSFER CONTROL ten times to program Y, the COBOL II task thread was built eleven times -- once for program X and ten times for program Y. Moreover, if program Y terminated with a DC RETURN instead of a GOBACK, all the storage associated with each invocation was preserved until task termination. This causes serious overhead of CPU and potentially of storage utilization.
To reduce the overhead of constantly creating new COBOL II environments, the DC/UCF COBOL II support was modified to load only one copy of IGZCTCO per task. Using this method, if main program X issues a TRANSFER CONTROL ten times to program Y, the COBOL II environment is built only once. The drawback is that certain functionality is changed. In particular, WORKING STORAGE is not reinitialized each time program Y is entered. Also, recursive TRANSFER CONTROL (Program X issues TRANSFER CONTROL TO X) is not allowed. Since some existing applications depended on those features, optional APARs were developed to allow use of one method or the other. Unfortunately, the DC/UCF default methodology changed from release to release and sometimes within one release.

Note: IGZCTCO is handled differently in DC/UCF systems that are operating with an IBM Language Environment runtime system (such as LE for z/OS). COBOL II programs can be used in these systems, but prior to release 14.1, a new IGZCTCO was used for every TRANSFER to a COBOL II program.

Beginning with release 14.1, online COBOL II programs in an LE runtime environment will run most efficiently if RHDCLEFE is defined in the DC/UCF Sysgen as documented in the release 14.1 Features Guide. This gives functionality similar to that documented for the "single IGZCTCO" method shown below. In that case, the COBOL II program is handled as though it were compiled under LE COBOL.

The following table describes how to utilize each of the two methods for the latest maintenance of all currently supported DC/UCF releases. This table supercedes the documentation in any previous APARs or PDCs. Note that the table is divided into several sections depending on the COBOL compiler level and the runtime level.

**Part 1**

This part of the table contains programs compiled under COBOL II and using COBOL II runtime libraries.

<table>
<thead>
<tr>
<th>Release</th>
<th>Method 1 (Most CPU efficient) Use single IGZCTCO per task</th>
<th>Method 2 (Special functionality) Use new IGZCTCO each TRANSFER</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.21PS</td>
<td>Default</td>
<td>Apply optional APAR 88-06-1105</td>
</tr>
<tr>
<td>12.01</td>
<td>Default (see note below)</td>
<td>Apply optional APAR LS12053.</td>
</tr>
<tr>
<td>14.0 and later</td>
<td>Turn on optional bit 49 in RHDCOPTF.</td>
<td>Default.</td>
</tr>
</tbody>
</table>

Note: In release 12.01, prior to maintenance level 9607, it is necessary to apply APAR GO97250 to obtain the default condition shown above. With application of GO97250, optional APAR GS19348 is obsolete.

Note: Optional bit 49 is not valid in a LE/370 runtime environment.
Part 2

This part of the table contains programs compiled under COBOL II and using LE runtime libraries. RHDCLEFE is not in use.

<table>
<thead>
<tr>
<th>Release</th>
<th>Method 1 (Most CPU efficient) Use single IGZCTCO per task</th>
<th>Method 2 (Special functionality) Use new IGZCTCO each TRANSFER</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.21PS and later</td>
<td>Not available</td>
<td>Default</td>
</tr>
</tbody>
</table>

Part 3

This part of the table contains programs compiled under COBOL II and using LE runtime libraries. RHDCLEFE is in use.

<table>
<thead>
<tr>
<th>Release</th>
<th>Method 1 (Most CPU efficient) Reuse same process/enclave</th>
<th>Method 2 (Special functionality) Use new process/enclave</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.21PS</td>
<td>Not available</td>
<td>Default</td>
</tr>
<tr>
<td>12.01</td>
<td>Not available</td>
<td>Default</td>
</tr>
<tr>
<td>14.0</td>
<td>Not available</td>
<td>Default</td>
</tr>
<tr>
<td>14.1 and later</td>
<td>Default</td>
<td>Optional bit 196.</td>
</tr>
</tbody>
</table>

Part 4

This part of the table contains programs compiled under LE COBOL and using LE runtime libraries.

<table>
<thead>
<tr>
<th>Release</th>
<th>Method 1 (Most CPU efficient) Reuse same process/enclave (see first note below)</th>
<th>Method 2 (Special functionality) Use new process/enclave (see first note below)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.21 PS</td>
<td>Not available</td>
<td>Default</td>
</tr>
<tr>
<td>12.01</td>
<td>Not available</td>
<td>Default</td>
</tr>
<tr>
<td>14.0</td>
<td>Default</td>
<td>Optional bit 196. (see second note below)</td>
</tr>
<tr>
<td>14.1 and later</td>
<td>Default</td>
<td>Optional bit 196.</td>
</tr>
</tbody>
</table>

⚠️ **Note:** When using RHDCLEFE with release 14.1 and later, the default is to preserve both the LE environment (process) and the LE enclave when invoking the same program multiple times in the same DC task. When not using RHDCLEFE, the environment is preserved for LE COBOL programs, but not the enclave.
In release 14.0, prior to maintenance level 9810, it is necessary to apply APAR LS40957 in order for optional bit 196 to have any effect. That APAR is automatically applied at level 9810 and above.

PSW Program Mask Settings

The program mask in the PSW controls whether or not certain arithmetic exceptions will cause a program check or be ignored. If the exception is ignored, significant digits of data may be lost. If the bit is on, the exception causes a program check. If the bit is off, the exception is ignored. The exceptions controlled by the program mask are as follows:

<table>
<thead>
<tr>
<th>PSW bit</th>
<th>Exception</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>Fixed-point Overflow</td>
</tr>
<tr>
<td>21</td>
<td>Decimal overflow</td>
</tr>
<tr>
<td>22</td>
<td>Exponent underflow</td>
</tr>
<tr>
<td>23</td>
<td>Significance</td>
</tr>
</tbody>
</table>

When the CA IDMS/DC/UCF system is in system mode (i.e., code in the system nucleus is executing), the program mask is always set to B'1110'. This enables a program check for all exceptions except significance exceptions.

A program mask of B'1110' is the default for initial entry into a user mode program. Some high level languages may change the program mask. For example, some versions of COBOL change the mask to B'0000'. The DC default is to honor such a change. The DC system does that by saving the program mask when a user-mode program makes a system request (for example, an OBTAIN or a GET STORAGE). While the request is being processed, the program mask is always set to B'1110'. When the system processing is completed, the program mask is restored before return to the user-mode program which made the request.

The default program mask settings can be modified through the use of options module RHDCOPTF. One option is to force the program mask to be set to B'1110' (the system default) upon return to a user-mode program after a system request as well as upon initial entry to the program. To effect this option, set OPT00253 in RHDCOPTF. This option will cause the default mask to be in effect at all times with one exception. The exception would be during the period after the user-mode program changes the mask until the next time it makes a system request. Note that this exception does not apply to COBOL II or LE COBOL programs. The COBOL run time code will always make several requests to the DC system for storage or other resources before the actual application code is entered. This assures that the default mask will be in effect when the application code is executed.

If OPT00253 is set, option bits 148 and 184 through 188 (described below) ignored.

The value of the program mask upon initial entry to a user-mode program can also be modified as described below:

- If OPT00184 is set and OPT00253 is not set in RHDCOPTF, then the value of the program mask on initial entry to a user mode program will be set based on #DEFOPTF bits 185-188 as follows:
  - If OPT00185 is set, fixed-point overflow exceptions will result in an interrupt (program check). When it is not set, fixed-point overflows will not result in an interrupt.
If OPT00186 is set, decimal overflow exceptions will result in an interrupt (program check). When it is not set, decimal overflows will not result in an interrupt.

If OPT00187 is set, exponent underflow exceptions will result in an interrupt (program check). When it is not set, exponent underflows will not result in an interrupt.

If OPT00188 is set, significance exceptions will result in an interrupt (program check). When it is not set, significance exceptions will not result in an interrupt.

If neither OPT00184 nor OPT00253 are set and OPT00148 is set, then the initial program mask will be set to binary 1010, i.e., fixed-point overflow and exponent underflow will cause an interrupt, but decimal overflow and significance exceptions will not. OPT00148 has no effect in release 16.0. The same functionality can be obtained by setting OPT00184, OPT00185, and OPT00187.

Note: the bit settings described above affect all user mode programs, not just COBOL programs. They are presented here because the optional settings are most commonly used for specialized COBOL applications.

Loading VS COBOL Programs into XA Storage

VS COBOL II and LE COBOL programs can and normally should be linked with AMODE 31 and RMODE ANY. Old-style VS COBOL programs, which run in batch, must run with AMODE 24 and RMODE 24. However, when running online VS COBOL programs in a DC/UCF region, it is permissible to run with AMODE 31 and RMODE 24. This is the normal recommended AMODE/RMODE setting for online VS COBOL program. This allows the WORKING STORAGE for VS COBOL programs to be allocated in XA storage. Since multiple copies of WORKING STORAGE may be allocated simultaneously (when multiple tasks are running that use the same program), this feature considerably reduces the amount of below-the-line storage required.

Some sites have a large number of COBOL programs and may want to link VS COBOL programs with AMODE 31 and RMODE ANY. This allows the programs to be loaded into the 31-bit (above-the-line) PROGRAM POOL. Caution should be used before utilizing this feature. If a program that is loaded above the line issues a COBOL verb that causes an illegal SVC to be issued or if the program is compiled with illegal compile options, the entire DC/UCF region may be abended. In some cases, the entire operating system may be abended. Illegal COBOL verbs and compile options are listed in Introduction to CA IDMS Data Manipulation Language (COBOL) (see page 21).

If online VS COBOL programs are to be linked RMODE(ANY) and run under Release 12.01 or earlier, an optional APAR must be applied. No optional APAR is required for release 14.0 and above, but the cautions listed above must be observed. The optional APARs are as follows:

<table>
<thead>
<tr>
<th>Release</th>
<th>APAR</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.21PS (MVS)</td>
<td>90-09-1003</td>
</tr>
<tr>
<td>10.21PS (VSE)</td>
<td>Not available</td>
</tr>
<tr>
<td>12.01 (MVS)</td>
<td>CS82390</td>
</tr>
<tr>
<td>12.01 (VSE)</td>
<td>GS53516</td>
</tr>
</tbody>
</table>
Online Debugger Syntax (COBOL)

This section contains the following articles:

- General Registers Symbols (COBOL) (see page 409)
- DC/UCF System Symbols (COBOL) (see page 409)
- Address Symbols and Markers (COBOL) (see page 410)
- User Symbols (COBOL) (see page 410)
- Program Symbols (COBOL) (see page 411)
- Expression Operators (COBOL) (see page 411)
- Delimiters (COBOL) (see page 411)
- Debugger Commands (COBOL) (see page 412)

General Registers Symbols (COBOL)

General registers include the registers used by the program at the time of execution and the registers used by the DC/UCF system. The program status word (PSW) and register definitions are always preceded by a colon (:) and are specified by these symbols:

- :PSW for the current program status word
- :Rn for the user program register at the time of interrupt, where \( n \) represents the number of the register and can have a value of 0 through 15
- :REGS for all user program registers at the time of interrupt
- :SRn for a DC/UCF system register at the time of interrupt, where \( n \) represents the number of the register and can have a value of 0 through 15
- :SREGS for all DC/UCF system registers at the time of interrupt

**Important!** A single debug expression can reference only one general register.

DC/UCF System Symbols (COBOL)

Certain DC/UCF system symbols also function as debugger entities, and you can refer to them during a debugging session. A colon (:) must precede each symbol. These are the valid symbols:

- :BAT
  Specifies the base address table for session.
- **:CSA**
  Specifies the DC/UCF common storage area.

- **:DLB**
  Specifies the debug local block, control block required for debugging session.

- **:LTE**
  Specifies the current logical terminal element.

- **:PTE**
  Specifies the current physical terminal element.

- **:TCE**
  Specifies the current task control element.

- **:VECT**
  Specifies the vector table for debugger.

**Important!** A single debug expression can reference only one system entity.

### Address Symbols and Markers (COBOL)

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Symbol Name</th>
<th>Designated Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>@</td>
<td>At sign</td>
<td>Absolute address</td>
</tr>
<tr>
<td>$</td>
<td>Dollar sign</td>
<td>Load address</td>
</tr>
<tr>
<td>¢</td>
<td>Cent sign</td>
<td>Address of current dialog process</td>
</tr>
</tbody>
</table>

### User Symbols (COBOL)

- **:DRn** for a debugger general register, where *n* represents the number of the register and can have a value of 0 through 15

- **:DREGS** for all debugger registers

- **:H1** and **:H2** for halfword 1 and halfword 2

- **:F1** and **:F2** for fullword 1 and fullword 2

- **:UCHR** for a 48-byte character area

You can also refer to specified sections of this area:

- **:UC0**, the first 16 bytes
- :UC16, the next 16 bytes
- :UC32, the last 16 bytes

Program Symbols (COBOL)

Syntax Data Field Names

Syntax Line Numbers

Syntax Qualifying Program Symbols

Expression Operators (COBOL)

<table>
<thead>
<tr>
<th>Operator</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>Addition</td>
</tr>
<tr>
<td>-</td>
<td>Subtraction</td>
</tr>
<tr>
<td>*</td>
<td>Multiplication</td>
</tr>
<tr>
<td>/</td>
<td>Division</td>
</tr>
</tbody>
</table>

Delimiters (COBOL)

<table>
<thead>
<tr>
<th>Delimiter</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>Asterisk</td>
</tr>
<tr>
<td></td>
<td>Blank</td>
</tr>
<tr>
<td>,</td>
<td>Comma</td>
</tr>
<tr>
<td>=</td>
<td>Equal sign</td>
</tr>
<tr>
<td>!</td>
<td>Exclamation point</td>
</tr>
</tbody>
</table>
### Delimiter

<table>
<thead>
<tr>
<th>Delimiter</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>Hyphen</td>
</tr>
<tr>
<td>%</td>
<td>Percent sign</td>
</tr>
<tr>
<td>.</td>
<td>Period</td>
</tr>
<tr>
<td>+</td>
<td>Plus sign</td>
</tr>
<tr>
<td>/</td>
<td>Slash</td>
</tr>
</tbody>
</table>

### Debugger Commands (COBOL)

This topic displays the syntax for the following online debugger commands:

- **AT** (see page 412)
- **DEBUG** (see page 413)
- **EXIT** (see page 413)
- **IOUSER** (see page 413)
- **LIST** (see page 413)
- **MENU** (see page 413)
- **PROMPT** (see page 413)
- **QUALIFY** (see page 414)
- **QUIT** (see page 414)
- **RESUME** (see page 414)
- **SET** (see page 414)
- **SNAP** (see page 414)
- **WHERE** (see page 415)

### AT

**ADD Format**

```
AT debug-expression
```

**INQUIRE Format**

```
AT ALL debug-expression
```

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>BEfore</strong></td>
<td>maximum execution-count</td>
</tr>
<tr>
<td><strong>MAXimum</strong></td>
<td>execution-count</td>
</tr>
<tr>
<td><strong>AFTER</strong></td>
<td>execution-count</td>
</tr>
<tr>
<td><strong>EVERy</strong></td>
<td>execution-count</td>
</tr>
<tr>
<td><strong>ON</strong></td>
<td>ignore</td>
</tr>
<tr>
<td><strong>OFF</strong></td>
<td>ignore</td>
</tr>
</tbody>
</table>

**Debug Expression:**
- **debug-expression**

**Execution Count:**
- **execution-count**

**ON**
- **ignore**

**OFF**
- **ignore**
DEBUG

ADD format

```
<table>
<thead>
<tr>
<th>DEBUG</th>
<th>PROgram</th>
<th>entity-name</th>
<th>VERSION version-number</th>
</tr>
</thead>
<tbody>
<tr>
<td>DIAlog</td>
<td>MAP</td>
<td>SS</td>
<td>TABle</td>
</tr>
</tbody>
</table>
```

INQUIRE format

```
<table>
<thead>
<tr>
<th>DEBUG</th>
<th>entity-name</th>
<th>VERSION version-number</th>
<th>INQuire OFF</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALL</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

EXIT

```
| EXIT |
```

IOUSER

```
| IOUser |
```

LIST

MEMORY Format

```
<table>
<thead>
<tr>
<th>List</th>
<th>Display</th>
<th>Memory begin-debug-expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>TO end-debug-expression</td>
<td>LENgth</td>
<td>byte-count-number</td>
</tr>
<tr>
<td>C</td>
<td>X</td>
<td>XC</td>
</tr>
</tbody>
</table>
```

ATTRIBUTES Format

```
| List | Display SESsion ATTRibutes |
```

MENU

```
| MEnu | screen-name |
```

PROMPT

```
| PROMpt |
```
QUALIFY

RESET Format

►►─── QUAlify ─┬──────────────────────┬─ PROCess process-name
│ DIAlog dialog-name └─ VERSion version-number

INQUIRE Format

►►─── QUAlify INQuire ──┬─

QUIT

►►─── QUIT ──┬─

RESUME

►►─── RESume ──┬─

SET

MEMORY Format

►►─┬─ Set ──┬─ Vary ─ Memory ─ debug-expression ─ EQUals ─
│ data-field-name ┤ C ┘─ RESET └─ NORest ┘
│ H halfword ────┘ X └─ X ─
│ F fullword ───┐ C ┘─
│ X hex-value ┘─XC └─
│ C character-string ─
│ P packed-value ─

ATTRIBUTES Format

►►─┬─ Set ──┬─ CHAr ─ HEXE ─ BOTH ─

SNAP

►►─┬─ SNAp ──┬─ TASk ─ begin-debug-expression ─
│ TO ─ end-debug-expression ─ byte-count-number ─ LENgth ┘
│ TITle title ┘
WHERE