

**UNISYS**

**A Series  
DMSII Interpretive  
Interface  
Programming  
Reference Manual**

Release 3.9.0

September 1991

Priced Item

U S America  
8600 0155-000

**UNISYS**

**A Series  
DMSII Interpretive  
Interface  
Programming  
Reference Manual**

Copyright © 1991 Unisys Corporation  
All rights reserved.  
Unisys is a registered trademark of Unisys Corporation.

Release 3.9.0

September 1991

Priced Item

U S America  
8600 0155-000

The names, places, and/or events used in this publication are not intended to correspond to any individual, group, or association existing, living, or otherwise. Any similarity or likeness of the names, places, and/or events with the names of any individual, living or otherwise, or that of any group or association is purely coincidental and unintentional.

**NO WARRANTIES OF ANY NATURE ARE EXTENDED BY THE DOCUMENT.** Any product and related material disclosed herein are only furnished pursuant and subject to the terms and conditions of a duly executed Program Product License or Agreement to purchase or lease equipment. The only warranties made by Unisys, if any, with respect to the products described in this document are set forth in such License or Agreement. Unisys cannot accept any financial or other responsibility that may be the result of your use of the information in this document or software material, including direct, indirect, special or consequential damages.

You should be very careful to ensure that the use of this information and/or software material complies with the laws, rules, and regulations of the jurisdictions with respect to which it is used.

The information contained herein is subject to change without notice. Revisions may be issued to advise of such changes and/or additions.

Correspondence regarding this publication may be forwarded using the Product Information card at the back of the manual, or may be addressed directly to Unisys, Product Information, 19 Morgan, Irvine, CA 92718.

# Page Status

Page	Issue
iii	-000
iv	Blank
v through xi	-000
xii	Blank
xiii through xvii	-000
xviii	Blank
xix	-000
xx	Blank
xxi through xxii	-000
1-1 through 1-9	-000
1-10	Blank
2-1 through 2-18	-000
3-1 through 3-79	-000
3-80	Blank
4-1 through 4-63	-000
4-64	Blank
5-1 through 5-10	-000
6-1 through 6-4	-000
7-1 through 7-8	-000
A-1 through A-17	-000
A-18	Blank
B-1 through B-20	-000
C-1 through C-31	-000
C-32	Blank
D-1 through D-8	-000
Glossary-1 through 21	-000
Glossary-22	Blank
Bibliography-1 through 3	-000
Bibliography-4	Blank
Index-1 through 21	-000
Index-22	Blank

Unisys uses an 11-digit document numbering system. The suffix of the document number (1234 5678-xyz) indicates the document level. The first digit of the suffix (x) designates a revision level; the second digit (y) designates an update level. For example, the first release of a document has a suffix of -000. A suffix of -130 designates the third update to revision 1. The third digit (z) is used to indicate an errata for a particular level and is not reflected in the page status summary.



# About This Manual

## Purpose

The *A Series DMSII Interpretive Interface Programming Reference Manual* describes how to code effective and efficient application programs that access and manipulate a Data Management System II (DMSII) database, using the DMSII interpretive interface.

## Scope

This manual details how to code DMINTERPRETER library entry points and how to generate the DMINTERPRETER library.

## Audience

This manual has two primary audiences: application programmers who are responsible for programs that report or update information in a DMSII database and database administrators who generate the DMINTERPRETER library.

## Prerequisites

Before programming DMSII applications, you should be familiar with

- Basic A Series concepts
- Basic DMSII concepts
- The Data and Structure Definition Language (DASDL)
- Libraries and entry points
- The programming languages you intend to use

You should also know how to use the A Series Command and Edit (CANDE) message control system (MCS) to enter code and know how to use CANDE and the Work Flow Language (WFL) to compile and run jobs.

# How to Use This Manual

This manual is designed as a reference to help you code effective and efficient DMSII application programs that use the interpretive interface.

If you are new to the interpretive interface, read Section 1 for an overview of the interface. Next, read Section 2 for an explanation of how to access the library and entry points in your application program. Then read Sections 3 through 6 for information on coding entry points.

If you are an experienced DMSII interpretive interface programmer, you can go directly to the entry point explanations in Sections 3 through 6. The entry points are divided by task:

- Section 3 covers tasks performed by standard entry points.
- Section 4 deals with data transfer tasks.
- Section 5 focuses on exception-handling tasks.
- Section 6 explains attribute setting.

Use either the table of contents or the index to locate a specific task or entry point.

If you are writing a general-purpose program that can access any database and is data independent, read Section 7 for information on how to use DBSTRUCTURE, an intrinsic, read-only data set.

While Section 5 presents coding guidelines for handling exceptions and error conditions within your application program, Appendix B details the actual exception and error messages returned by the Accessroutines to your application program.

Database administrators and programmers who are responsible for the DMINTERPRETER library should read Appendix A carefully before generating the library.

Although the interpretive interface can be used by any programming language that can import library objects, only ALGOL, COBOL74, and FORTRAN77 program fragments are used for text examples. The samples of complete programs, in Appendix C, are also given only for ALGOL, COBOL74, and FORTRAN77. If you are programming in another language, such as C, COBOL, COBOL85, Pascal, or PL/1, consult the language reference manual.

The separate COBOL languages (COBOL, COBOL74, and COBOL85) are referred to collectively in the text, index, and glossary as COBOL. However, if there are any differences, the specific ANSI-version of the language is identified.

To distinguish interpretive interface entry points from language extensions, the text notes the entry points by their COBOL name in the DATABASE/DMINTERPRETER file. For example, while both DBFIND and FIND refer to the find operation, DBFIND is the entry point and FIND is the command.

The entry points for transferring data from the user work area to user-declared variables are referred to collectively as DBGET entry points. The entry points for transferring data from user-declared variables to the user work area are referred to collectively as DBPUT entry points.

Metatokens are used for describing syntactical elements. For example, the metatoken < data set qualified name > refers to the syntax of a data set name. Syntax diagrams and syntax elements are noted in the index under the appropriate metatoken. An explanation of railroad diagrams (a Unisys method of depicting syntax) is found in Appendix D.

This manual uses the word *access* as both a noun and a verb.

- As a noun, Access always appears with a capital A and refers to a database part.
- As a verb, access means to use or retrieve something.

This manual also uses the word *recreate* as both a verb and a DMSII operation.

- As a verb, re-create always appears with a hyphen.
- As a DMSII operation, recreate always appears without the hyphen.

Throughout the document, *A Series* is used to refer collectively to A Series and B 7900 systems. Unless otherwise noted, all references to related documentation within the text are for A Series product information. These references are included in the bibliography. All acronyms used in the guide, including those used in documentation titles, are spelled out and defined in the glossary.

For a broader view of programming DMSII applications, you can also do the following:

- Consult the *A Series DMSII Technical Overview* for background information and for an explanation of designing and maintaining databases.
- Read the *A Series DMSII Application Program Interfaces Programming Guide* for general guidelines and for an introduction to both the interpretive interface and the language extensions interface.
- Read the *A Series DMSII Data and Structure Definition Language (DASDL) Programming Reference Manual* for DASDL option specifics.
- Refer to the *A Series DMSII Utilities Operations Guide* for detailed information on the use of DMSII programs.

## Organization

This manual is organized into seven sections. Each section explores topics and tasks that pertain to the DMSII interpretive interface. In addition, four appendixes, a glossary, a bibliography, and an index appear at the end of the guide.

### Section 1. Understanding the DMSII Interpretive Interface

Section 1 overviews the basic structure and advantages of the interpretive interface. It introduces the major components of the interface and presents programming



considerations, recommendations, and guidelines that pertain to the interpretive interface only.

This section also overviews the four types of entry points provided by the DMINTERPRETER library: the standard, data transfer, exception handling, and attribute setting entry points.

### **Section 2. Accessing the Interpretive Interface**

Section 2 explains how, for each supported user language, an application program can access both a DMINTERPRETER library and its task-specific entry points, and then invoke the entry points.

### **Section 3. Manipulating the Database**

Section 3 details the function and coding of each standard entry point to the DMINTERPRETER library. These entry points perform functions comparable to those performed by language extensions in manipulating the database, such as opening the database, finding records, creating new records, changing records, and storing records.

### **Section 4. Transferring Data**

Section 4 details how the user areas are made available and how each data transfer entry point to the DMINTERPRETER library is coded.

### **Section 5. Handling Exceptions**

Section 5 details the function and coding of each exception handling entry point to the DMINTERPRETER library.

### **Section 6. Restricting Calls to the Accessroutines**

Section 6 details the function and coding of the attribute setting entry point to the DMINTERPRETER library, the entry point that restricts the number of calls to the Accessroutines.

### **Section 7. Determining the Database Structure**

Section 7 discusses how to use the data set DBSTRUCTURE to determine the structure of the database at run time.

### **Appendix A. Generating the DMINTERPRETER Library**

Appendix A details how to generate the DMINTERPRETER library either through the Work Flow Language (WFL) or interactively through the program BUILDINQ. The library can interface with an entire physical database, a logical database, or selected parts of a database.

### **Appendix B. DMSII Exceptions and Errors**

Appendix B lists the exceptions and errors returned to an application program by the Accessroutines.

### **Appendix C. Sample Programs That Use the Interpretive Interface**

Appendix C contains the required DASDL description and the listing for ALGOL, COBOL74, and FORTRAN77 application programs that demonstrate possible coding techniques for programs that use the interpretive interface.

### **Appendix D. Understanding Railroad Diagrams**

Appendix D explains how to read railroad diagrams.

## **Related Product Information**

The following list contains companion Unisys documents for this guide.

### ***A Series ALGOL Programming Reference Manual, Volume 1: Basic Implementation (form 8600 0098)***

This manual describes the basic features of the Extended ALGOL programming language. This manual is written for programmers who are familiar with programming concepts.

### ***A Series ALGOL Programming Reference Manual, Volume 2: Product Interfaces (form 8600 0734)***

This manual describes the extensions to the Extended ALGOL language that allow application programs to use the Advanced Data Dictionary System (ADDS), the Communications Management System (COMS), the Data Management System II (DMSII), the Screen Design Facility Plus (SDF Plus), or the Semantic Information Manager (SIM). This manual is written for programmers who are familiar with Extended ALGOL programming language concepts and terms.

### ***A Series ALGOL Test and Debug System (TADS) Programming Guide (form 1169539)***

This guide describes the features of ALGOL TADS, an interactive tool used for testing and debugging ALGOL programs and libraries. ALGOL TADS allows the programmer to monitor and control the execution of programs under test and examine the data at any given point during program execution. This guide is written for programmers who are familiar with ALGOL programming language concepts and terms.

### ***A Series C Programming Reference Manual (form 3950 8775)***

This manual describes the C programming language, including the A Series extensions. Where the implementation of A Series C differs from the proposed draft ANSI C standard, the differences are noted in the manual.

## About This Manual

---

### ***A Series COBOL ANSI-68 Programming Reference Manual (form 8600 0320)***

This manual provides a complete description of COBOL as developed by the CODASYL Committee and described by the American National Standards Institute in X3.23-1968. This manual is written for programmers who are familiar with programming concepts.

### ***A Series COBOL ANSI-74 Programming Reference Manual, Volume 1: Basic Implementation (form 8600 0296)***

This manual describes the basic features of the standard COBOL ANSI-74 programming language, which is fully compatible with the American National Standard, X3.23-1974. This manual is written for programmers who are familiar with programming concepts.

### ***A Series COBOL ANSI-74 Programming Reference Manual, Volume 2: Product Interfaces (form 8600 0130)***

This manual describes the extensions to the standard COBOL ANSI-74 language. These extensions are designed to allow application programs to interface with the Advanced Data Dictionary System (ADDs), the Communications Management System (COMs), the Data Management System II (DMSII), the DMSII Transaction Processing System (TPS), the Screen Design Facility (SDF), the Screen Design Facility Plus (SDF Plus), and Semantic Information Manager (SIM) products. This manual is written for programmers who are familiar with COBOL74 programming language concepts and terms.

### ***A Series COBOL ANSI-85 Programming Reference Manual, Volume 1: Basic Implementation (form 8600 1518)***

This manual describes the basic features of the COBOL ANSI-85 programming language, as implemented on Unisys A Series systems. This manual is written for programmers who are familiar with programming concepts.

### ***A Series COBOL ANSI-85 Programming Reference Manual, Volume 2: Product Interfaces (form 8600 1526)***

This manual describes the extensions to the standard COBOL85 language. These extensions are designed to allow application programs to interface with the Communications Management System (COMs) and Data Management System II (DMSII) products. This manual is written for programmers who are familiar with COBOL85 programming language concepts and terms.

### ***A Series DMSII Application Program Interfaces Programming Guide (form 5044225). Formerly A Series DMSII User Language Interface Programming Guide***

This guide explains how to write effective and efficient application programs that access and manipulate a Data Management System II (DMSII) database using either the DMSII interpretive interface or the DMSII language extensions. This guide is written for application programmers and database administrators who are already familiar with the basic concepts of DMSII.

***A Series DMSII Data and Structure Definition Language (DASDL)  
Programming Reference Manual (form 8600 0213)***

This manual provides instructions for defining and maintaining a Data Management System II (DMSII) database using DASDL. This manual is written for database administrators and staff.

***A Series FORTRAN77 Programming Reference Manual (form 3950 8759)***

This manual describes the FORTRAN 77 programming language, which is fully compatible with the American National Standard X3.9-1978. This manual is written for programmers who are familiar with programming concepts.

***A Series FORTRAN77 Test and Debug System (TADS) Programming Guide  
(form 1222667)***

This guide describes an interactive tool for testing and debugging FORTRAN77 programs and libraries. This manual is written for programmers who are familiar with FORTRAN77 programming language concepts and terms.

***A Series Task Attributes Programming Reference Manual (form 8600 0502).  
Formerly A Series Work Flow Administration and Programming Guide***

This manual describes all the task attributes available on A Series systems. It also gives examples of statements for reading and assigning task attributes in various programming languages. The *A Series Task Management Programming Guide* is a companion manual.

***A Series Task Management Programming Guide (form 8600 0494). Formerly  
Work Flow Administration and Programming Guide***

This guide explains how to initiate, monitor, and control processes on an A Series system. It describes process structures and process family relationships, introduces the uses of many task attributes, and gives an overview of interprocess communication techniques. The *A Series Task Attributes Programming Reference Manual* is a companion manual.



# Contents

About This Manual .....	v
<b>Section 1. Understanding the DMSII Interpretive Interface</b>	
<b>Compilers That Can Use the Interpretive Interface .....</b>	<b>1-1</b>
<b>Interpretive Interface Run-Time and Compile-Time Interfaces ...</b>	<b>1-1</b>
<b>Using the Run-Time Libraries .....</b>	<b>1-4</b>
Reporting Errors through the DMSUPPORT Library ...	1-4
Accessing a Database through the DMINTERPRETER Library .....	1-4
Identifying the Four Types of DMINTERPRETER Entry Points .....	1-4
Accessing DMINTERPRETER Library Entry Points For ALGOL and FORTRAN77 Programs ...	1-5
For COBOL Programs .....	1-6
Renaming Entry Points within an Application Program .....	1-6
Using Parameters in Entry Points .....	1-7
Returning Exceptions .....	1-7
<b>Programming Considerations for the Interpretive Interface Only .</b>	<b>1-7</b>
Sequencing Entry Points .....	1-8
Reducing Run-Time Overhead .....	1-8
<b>Section 2. Accessing the Interpretive Interface</b>	
<b>General Guidelines for Accessing the Interpretive Interface ....</b>	<b>2-1</b>
<b>Accessing the Interpretive Interface in ALGOL Programs .....</b>	<b>2-2</b>
Declaring a DMINTERPRETER Library in an ALGOL Program .....	2-2
Declaring ALGOL Entry Points as Boolean Procedures .	2-3
Using the Release Tape to Declare ALGOL Entry Points .....	2-4
Using a Procedure Declaration for ALGOL Entry Points .....	2-5
Declaring a Subset of ALGOL Entry Points .	2-5
Renaming ALGOL Entry Points .....	2-6
Coding ALGOL Procedure Declarations ....	2-6
Invoking ALGOL Entry Points .....	2-7
<b>Accessing the Interpretive Interface in COBOL Programs .....</b>	<b>2-8</b>
Exported Names of COBOL Entry Points on the Release Tape .....	2-8
Using the COBOL CHANGE Construct to Invoke a DMINTERPRETER Library .....	2-9
Invoking COBOL Entry Points .....	2-11
Passing Parameters with the USING Clause in COBOL .....	2-11

## Contents

---

Exception Handling with the GIVING Clause in COBOL .....	2-12
<b>Accessing the Interpretive Interface in FORTRAN77 Programs . .</b>	<b>2-12</b>
Declaring a DMINTERPRETER Library in FORTRAN77 Programs .....	2-13
Declaring FORTRAN77 Entry Points as Logical Functions .....	2-14
Using the Release Tape to Declare FORTRAN77 Entry Points .....	2-14
Using the Logical Function for FORTRAN77 Entry Points .....	2-16
Declaring a Subset of the FORTRAN77 Entry Points .....	2-16
Coding the FORTRAN77 Library Entry Point Declaration .....	2-16
Renaming FORTRAN77 Entry Points .....	2-17
Coding a FORTRAN77 IN LIBRARY Statement .....	2-17
Invoking FORTRAN77 Entry Points .....	2-18
<b>Section 3. Manipulating the Database</b>	
<b>Guidelines for Using Standard Entry Points .....</b>	<b>3-3</b>
<b>Opening a Database .....</b>	<b>3-6</b>
<b>Creating Reserve Space for New Records .....</b>	<b>3-9</b>
<b>Setting the Current Path .....</b>	<b>3-13</b>
<b>Explicitly Freeing Records .....</b>	<b>3-17</b>
<b>Explicitly Freeing Structures .....</b>	<b>3-21</b>
<b>Finding Records .....</b>	<b>3-24</b>
<b>Locking Records .....</b>	<b>3-29</b>
<b>Locking Structures .....</b>	<b>3-34</b>
<b>Securing Records .....</b>	<b>3-37</b>
<b>Securing Structures .....</b>	<b>3-42</b>
<b>Begin Transaction – Entering Transaction State .....</b>	<b>3-45</b>
<b>Aborting Transactions .....</b>	<b>3-48</b>
<b>Saving Transaction Points .....</b>	<b>3-50</b>
<b>Canceling Transactions Back to Savepoints .....</b>	<b>3-53</b>
<b>Deleting Data Records .....</b>	<b>3-56</b>
<b>Re-creating Records .....</b>	<b>3-61</b>
<b>Storing Records .....</b>	<b>3-66</b>
<b>End Transaction – Leaving Transaction State .....</b>	<b>3-69</b>
<b>Closing a DMSII Database .....</b>	<b>3-72</b>
<b>Executing Language Extensions .....</b>	<b>3-74</b>
<b>Section 4. Transferring Data</b>	
<b>Guidelines for Using Data Transfer Entry Points .....</b>	<b>4-3</b>
<b>Moving Character Strings to Variables .....</b>	<b>4-6</b>
<b>Moving Kanji Alpha Character Strings to Variables .....</b>	<b>4-11</b>
<b>Moving Numeric Values to Variables .....</b>	<b>4-16</b>
<b>Moving Double-Precision Values to Variables .....</b>	<b>4-21</b>

Retrieving Boolean Values . . . . .	4-26
Placing Strings into Data Items . . . . .	4-31
Placing Strings into Kanji Alpha Items . . . . .	4-36
Placing Numeric Values into Data Items . . . . .	4-41
Placing Double-Precision Values into Data Items . . . . .	4-46
Setting Data Items to Boolean Values . . . . .	4-51
Setting Data Items to Null Values . . . . .	4-56
Constructing Data Transfers during Program Execution . . . . .	4-60
<b>Section 5. Handling Exceptions</b>	
Returning the Exception Word . . . . .	5-3
Returning the Text of an Exception Message . . . . .	5-5
Identifying the Type of Exception . . . . .	5-8
<b>Section 6. Restricting Calls to the Accessroutines</b>	
<b>Section 7. Determining the Database Structure</b>	
Generating the DBSTRUCTURE Data Set . . . . .	7-1
Accessing the DBSTRUCTURE Data Set . . . . .	7-1
Describing the Structure and Contents of DBSTRUCTURE . . . . .	7-2
<b>Appendix A. Generating the DMINTERPRETER Library</b>	
Capabilities of the BUILDINQ Program . . . . .	A-1
Files Associated with a DMINTERPRETER Library . . . . .	A-1
Ensuring Directory-Level Consistency between Files . . . . .	A-4
Generating the DMINTERPRETER Library Interactively . . . . .	A-4
Remote Generation Using the ZIP Option . . . . .	A-5
Step 1. Initiating the BUILDINQ Program . . . . .	A-5
Step 2. Selecting the Database . . . . .	A-5
Step 3. Selecting a View of the Database . . . . .	A-6
Step 4. Selecting Data Sets . . . . .	A-10
Step 5. Entering the Name of a Logical Database . . . . .	A-11
Step 6. Determining the Compiled Access Mode for the DMINTERPRETER Library . . . . .	A-11
Step 7. Renaming Hyphenated Logical Database Names . . . . .	A-11
Step 8. Entering a Database Name for the DMINTERPRETER Code File . . . . .	A-12
Step 9. Entering the Compilation Queue . . . . .	A-12
Step 10. Displaying the Verification Message . . . . .	A-12
Remote Generation Using the NOZIP Option . . . . .	A-13
Selecting the NOZIP Option . . . . .	A-13
Compiling a DMINTERPRETER Library Where NOZIP Is Set . . . . .	A-14



## Contents

---

Generating the DMINTERPRETER Library from WFL Job Decks .	A-15
---	------

### Appendix B. DMSII Exceptions and Errors

Categorizing Exceptions and Errors .....	B-2
ABORT Category .....	B-2
AUDITERROR Category .....	B-3
CLOSEERROR Category .....	B-4
DATAERROR Category .....	B-4
DEADLOCK Category .....	B-5
DUPLICATES Category .....	B-5
FATALERROR Category .....	B-6
INTEGRITYERROR Category .....	B-6
INTLIBERROR Category .....	B-6
INUSE Category .....	B-8
IOERROR Category .....	B-9
KEYCHANGED Category .....	B-9
LIMITERROR Category .....	B-10
NORECORD Category .....	B-10
NOTFOUND Category .....	B-11
NOTLOCKED Category .....	B-12
OPENERROR Category .....	B-12
READONLY Category .....	B-15
SECURITYERROR Category .....	B-16
SYSTEMERROR Category .....	B-16
VERSIONERROR Category .....	B-18
Summarizing the Relationship of Data Management Operations to Exceptions and Errors .....	B-19

### Appendix C. Sample Programs That Use the Interpretive Interface

DASDL Database for Sample Programs .....	C-2
ALGOL Application Program Using the Interpretive Interface ...	C-5
COBOL74 Application Program Using the Interpretive Interface .	C-15
FORTRAN77 Application Program Using the Interpretive Interface .....	C-25

### Appendix D. Understanding Railroad Diagrams

What Are Railroad Diagrams? .....	D-1
Constants and Variables .....	D-2
Constraints .....	D-2
Following the Paths of a Railroad Diagram .....	D-5
Railroad Diagram Examples with Sample Input .....	D-6

Glossary .....	1
----------------	---

**Bibliography** ..... 1

**Index** ..... 1



# Figures

1-1.	Database Processing with the Interpretive Interface . . . . .	1-3
3-1.	Standard Entry Point Operations In and Out of Transaction State . . . . .	3-5
4-1.	Three Steps in Transferring Data . . . . .	4-4
A-1.	Relationship of BUILDINQ Files . . . . .	A-3
A-2.	Compiling a DMINTERPRETER Library for the Total Database . . . . .	A-7
A-3.	Compiling a DMINTERPRETER Library for Selected Data Sets . . . . .	A-8
A-4.	Compiling a DMINTERPRETER Library for a Logical Database . . . . .	A-9
A-5.	Using the BUILDINQ Program with the NOZIP Option . . . . .	A-14
B-1.	Relationship of Data Management Operations to Errors and Exceptions . .	B-20
D-1.	Railroad Constraints . . . . .	D-5



# Tables

1-1.	Purpose of Each Type of Entry Point .....	1-5
2-1.	ALGOL Entry Points .....	2-4
2-2.	COBOL, COBOL74, and COBOL85 Entry Points .....	2-8
2-3.	FORTTRAN77 Entry Points .....	2-15
3-1.	Standard Entry Points to the DMINTERPRETER Library by Task .....	3-1
3-2.	Opening a Database .....	3-6
3-3.	Creating Reserve Space for New Records .....	3-9
3-4.	Setting the Current Path .....	3-13
3-5.	Explicitly Freeing Records .....	3-17
3-6.	Explicitly Freeing Structures .....	3-21
3-7.	Finding Records .....	3-25
3-8.	Locking Records .....	3-29
3-9.	Locking Structures .....	3-34
3-10.	Securing Records .....	3-37
3-11.	Securing Structures .....	3-42
3-12.	Begin Transaction—Entering Transaction State .....	3-45
3-13.	Export Names of Entry Points Used to Abort a Transaction .....	3-48
3-14.	Saving Transaction Points .....	3-50
3-15.	Canceling Transactions Back to Savepoints .....	3-53
3-16.	Deleting a Data Record .....	3-57
3-17.	Re-creating a Record .....	3-62
3-18.	Storing Records .....	3-66
3-19.	End Transaction—Leaving Transaction State .....	3-69
3-20.	Export Names of Entry Points Used to Close a Database .....	3-72
3-21.	Executing Language Extensions .....	3-74
4-1.	Data Entry Points to the DMINTERPRETER Library by Task .....	4-1
4-2.	Moving Character Strings to Variables .....	4-6
4-3.	Moving Kanji Character Strings to Variables .....	4-11
4-4.	Moving Numeric Values to Variables .....	4-17
4-5.	Moving Double-Precision Values to Variables .....	4-21
4-6.	Retrieving Boolean Values .....	4-26
4-7.	Placing Strings into Data Items .....	4-31
4-8.	Placing Strings into Kanji Alpha Items .....	4-36
4-9.	Placing Numeric Values into Data Items .....	4-41
4-10.	Placing Double-Precision Values into Data Items .....	4-46
4-11.	Setting Data Items to Boolean Values .....	4-51
4-12.	Setting Data Items to Null Values .....	4-56
4-13.	Constructing Transfers during Program Execution .....	4-60
5-1.	Exception Handling Entry Points to the DMINTERPRETER Library by Task .....	5-2
5-2.	Returning an Exception Word .....	5-3
5-3.	Returning the Text of an Exception Message .....	5-6
5-4.	Identifying the Type of Exception .....	5-9

## Tables

---

6-1.	Attribute Setting Entry Points—Restricting Calls to Accessroutines . . . . .	6-2
7-1.	DBSTRUCTURE Items . . . . .	7-3
7-2.	DBSTRUCTURE Items for a DB-TYPE of DATASET . . . . .	7-4
7-3.	DBSTRUCTURE Items for a DB-TYPE of SET . . . . .	7-5
7-4.	DBSTRUCTURE Items for a DB-TYPE of ITEM . . . . .	7-6
7-5.	DBSTRUCTURE Items for a DB-TYPE of LINK . . . . .	7-8
A-1.	Files Associated with a DMINTERPRETER Library . . . . .	A-2
A-2.	WFL Cards Used to Generate a DMINTERPRETER Library . . . . .	A-15
B-1.	Major Categories for Exceptions and Errors by Category Number . . . . .	B-2
B-2.	ABORT Subcategories . . . . .	B-3
B-3.	AUDITERROR Subcategories . . . . .	B-3
B-4.	CLOSEERROR Subcategories . . . . .	B-4
B-5.	DATAERROR Subcategories . . . . .	B-4
B-6.	DEADLOCK Subcategories . . . . .	B-5
B-7.	DUPLICATES Subcategories . . . . .	B-6
B-8.	INTEGRITYERROR Subcategories . . . . .	B-6
B-9.	INTLIBERROR Subcategories . . . . .	B-7
B-10.	INUSE Subcategories . . . . .	B-8
B-11.	IOERROR Subcategories . . . . .	B-9
B-12.	KEYCHANGED Subcategories . . . . .	B-10
B-13.	LIMITERROR Subcategories . . . . .	B-10
B-14.	NORECORD Subcategories . . . . .	B-11
B-15.	NOTFOUND Subcategories . . . . .	B-11
B-16.	NOTLOCKED Subcategories . . . . .	B-12
B-17.	OPENERROR Subcategories . . . . .	B-13
B-18.	READONLY Subcategories . . . . .	B-16
B-19.	SECURITYERROR Subcategories . . . . .	B-16
B-20.	SYSTEMERROR Subcategories . . . . .	B-16
B-21.	VERSIONERROR Subcategories . . . . .	B-18

# Section 1

## Understanding the DMSII Interpretive Interface

There are two DMSII application program interfaces:

- The DMSII language extensions
- The DMSII interpretive interface

The language extensions are explained in Volume 2 of the ALGOL, COBOL, COBOL74, COBOL85, and RPG reference manuals. The interpretive interface is detailed in this manual. Consult the *DMSII Application Programming Guide* for a comparison of the language extensions and the interpretive interface.

This section describes the major features of the interpretive interface.

### Compilers That Can Use the Interpretive Interface

Only a standard language compiler that can import library objects can use the interpretive interface. For A Series systems, this restriction means that application programs written for use with the following compilers are able to use the interpretive interface:

ALGOL	COBOL74	Pascal
C	COBOL85	PL/1
COBOL	FORTAN77	RPG

### Interpretive Interface Run-Time and Compile-Time Interfaces

The interpretive interface uses the standard language compiler as its compile-time interface and the Accessroutines as its run-time interface. The standard language compiler is responsible for producing the error-free object code file for your application program. The Accessroutines communicate with the database, allowing you to write and compile application programs before the database is compiled. The application program never directly accesses the database. This arrangement means that

- A single application program, through the library mechanism, can invoke as many databases as desired. (A DMINTERPRETER library must be compiled for each database.)
- You can use the open operation to create general-purpose programs that can access any database.



## Understanding the DMSII Interpretive Interface

---

- You can postpone decisions about database operations until run time.
- Programs can use the intrinsic data set DBSTRUCTURE to interrogate the structure of a database at run time.

Figure 1-1 shows the modules and timing involved in compiling an application program that uses the interpretive interface.

# Understanding the DMSII Interpretive Interface

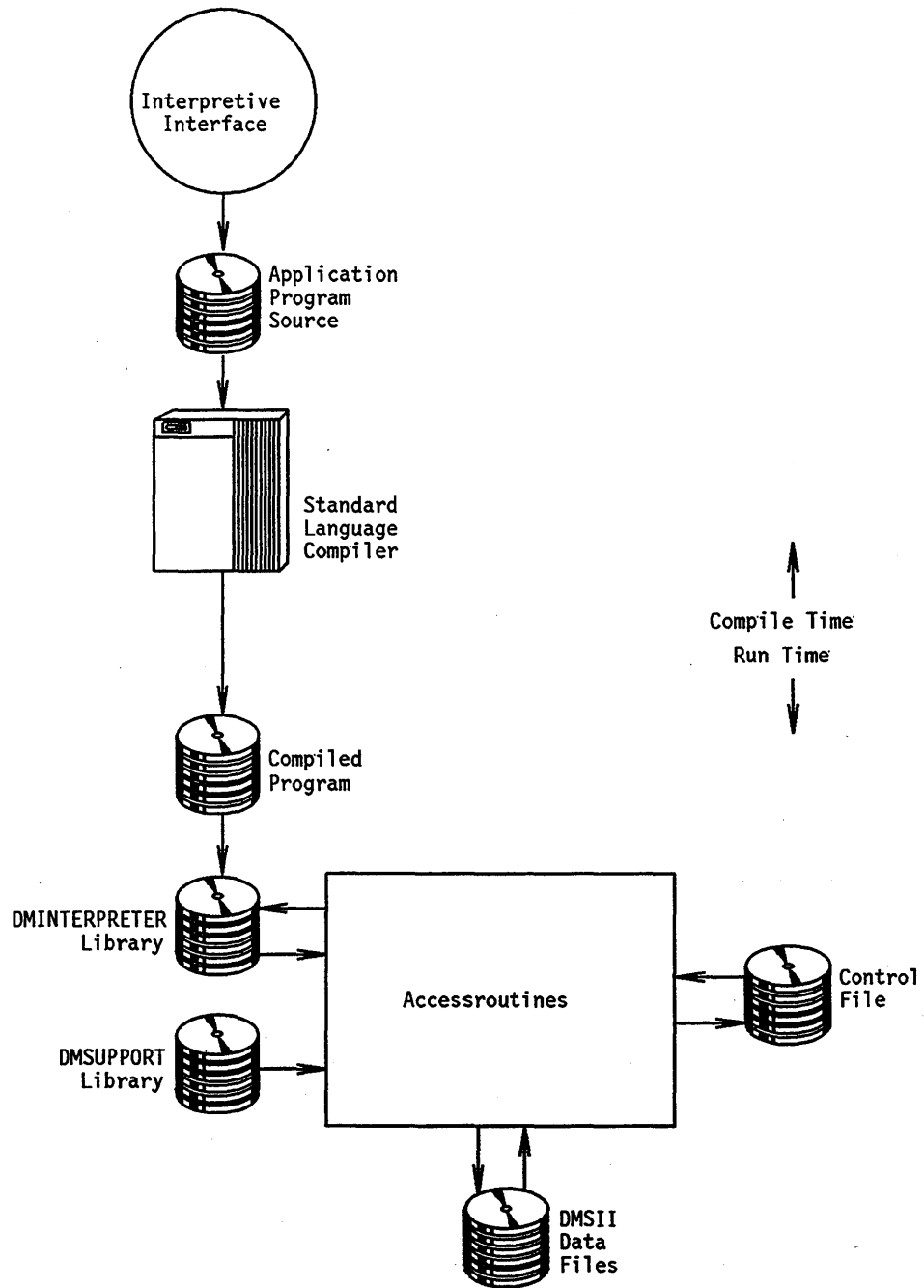


Figure 1-1. Database Processing with the Interpretive Interface

### Using the Run-Time Libraries

The interpretive interface uses two run-time libraries: the DMSUPPORT library and the DMINTERPRETER library.

- The DMSUPPORT library provides entry points that allow you to obtain DMSII error and exception codes.
- The DMINTERPRETER library provides entry points that allow your program to access the database.

### Reporting Errors through the DMSUPPORT Library

The DMSUPPORT library is a general support library used with all DMSII software. The library provides error messages and exception handling for all DMSII software.

For more information about exception handling through entry points, read Section 5, "Handling Exceptions." For details about the exception and error messages, read Appendix B, "DMSII Exceptions and Errors" and the *DMSII Application Programming Guide*. For a details of the DMSUPPORT library, consult the *DMSII Utilities Operations Guide*.

### Accessing a Database through the DMINTERPRETER Library

The DMINTERPRETER library provides entry points that allow application programs to access, manage, and manipulate a DMSII database without executing DMSII language extensions. Section 2, "Accessing the Interpretive Interface," explains how to access the library and the entry points.

Each entry point is a procedure in a library program (known as a library) that can be called by another program through a process known as *exporting*. A library exports entry points, and the calling program imports entry points. Exporting allows the library procedure to be accessed by programs that are external to the library. Sections 3 through 6 detail the DMINTERPRETER library entry points.

Each database has its own DMINTERPRETER library written in DMALGOL. Using the program BUILDINQ, the database administrator generates the library after compiling the DASDL description for the database. Read Appendix A, "Generating the DMINTERPRETER Library," for the details of executing BUILDINQ.

### Identifying the Four Types of DMINTERPRETER Entry Points

The interpretive interface uses four types of entry points: standard, data transfer, exception handling, and attribute setting. Each type performs a specific function, as shown in Table 1-1.

**Table 1-1. Purpose of Each Type of Entry Point**

Type	Purpose	Detailed in
<b>Standard</b>	Performs functions that correspond to functions specified in DMSII language extensions. For example, both interfaces provide a means of opening and closing a database.	Section 3, "Manipulating the Database"
<b>Data transfer</b>	Moves data between the database and the user work area, and between the user work area and the program.	Section 4, "Transferring Data"
<b>Exception handling</b>	Returns result of a call. The result reports whether the call was successful and, if it was not successful, the category, subcategory, and structure number associated with the exception.	Section 5, "Handling Exceptions"
<b>Attribute setting</b>	Restricts the number of times the DMINTERPRETER library can call the Accessroutines to complete a find operation.	Section 6, "Restricting Calls to the Accessroutines"

### Accessing DMINTERPRETER Library Entry Points

DATABASE/DMINTERPRETER, the DMINTERPRETER library symbolic file, contains declarations of library entry points for each language. Depending on which user language you use to code your program, you must use different procedures and syntax to access (import) the exported declarations.

#### For ALGOL and FORTRAN77 Programs

If you are coding either an ALGOL and FORTRAN77 program, you must follow a three-step process to access entry points:

1. Declare the library.
2. Declare the entry points.
3. Invoke the entry points.

Once the DMINTERPRETER library for the database is established, declare the entry points either by

- Using the \$INCLUDE compiler control option to declare all the entry points
- Declaring each entry point individually

If you use the \$INCLUDE option, all the entry points are exported. If you declare entry points individually, you can create your own subset of entry points and you can rename the entry points.

## Understanding the DMSII Interpretive Interface

---

Once an entry point is declared, it can be invoked (called) to perform its run-time function.

Read Section 2, "Accessing the Interpretive Interface," for detailed explanations of how to access the library and how to use the \$INCLUDE compiler control option.

### For COBOL Programs

If you are coding programs in any supported COBOL language, you must follow a two-step process to access entry points:

1. Declare the library.
2. Invoke all the entry points.

Once the library is established, the entry points can be invoked; they are not declared.

Read Section 2, "Accessing the Interpretive Interface," for a detailed explanation of how to access the library and how to invoke the entry points.

### Renaming Entry Points within an Application Program

The DMINTERPRETER library symbolic file, DATABASE/DMINTERPRETER, contains the names by which the interpretive interface expects the entry points to be called. These are the exported names. DATABASE/DMINTERPRETER is part of the release tape.

ALGOL exported entry point names begin with ALGOL, for instance, ALGOLFIND. FORTRAN77 exported entry point names begin with FORTRAN77, such as FORTRAN77FIND. By using the ACTUALNAME clause of their library declaration, ALGOL and FORTRAN77 application programs can rename exported entry points within the program.

Refer to the Section 2, "Accessing the Interpretive Interface," for the specifics of renaming entry points.

COBOL exported entry point names always begin with the prefix DB, for example, DBFIND. Application programs written in any of the supported COBOL languages cannot rename entry points.

### Example of Renaming an Entry Point

In this ALGOL program fragment, ALGOLFIND is renamed as MYFIND. MYFIND is an internal name that is used only within the application program. The name of the entry point in the DMI library remains ALGOLFIND.

```
LIBRARY DMI;  
BOOLEAN PROCEDURE MYFIND(DIRECTION,STRUCTURE,CONDITION);  
  STRING DIRECTION, STRUCTURE, CONDITION;  
LIBRARY DMI (ACTUALNAME = "ALGOLFIND");
```

### Using Parameters in Entry Points

Regardless of the application language, each entry point uses the same number of parameters, has the same required parameters, and requires input in the same order. For instance, the entry point to move a numeric value to a variable has three required parameters. In all languages, you enter the name of the structure first, the name of the item second, and the name of the variable third.

While the data type of the same parameter might vary among the different languages, the parameter contains the same information. For example, a parameter might be a string in ALGOL, a display in COBOL, and a character in FORTRAN77.

Basic syntax diagrams describing the internal format of the entry point parameters are provided in Sections 3 through 6 with individual entry point descriptions for each language. The syntax and semantics of the parameters for each entry point are very similar to those for the corresponding language extension. Behavioral differences between DMINTERPRETER entry points and language extensions are described in the text.

### Returning Exceptions

DMSII software evaluates each call to a DMINTERPRETER entry point. If the call cannot be successfully or correctly completed, DMSII returns an exception word to the program.

The exception handling entry points use the DMSUPPORT library to interpret and report the exceptions. Read Section 5, "Handling Exceptions," for details of the exception handling entry points.

## Programming Considerations for the Interpretive Interface Only

The following guidelines apply to the interpretive interface only. They explain how to sequence entry points and how to code your program. Consult the *DMSII Application Programming Guide* for more extensive programming guidelines for both the interpretive interface and the language extension interface.

How you sequence the DMINTERPRETER entry points within your application program can determine how efficiently you access, manage, and manipulate the database. Your choice of application programming language, your choice of entry points, and your style of coding can reduce run-time overhead.

### Sequencing Entry Points

The interpretive interface is designed to provide a record-at-a-time interface to a database. For programming efficiency, use the following guidelines:

- Locate records by using the DBFIND standard entry point. Examine the values of each item by using the appropriate DBGET data transfer entry point.
- Add a new record to the database by first calling the DBCREATE standard entry point. Then use the appropriate DBPUT data transfer entry point to move the values of each item into the user work area. Finally, store the record by calling the DBSTORE standard entry point.
- Modify a record by first calling the DBLOCK standard entry point to locate it. Then call the appropriate DBPUT data transfer entry points to alter the values of the items. Use the DBSTORE standard entry point to store the modified values.
- Remove a record from the database by calling the DBDELETE standard entry point.

### Reducing Run-Time Overhead

When accessing the interpretive interface, follow these recommendations to reduce run-time overhead:

- Whenever possible, write application programs in ALGOL.  
Because ALGOL programs directly call the entry points that perform the functions of the interpretive interface, they markedly reduce run-time overhead.
- In ALGOL, use preinitialized string variables instead of string expressions as parameters. Doing so eliminates the need to allocate string temporaries in the calling procedure.
- Use consecutive calls to minimize structure-switching overhead.  
The DMINTERPRETER library remembers the last structure on which it operated. Therefore, you can reduce structure-switching overhead if calls on one structure are not interspersed with calls on another structure.
- Minimize the use of blanks.  
Avoid passing unnecessary blanks to the interpretive interface. To eliminate the unnecessary blanks, allocate string temporaries and make calls to additional procedures.

The following example shows code with unnecessary blanks:

```
DBSTORE (" DS" )
```

The interpretive interface processes the following call faster:

```
DBSTORE("DS")
```

*Note: Do not eliminate all blanks. Include blanks where they make your code more readable. Always include required blanks.*

- Unconditionally call DBBEGINTRANSACTION and DBENDTRANSACTION entry points.

DMSII software ignores `DBBEGINTRANSACTION` and `DBENDTRANSACTION` calls for unaudited databases. You can simplify application programs by unconditionally calling these entry points. The programs can then be run unaltered against an audited or unaudited database.

- Use code-locking protocols within your application program.

The `DMINTERPRETER` library stack uses global items to retain information between calls. Using global items greatly speeds up execution, but prevents multiprocessing. Thus, if the library declared by a parent task is visible to two or more offspring tasks, the offspring can receive unexpected results if they call the library simultaneously.

Locking conventions within the `DMINTERPRETER` library can prevent unexpected results, but the locking conventions slow down execution speed. Consequently, when the library is visible to multiple offspring tasks, the application program should implement its own locking protocols.

- Compile separate code files for each database.

Do not use a database equation when accessing the `DMINTERPRETER` library. Instead, compile a separate `DMINTERPRETER` library code file for each database.





# Section 2

## Accessing the Interpretive Interface

The DMSII interpretive interface works with any programming language that can import library objects. This section, organized alphabetically by language, details the steps and syntax used in ALGOL, COBOL, and FORTRAN77 application programs to access a DMINTERPRETER library and its entry points.

### General Guidelines for Accessing the Interpretive Interface

As you write your application program, consider the following factors:

- Invoking or declaring a DMSII database actually invokes or declares the associated DMINTERPRETER library.
- Invoking or declaring a DMINTERPRETER library has two main purposes:
  - To make the names of the data items available to the application program
  - To perform syntax checking
- Access to a database must be explicitly requested by opening the database.
- Declaring an entry point makes the entry point available to the program.
  - In COBOL, invoking the database also declares the entry points.
  - In ALGOL and FORTRAN77, declaring the entry points is a separate programming step.
- Invoking an entry point, also known as calling the entry point, causes the actual task to be performed at run time.
- The symbolic file DATABASE/DMINTERPRETER, supplied on the release tape, contains declarations of library entry points for each language.

If you code your programs in COBOL, you cannot select a subset nor can you rename the entry points.

If you code your programs in ALGOL or FORTRAN77, you can choose to

- Declare all the entry points through the \$INCLUDE compiler control option.
- Select a subset of the entry points.
- Rename the entry points.

Read Sections 3 through 6 for details of how to code each of the entry points.

## Accessing the Interpretive Interface in ALGOL Programs

Application programs written in ALGOL use a 3-step process to access the interpretive interface. The program code must

1. Declare the DMINTERPRETER library.
2. Declare the entry points as Boolean procedures.
3. Invoke the entry points.

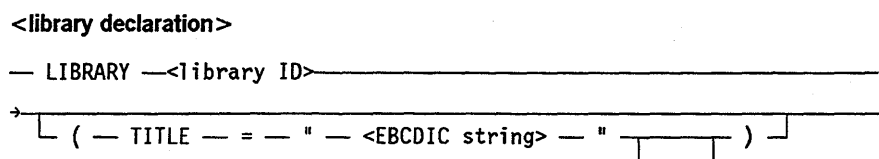
### Declaring a DMINTERPRETER Library in an ALGOL Program

ALGOL programs must link to a DMINTERPRETER library directly; the DMINTERPRETER library contains the procedures that are named in the EXPORT declaration.

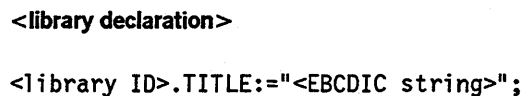
Use either the long or short form of the LIBRARY declaration to declare an identifier (ID) as the library ID. The TITLE attribute can be used to specify the object code file title of the library. Both the long and short form of the LIBRARY declaration are valid.

You can declare a library in any block of your user program. The library and its entry points are valid only within the scope of the block. If you exit the block, the program is no longer linked to the library.

The following syntax diagram shows the long form of the LIBRARY declaration:



Use the following syntax for the short form of the LIBRARY declaration:



As shown in the following syntax diagram, a library identifier is an ALGOL identifier.



The variable elements of the diagrams are explained as follows:

Element	Explanation
<library ID>	An internal name within the application program used to refer to the DMINTERPRETER library.
<identifier>	Any sequence of characters beginning with a letter and consisting of letters, digits, hyphens (-), and underscores (_).
<EBCDIC string>	A connected group or sequence of EBCDIC characters, excluding double quotation marks (").  In the long form, the EBCDIC string must have a period (.) as its last character and must be a properly formed file title.  In the short form, the EBCDIC string need not have a period as the last character.

Consult the *ALGOL Reference Manual, Vol. 1* for detailed information on libraries, the EXPORT declaration, and the LIBRARY declaration.

### Examples of ALGOL Linkage to a DMINTERPRETER Library

The following LIBRARY declaration specifies DMI as the library ID:

```
LIBRARY DMI;
```

In the next example, the TITLE attribute assigns the external name DMINTERPRETER/EMPJOB to the library ID. Because the example shows the long form of the declaration, the EBCDIC string has a period as its last character.

```
LIBRARY DMI (TITLE="DMINTERPRETER/EMPJOB.");
```

The next example shows the short form of LIBRARY declaration. Again, the library ID is DMI and the external name is DMINTERPRETER/EMPJOB. Note that the EBCDIC string does not have a period as its last character.

```
DMI.TITLE:= "DMINTERPRETER/EMPJOB";
```

### Declaring ALGOL Entry Points as Boolean Procedures

In ALGOL, you must declare each entry point of a DMINTERPRETER library as a Boolean procedure. You can declare the entry points in one of two ways:

- Use the declaration of the DMINTERPRETER library and the ALGOL entry points contained on the release tape.
- Individually declare the entry points in your application program.

### Using the Release Tape to Declare ALGOL Entry Points

The DATABASE/DMINTERPRETER symbolic file provided on the release tape contains

- A declaration of the DMINTERPRETER library
- Declarations of the ALGOL entry points
- Type declarations for the entry points

To incorporate all these declarations in your application program, use the \$INCLUDE compiler control option and specify the sequence range 20100000 through 20199999 within the DATABASE/DMINTERPRETER symbolic file. All the ALGOL entry points in the library are exported.

On the release tape, the library identifier is DMI. The exported names of the entry points are declared and renamed with the prefix DB. Table 2-1 lists the exported name of each entry point, the type of the entry point, and how it is renamed.

Table 2-1. ALGOL Entry Points

Exported Name	Type	Renamed As
ALGOLABORTTRANSACTION	Standard	DBABORTTRANSACTION
ALGOLBEGINTRANSACTION	Standard	DBBEGINTRANSACTION
ALGOLCANCELTRPOINT	Standard	DBCANCELTRPOINT
ALGOLCLOSE	Standard	DBCLOSE
ALGOLCREATE	Standard	DBCREATE
ALGOLDATA	Data transfer	DBDATA
ALGOLDELETE	Standard	DBDELETE
ALGOLENDTRANSACTION	Standard	DBENDTRANSACTION
ALGOLEXCEPTIONNAME	Exception handling	DBEXCEPTIONNAME
ALGOLEXCEPTIONTEXT	Exception handling	DBEXCEPTIONTEXT
ALGOLFIND	Standard	DBFIND
ALGOLFREE	Standard	DBFREE
ALGOLFREESTR	Standard	DBFREESTR
ALGOLGETBOOLEAN	Data transfer	DBGETBOOLEAN
ALGOLGETDOUBLE	Data transfer	DBGETDOUBLE
ALGOLGETKANJI	Data transfer	DBGETKANJI
ALGOLGETREAL	Data transfer	DBGETREAL
ALGOLGETSTRING	Data transfer	DBGETSTRING
ALGOLLOCK	Standard	DBLOCK

continued

Table 2-1. ALGOL Entry Points (cont.)

Exported Name	Type	Renamed As
ALGOLLOCKSTR	Standard	DBLOCKSTR
ALGOOPEN	Standard	DBOPEN
ALGOLPUTBOOLEAN	Data transfer	DBPUTBOOLEAN
ALGOLPUTDOUBLE	Data transfer	DBPUTDOUBLE
ALGOLPUTKANJI	Data transfer	DBPUTKANJI
ALGOLPUTNULL	Data transfer	DBPUTNULL
ALGOLPUTREAL	Data transfer	DBPUTREAL
ALGOLPUTSTRING	Data transfer	DBPUTSTRING
ALGOLRECREATE	Standard	DBRECREATE
ALGOLSAVETRPOINT	Standard	DBSAVETRPOINT
ALGOLSECURE	Standard	DBSECURE
ALGOLSECURESTR	Standard	DBSECURESTR
ALGOLSET	Standard	DBSET
ALGOLSETLIMIT	Attribute setting	DBSETLIMIT
ALGOLSTATUS	Exception handling	DBSTATUS
ALGOLSTORE	Standard	DBSTORE
ALGOLVERB	Standard	DBVERB

A table comparing the types of entry points (standard, data transfer, exception handling, and attribute setting) appears under "Identifying the Four Types of DMINTERPRETER Entry Points" in Section 1. The function and coding for each entry point are detailed in Sections 3 through 6.

### Using a Procedure Declaration for ALGOL Entry Points

Individually declare entry points in a procedure declaration if you want to

- Declare a subset of the entry points.
- Rename the entry points.

#### Declaring a Subset of ALGOL Entry Points

If you use the \$INCLUDE option, all ALGOL entry points in the library are exported. However, in some instances, your program only needs a few or selected entry points. In these cases, to improve processing time, declare each individual entry points in separate procedure declarations rather than export all the entry points.

## Accessing the Interpretive Interface

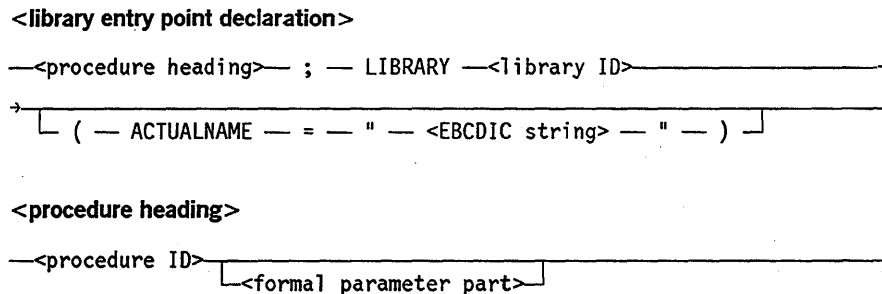
---

### Renaming ALGOL Entry Points

Unless specifically renamed, a library entry point is exported (accessed by the ALGOL program) by its name in the DATABASE/DMINTERPRETER symbolic file. The ACTUALNAME clause in a procedure declaration lets you rename a library entry point.

### Coding ALGOL Procedure Declarations

Use the procedure declaration to declare individual library entry points in an ALGOL program. The procedure identifier, given in the procedure heading, links to the appropriate entry point given in the ACTUALNAME clause. The following diagram shows the syntax of the procedure declaration:



Each variable element in the procedure declaration is explained as follows:

Element	Explanation
<procedure ID>	A unique sequence of characters beginning with a letter and consisting of letters, digits, hyphens (-), and underscores (_) that identify the procedure. The identified procedure cannot be declared FORWARD or EXTERNAL.
<formal parameter part>	A list of items to be passed as parameters when the procedure is invoked.
<library ID>	An internal name within the application program used to refer to the DMINTERPRETER library. The library ID must be previously declared in an available LIBRARY declaration.
<EBCDIC string>	The actual name that is given for this procedure in the template of this library. The name is given as a connected group of EBCDIC characters, excluding the quotation marks("").  In the ACTUALNAME clause, the EBCDIC string cannot contain any leading, trailing, or embedded blanks and must be a valid ALGOL identifier.

### Example of Declaring and Renaming an Entry Point in ALGOL

This example shows how an entry point can be declared within an ALGOL program. The LIBRARY declaration gives DMI as the library ID. The LIBRARY declaration is followed by the two-part LIBRARY entry point declaration; the procedure declaration and the library notation.

The Boolean procedure declaration names the entry point `DBOPEN` and defines its parameter, `OPENTYPE`, as a string. The next `LIBRARY` declaration tells the application program to use the procedure `ALGOLOPEN` in the DMI library every time `DBOPEN` is invoked. For this program only, `ALGOLOPEN` is renamed as `DBOPEN`. The actual name of the entry point in the library remains `ALGOLOPEN`.

```
LIBRARY DMI;  
BOOLEAN PROCEDURE DBOPEN (OPENTYPE);  
  STRING OPENTYPE;  
LIBRARY DMI (ACTUALNAME="ALGOLOPEN");
```

### Invoking ALGOL Entry Points

Once you have declared a `DMINTERPRETER` library and its entry points in your ALGOL application program, you can invoke any declared entry point as a Boolean procedure.

After an entry point call on the `DMINTERPRETER` library, a result descriptor is returned as a Boolean value. This result descriptor is a 48-bit word, which is the standard DMSII exception word. It includes the exception category and subcategory, and the structure number.

When a call on the `DMINTERPRETER` library causes an exception, the last bit in the 48-bit word is set to 1. When a call on the library is successful, the 48-bit word contains all zeros.

Content of Exception Word	Meaning
0	The call was successful.
1	An exception occurred.

For more information on exceptions, refer to Section 5, "Handling Exceptions."

### Typical Method for Invoking an ALGOL Entry Point

The following is a typical method for invoking an ALGOL entry point, where `RSLT` is a Boolean variable, `DBOPEN` is the renamed entry point `ALGOLOPEN`, and `EXCEPTIONHANDLER` is a procedure for handling exceptions:

```
RSLT:=DBOPEN(OPEN_TYPE);  
IF RSLT  
  THEN EXCEPTIONHANDLER;
```



## Accessing the Interpretive Interface in COBOL Programs

Application programs written in any of the supported COBOL languages use a 2-step process to access the interpretive interface. The program code must

1. Change the TITLE attribute of the DMINTERPRETER library.
2. Invoke the entry points through a CALL statement.

Consult volume 1 of the appropriate COBOL reference manual for details of COBOL, COBOL74, and COBOL85 syntax.

### Exported Names of COBOL Entry Points on the Release Tape

The DATABASE/DMINTERPRETER symbolic file provided on the release tape contains the exported names of the COBOL entry points. You cannot explicitly declare or rename any library entry point. Table 2-2 lists the type and exported name for each entry point.

*Note: The parameters for COBOL entry points are translated into parameters that are compatible with ALGOL. The ALGOL entry points within the DMINTERPRETER library perform the actual operations against the database.*

Table 2-2. COBOL, COBOL74, and COBOL85 Entry Points

Type	Exported Name
Standard	DBABORTTRANSACTION
Standard	DBBEGINTRANSACTION
Standard	DBCANCELTRPOINT
Standard	DBCLOSE
Standard	DBCREATE
Data transfer	DBDATA
Standard	DBDELETE
Standard	DBENDTRANSACTION
Exception handling	DBEXCEPTIONNAME
Exception handling	DBEXCEPTIONTEXT
Standard	DBFIND
Standard	DBFREE
Standard	DBFREESTR

continued

Table 2-2. COBOL, COBOL74, and COBOL85 Entry Points (cont.)

Type	Exported Name
Data transfer	DBGETBOOLEAN
Data transfer	DBGETDISPLAY
Data transfer	DBGETDOUBLE
Data transfer	DBGETKANJI
Data transfer	DBGETREAL
Standard	DBLOCK
Standard	DBLOCKSTR
Standard	DBOPEN
Data transfer	DBPUTBOOLEAN
Data transfer	DBPUTDISPLAY
Data transfer	DBPUTDOUBLE
Data transfer	DBPUTKANJI
Data transfer	DBPUTNULL
Data transfer	DBPUTREAL
Standard	DBRECREATE
Standard	DBSAVETRPOINT
Standard	DBSECURE
Standard	DBSECURESTR
Standard	DBSET
Attribute setting	DBSETLIMIT
Exception handling	DBSTATUS
Standard	DBSTORE
Standard	DBVERB

A table comparing the types of entry points (standard, data transfer, exception handling, and attribute setting) appears under “Identifying the Four Types of DMINTERPRETER Entry Points” in Section 1. The function and coding for each entry point are detailed in Sections 3 through 6.

## Using the COBOL CHANGE Construct to Invoke a DMINTERPRETER Library

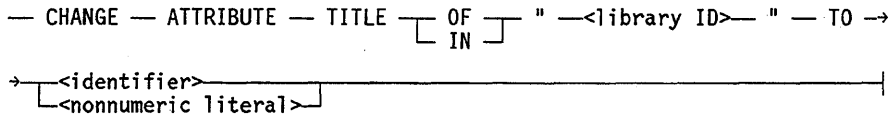
To invoke the DMINTERPRETER library in COBOL application programs, use the CHANGE construct to change the TITLE attribute of the library to the actual name of the library code file.

## Accessing the Interpretive Interface

---

The CHANGE construct changes the library ID to an identifier or a nonnumeric literal within the application program. Every time the program makes a call on the library ID, the call is directed to the library whose name corresponds to the identifier or the nonnumeric literal. The following syntax diagram shows how to code the CHANGE construct:

### CHANGE Construct



Each variable element of the CHANGE construct is explained as follows:

Element	Explanation
<library ID>	<p>The file name of the library to be called. The file name can be composed of letters, numbers, hyphens (-), underscores (_), and slashes (/). The library ID, including slashes, is used as the title of the library.</p> <p>If the library ID does not contain slashes, the entire name can be used as the internal name (INTNAME) for the library.</p> <p>If the library ID contains slashes, then the identifier that follows the last slash is used as the internal name (INTNAME) for the library.</p>
<identifier>	<p>A data-name followed by the syntactically correct combination of qualifiers, subscripts, and indexes needed to uniquely reference the data item.</p>
<nonnumeric literal>	<p>A character string bounded by double quotation marks ("). The string can include any character in the character set of the computer. (To represent a single quotation mark character, use two single contiguous quotation marks. To represent a double quotation mark, use three single contiguous quotation marks.)</p>

### Example of COBOL Linkage to a DMINTERPRETER Library

This example changes the library identifier DMINTERPRETER to the identifier DMI-NAME. Within the program, all references to the library are to DMI-NAME.

\*\*\*Declaring the identifier

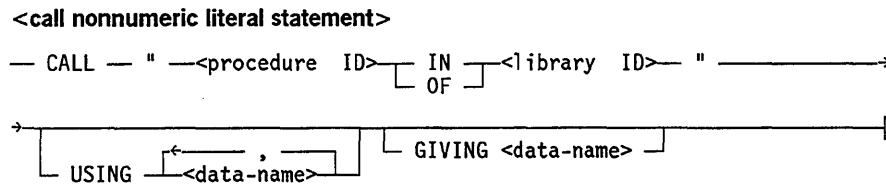
```
01 DMI-NAME      PIC X(24) VALUE IS "DMINTERPRETER/EMPJOB".
```

\*\*\*Linking to DMINTERPRETER library

```
CHANGE ATTRIBUTE TITLE OF "DMINTERPRETER" TO DMI-NAME.
```

## Invoking COBOL Entry Points

Use a CALL nonnumeric literal statement to invoke entry points to the DMINTERPRETER library. The CALL statement passes the required parameters in a USING clause and receives a result in a GIVING clause, as shown in the following syntax:



The variable elements of the CALL statement are explained as follows:

Element	Explanation
<procedure ID>	Name of the procedural entry point.
<library ID>	The file name of the library to be called. The file name can be composed of letters, numbers, hyphens (-), underscores (_), and slashes (/).
<data-name>	In the USING clause, the name of the entry point to the DMINTERPRETER library.  In the GIVING clause, the name of the variable where the result of the call is stored.

Refer to Sections 3 through 6 for examples and detailed explanations of how to use the required parameters for each entry point.

## Passing Parameters with the USING Clause in COBOL

Although parameters for the standard entry points and for most of the data transfer entry points are declared as the same type and at the same level for all COBOL languages, there are some differences:

- Standard entry points and most data transfer entry points in all COBOL languages are DISPLAY items declared at level 01.
- In COBOL, the following data transfer entry points must be declared as COMP-4 or COMP-5 items at level 77. In COBOL74 and COBOL85, they must be declared as REAL or DOUBLE items at level 77.

DBGETREAL	DBPUTREAL
DBGETDOUBLE	DBPUTDOUBLE

## Accessing the Interpretive Interface

---

### Exception Handling with the GIVING Clause in COBOL

After a call on an entry point of the DMINTERPRETER library, a result descriptor is returned as an integer into the result variable identified in the GIVING clause. The returned integer is either 0 or 1, as shown in the following table.

Returned Integer	Meaning
0	The call was successful.
1	An exception occurred.

For COBOL, the result variable must be a COMP-2 item declared at level 01. For COBOL74 and COBOL85, the result variable must be a COMP item declared at level 01.

If an exception occurs, the program must then call the exception handling entry points. For more information on exceptions, refer to Section 5, "Handling Exceptions."

#### Typical Method for Invoking a COBOL Entry Point

The following example invokes the COBOL entry point DBCREATE in the DMINTERPRETER library. Two parameters are passed: one in the variable DATA-SET-NAME and one in the variable SPACE-1. The result descriptor is returned in the variable RESULT.

```
Ø1 DATA-SET-NAME          PIC X(17) VALUE IS "EMP".
Ø1 SPACE-1                 PIC X(1) VALUE IS " ".

CALL "DBCREATE OF DMINTERPRETER"
    USING DATA-SET-NAME, SPACE-1
    GIVING RESULT.
```

## Accessing the Interpretive Interface in FORTRAN77 Programs

Application programs written in FORTRAN77 use a 3-step process to access the interpretive interface:

1. Declare the DMINTERPRETER library.
2. Declare the entry points as logical functions.
3. Invoke the entry points.

Your FORTRAN77 program must explicitly declare a DMINTERPRETER library and each of the DMINTERPRETER library entry points that will be used. The declarations must appear once only, in the BLOCK GLOBALS subprogram. You must include the type declarations in every subprogram within the program.

## Declaring a DMINTERPRETER Library in FORTRAN77 Programs

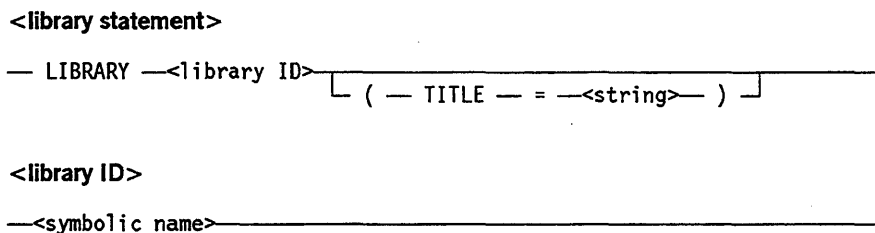
FORTRAN77 programs must link to a DMINTERPRETER library directly; the DMINTERPRETER library contains the named functions.

Use the LIBRARY statement to declare a symbolic name as the library ID. You can use the TITLE attribute to assign the external name of the library code file to the library ID. Then, within the application program, you can use the library ID as an internal name of the DMINTERPRETER library.

In FORTRAN77 programs, libraries are declared in a BLOCK GLOBALS subprogram. Each program has only one BLOCK GLOBALS subprogram. The subprogram must precede all other source statements. The subprogram is initiated by a BLOCK GLOBALS statement and terminated by an END statement.

Consult the *FORTRAN77 Reference Manual* for more information on direct linkage, BLOCK GLOBALS subprograms, and the LIBRARY statement.

The following diagram gives the syntax of the FORTRAN77 library statement:



Each variable element in the library statement is explained as follows:

Element	Explanation
<library ID>	An internal name within the application program used to refer to the DMINTERPRETER library.
<symbolic name>	A sequence of from 1 to 31 letters, digits, underscores ( _ ), or the dollar sign (\$) that identifies the internal name of a DMINTERPRETER library. The first character must be either a letter or the dollar sign (\$).
<string>	Name of the library code file. The string can consist of no more than 255 characters, terminated by a period (.). The string must be delimited with an apostrophe (') or double quotation mark ("). An empty string is not allowed.

**Note:** Any input record that begins with a dollar sign (\$) in column 1 or column 2 is treated as a compiler control image. If your program code has a symbolic name that begins with a dollar sign and starts in column 1 or column 2, your program will not compile successfully.

## Accessing the Interpretive Interface

---

### Example of FORTRAN77 Linkage to a DMINTERPRETER Library

The following code shows how the LIBRARY statement is used in a BLOCK GLOBALS subprogram. The library ID is DMI. The TITLE attribute is used to assign the external name DMINTERPRETER/EMPJOB to the library ID.

```
BLOCK GLOBALS
  LIBRARY DMI(TITLE="DMINTERPRETER/EMPJOB.")
END
```

### Declaring FORTRAN77 Entry Points as Logical Functions

In FORTRAN77, you must declare each entry point of a DMINTERPRETER library as a logical function. You can do so in one of two ways:

- Use the declaration of the DMINTERPRETER library and the FORTRAN77 entry points contained on the release tape.
- Individually declare the entry points in your application program.

### Using the Release Tape to Declare FORTRAN77 Entry Points

The DATABASE/DMINTERPRETER symbolic file provided on the release tape contains

- A declaration of the DMINTERPRETER library
- Declarations of the FORTRAN77 entry points
- Type declarations for the entry points

To incorporate these declarations in your application programs, use the \$INCLUDE compiler control option and specify the following sequence ranges within the DATABASE/DMINTERPRETER symbolic file:

Declaration	Sequence Range
Library entry point declaration	20200000 through 20249999
Type declaration	20290000 through 20299999

All the FORTRAN77 entry points in the library are exported.

On the release tape, the library identifier is DMI. The exported names of the entry points are declared and renamed with the prefix DB. Table 2-3 lists the exported name of each entry point, the type of the entry point, and how it is renamed.

Table 2-3. FORTRAN77 Entry Points

Exported Name	Type	Renamed As
FORTAN77ABORTTRANSACTION	Standard	DBATR
FORTAN77BEGINTRANSACTION	Standard	DBBTR
FORTAN77CANCELTRPOINT	Standard	DBCNCL
FORTAN77CLOSE	Standard	DBCLOSE
FORTAN77CREATE	Standard	DBCR
FORTAN77DATA	Data transfer	DBDATA
FORTAN77DELETE	Standard	DBDEL
FORTAN77ENDTRANSACTION	Standard	DBETR
FORTAN77EXCEPTIONNAME	Exception handling	DBEXCN
FORTAN77EXCEPTIONTEXT	Exception handling	DBEXCT
FORTAN77FIND	Standard	DBFIND
FORTAN77FREE	Standard	DBFREE
FORTAN77FREESTR	Standard	DBFSTR
FORTAN77GETCHARACTER	Data transfer	DBGETC
FORTAN77GETDOUBLE	Data transfer	DBGETD
FORTAN77GETKANJI	Data transfer	DBGETK
FORTAN77GETLOGICAL	Data transfer	DBGETL
FORTAN77GETREAL	Data transfer	DBGETR
FORTAN77LOCK	Standard	DBLOCK
FORTAN77LOCKSTR	Standard	DBLSTR
FORTAN77OPEN	Standard	DBOPEN
FORTAN77PUTCHARACTER	Data transfer	DBPUTC
FORTAN77PUTDOUBLE	Data transfer	DBPUTD
FORTAN77PUTKANJI	Data transfer	DBPUTK
FORTAN77PUTLOGICAL	Data transfer	DBPUTL
FORTAN77PUTNULL	Data transfer	DBPUTN
FORTAN77PUTREAL	Data transfer	DBPUTR
FORTAN77RECREATE	Standard	DBRCR
FORTAN77SAVETRPOINT	Standard	DBSAVE
FORTAN77SECURE	Standard	DBSEC
FORTAN77SECURESTR	Standard	DBSSTR

continued



Table 2-3. FORTRAN77 Entry Points (cont.)

Exported Name	Type	Renamed As
FORTRAN77SET	Standard	DBSET
FORTRAN77SETLIMIT	Attribute setting	DBSETL
FORTRAN77STATUS	Exception handling	DBSTAT
FORTRAN77STORE	Standard	DBSTORE
FORTRAN77VERB	Standard	DBVERB

A table comparing the types of entry points (standard, data transfer, exception handling, and attribute setting) appears under “Identifying the Four Types of DMINTERPRETER Entry Points” in Section 1. The function and coding for each entry point are detailed in Sections 3 through 6.

### Using the Logical Function for FORTRAN77 Entry Points

Individually declare entry points if you want to do the following:

- Declare a subset of the FORTRAN77 entry points.
- Rename the FORTRAN77 entry points.

To individually declare entry points, you need to use the LOGICAL FUNCTION library entry point declaration, type statements, and the IN LIBRARY statement. The entry point declaration and IN LIBRARY statement are explained in the following text. For specific information about type statements, consult the *FORTRAN77 Reference Manual*.

#### Declaring a Subset of the FORTRAN77 Entry Points

If you use the \$INCLUDE option, all FORTRAN77 entry points in the library are exported. However, in some instances, your program only needs a few or selected entry points. In these cases, to improve processing time, declare the individual entry points rather than export all the entry points.

#### Coding the FORTRAN77 Library Entry Point Declaration

The LOGICAL FUNCTION library entry point declaration identifies the entry point and parameters. The following diagram shows the syntax of the declaration:

<library entry point declaration>

— LOGICAL FUNCTION —<logical function name>— ( —<parameter>— ) —

Each variable element in the LOGICAL FUNCTION is explained as follows:

Element	Explanation
<logical function name>	A unique name that identifies the entry point in the library
<parameter>	A list of items to be passed as parameters when the entry point is invoked

You must give the data type for each parameter in a type statement. For example, the following type statement identifies the data type of the parameter OTYPE as a character:

```
LOGICAL FUNCTION DBOPEN (OTYPE)
CHARACTER OTYPE
```

The type statements must immediately follow the LOGICAL FUNCTION and must precede the IN LIBRARY statement.

### Renaming FORTRAN77 Entry Points

Unless specifically renamed, a library entry point is exported (accessed by the FORTRAN77 program) by its name in the DATABASE/DMINTERPRETER symbolic file. The ACTUALNAME clause in an IN LIBRARY statement allows you to rename a library entry point.

### Coding a FORTRAN77 IN LIBRARY Statement

The IN LIBRARY statement associates the logical function name (given in the library entry point declaration) with the library identifier. The ACTUALNAME clause allows you to rename the library entry point. The following diagram shows the syntax of the statement:

```
<in library statement>
— IN LIBRARY —<library ID> ( — ACTUALNAME — = —<string>— )
```

Each variable element in the statement is explained as follows:

Element	Explanation
<library ID>	A unique sequence of from 1 to 31 characters consisting of letters, digits, underscores (_), or the dollar sign (\$) that identifies the symbolic name of the library. The first character must be either a letter or the dollar sign (\$).
<string>	The actual name given for this logical function in the template of the library.

**Note:** Any input record that begins with a dollar sign (\$) in column 1 or column 2 is treated as a compiler control image. If your program code has a symbolic name that begins with a dollar sign and starts in column 1 or column 2, your program will not compile successfully.

## Accessing the Interpretive Interface

---

### Example of Declaring an Entry Point in a FORTRAN77 Program

The following example shows how an entry point might be declared within a FORTRAN77 program. The logical function name is DBOPEN. Its one parameter, OTYPE, has a data type of character. The IN LIBRARY statement tells the application program to use the procedure FORTRAN77OPEN in the DMI library every time DBOPEN is invoked. For this program only, FORTRAN77OPEN is renamed DBOPEN.

This program must have a preceding LIBRARY statement in a BLOCK GLOBALS subprogram to declare the DMINTERPRETER library before declaring the library entry point.

```
LOGICAL FUNCTION DBOPEN (OTYPE)
  CHARACTER OTYPE
  IN LIBRARY DMI (ACTUALNAME = "FORTRAN77OPEN")
END
```

### Invoking FORTRAN77 Entry Points

Once you have declared a DMINTERPRETER library and its entry points in your FORTRAN77 application program, you can invoke any declared entry point using a logical assignment statement.

After an entry point call on the DMINTERPRETER library, DMSII software evaluates the logical function and returns a logical value to a program-assigned variable. The variable contains the result of the call, a result descriptor.

The result descriptor is a 48-bit word, which is the standard DMSII exception word. It includes the exception category and subcategory, as well as the structure number. When a call on the DMINTERPRETER library causes an exception, the last bit in the 48-bit word is set to 1. When a call on the library is successful, the 48-bit word contains all zeros.

Content of Exception Word	Meaning
0	The call was successful.
1	An exception occurred.

For more information on exceptions, refer to Section 5, "Handling Exceptions."

### Typical Method for Invoking a FORTRAN77 Entry Point

The following is a typical method for invoking a FORTRAN77 entry point, where RSLT is a logical variable, DBOPEN is the renamed entry point FORTRAN77OPEN, and EXCEPT is a subroutine for handling exceptions:

```
RSLT = DBOPEN(UPDATE)
IF (RSLT) CALL EXCEPT
```

# Section 3

## Manipulating the Database

Standard entry points perform the standard database manipulation tasks, such as locating a record, beginning and ending transaction state, and storing records.

Each standard database manipulation task with its corresponding entry points are listed in Table 3-1. Use the heading for each listed task to locate the pages that detail the entry point. For example, a description of the standard entry points ALGOOPEN, DBOPEN, and FORTRAN77OPEN is found under "Opening a Database." The tasks are presented in the general order in which they are usually coded, starting with opening the database and ending with closing the database. General guidelines for using the standard entry points precede the task descriptions.

**Table 3-1. Standard Entry Points to the DMINTERPRETER Library by Task**

<b>Task</b>	<b>Entry Points</b>
<b>Opening a database</b>	ALGOOPEN DBOPEN FORTRAN77OPEN
<b>Creating reserve space for new records</b>	ALGOCREATE DBCREATE FORTRAN77CREATE
<b>Setting the current path</b>	ALGOLSET DBSET FORTRAN77SET
<b>Explicitly freeing records</b>	ALGOLFREE DBFREE FORTRAN77FREE
<b>Explicitly freeing structures</b>	ALGOLFREESTR DBFREESTR FORTRAN77FREESTR
<b>Finding records</b>	ALGOLFIND DBFIND FORTRAN77FIND

continued

## Manipulating the Database

---

Table 3-1. Standard Entry Points to the DMINTERPRETER Library by Task (cont.)

Task	Entry Points
Locking records	ALGOLLOCK DBLOCK FORTRAN77LOCK
Locking structures	ALGOLLOCKSTR DBLOCKSTR FORTRAN77LOCKSTR
Securing records	ALGOLSECURE DBSECURE FORTRAN77SECURE
Securing structures	ALGOLSECURESTR DBSECURESTR FORTRAN77SECURESTR
Begin transaction—entering transaction state	ALGOLBEGINTRANSACTION DBBEGINTRANSACTION FORTRAN77BEGINTRANSACTION
Aborting transactions	ALGOLABORTTRANSACTION DBABORTTRANSACTION FORTRAN77ABORTTRANSACTION
Saving transaction points	ALGOLSAVETRPOINT DBSAVETRPOINT FORTRAN77SAVETRPOINT
Canceling transactions back to savepoints	ALGOLCANCELTRPOINT DBSCANCELTRPOINT FORTRAN77CANCELTRPOINT
Deleting data records	ALGOLDELETE DBDELETE FORTRAN77DELETE
Re-creating records	ALGOLRECREATE DBRECREATE FORTRAN77RECREATE

continued

Table 3-1. Standard Entry Points to the DMINTERPRETER Library by Task (cont.)

Task	Entry Points
Storing records	ALGOLSTORE
	DBSTORE
	FORTRAN77STORE
End transaction—leaving transaction state	ALGOENDTRANSACTION
	DBENDTRANSACTION
	FORTRAN77ENDTRANSACTION
Closing a database	ALGOLCLOSE
	DBCLOSE
	FORTRAN77CLOSE
Executing language extensions	ALGOLVERB
	DBVERB
	FORTRAN77VERB

The discussion of each task includes the following:

- A brief explanation of the function of the entry point
- Syntax diagrams and semantics that describe required parameters
- Program fragments that illustrate
  - Declaring variables for the entry points
  - Invoking the entry point
  - Returning results from the DMINTERPRETER library

Unless specifically noted, COBOL, COBOL74, and COBOL85 have the same requirements.

## Guidelines for Using Standard Entry Points

For programming efficiency, use the following guidelines:

- Always open the database explicitly as the first database operation.
- Create and find records outside transaction state.
- Make all data transfers outside transaction state. (Read Section 4, “Transferring Data,” for details of transferring data using DBGET and DBPUT data transfer entry points.)

## Manipulating the Database

---

- Re-create a record immediately after deleting the base record and before any operation can alter the user work area. You can re-create the record either in or out of transaction state.
- Enter transaction state to store and delete records.
- Always explicitly close the database.

Figure 3-1 illustrates the ideal flow for standard entry point operations in and out of transaction state. The figure emphasizes that you must declare (invoke) the database before using any entry point operation. The figure does not show how the flow is effected by the use of data transfer entry points, exception handling entry points, or the attribute setting entry point.

Read Sections 4 through 6 for the syntax of data transfer entry points, exception handling entry points, and the attribute setting entry point, respectively. Section 2, "Accessing the Interpretive Interface," details how each language accesses the interpretive interface and how each language invokes an entry point.

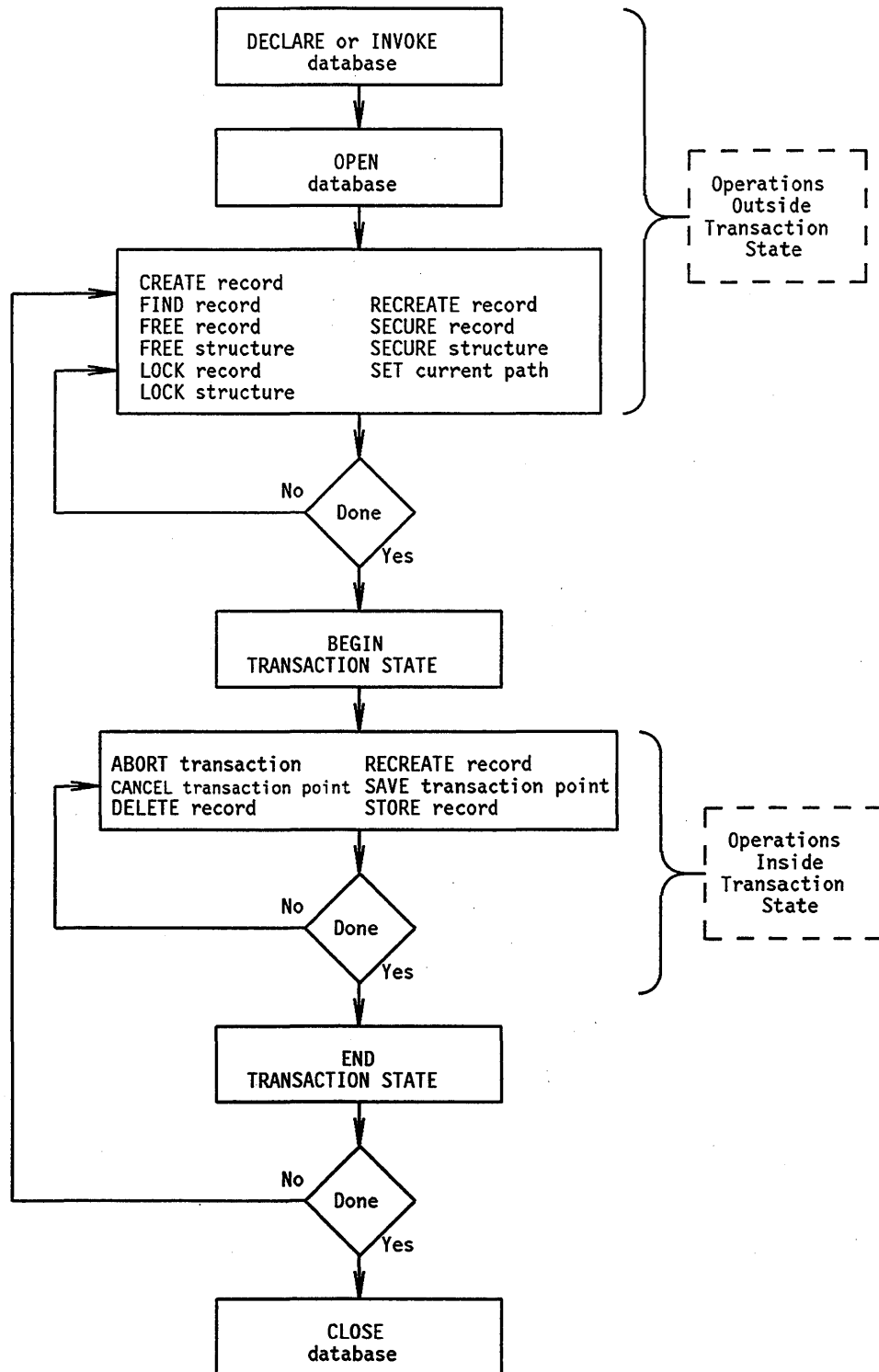


Figure 3-1. Standard Entry Point Operations In and Out of Transaction State



# Opening a Database

An open operation establishes the way in which the program accesses data values and performs database operations.

- Inquiry access allows the program to retrieve data but not add, delete, or alter data.
- Update access allows the program to retrieve and manipulate data.

You must explicitly open the database before performing any database operation. Until the database is explicitly opened, any call on a standard entry point, data transfer entry point, or attribute setting entry point results in an error. However, whether the database is opened or closed, your program can call the exception handling entry points and perform operations on the intrinsic data set DBSTRUCTURE.

To reduce system overhead, open each database only once in any given program execution.

### Passing a Parameter

The standard entry point used to open the database has one required parameter — open disposition:

Parameter	Explanation
<open disposition>	Determines the access mode—whether the database is opened for inquiry (read-only) or for both inquiry and updates

Note that you do not pass the name of the database to the entry point. The entry point assumes you want to open the most recently declared database.

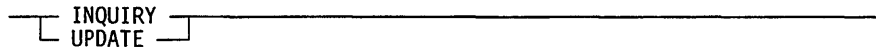
Table 3-2 gives the exported names of the entry points, along with the parameter and the data type of the parameter.

Table 3-2. Opening a Database

Exported Name of Entry Point	Parameter	Data Type of Parameter
ALGOOPEN	<open disposition>	STRING
DBOPEN	<open disposition>	DISPLAY
FORTTRAN77OPEN	<open disposition>	CHARACTER

The following syntax diagram shows that the open disposition parameter can set the access mode to one of two values:

<open disposition>



The values of the open disposition parameter are explained as follows:

Element	Explanation
INQUIRY	Allows the program to retrieve data only; does not allow updates. Any request to add, delete, or alter records results in an exception condition.
UPDATE	Allows the program to retrieve data and to add, delete, or alter data.

### Programming Examples

The following program fragments set the open disposition to INQUIRY. The database is open for both inquiries and updates.

#### ALGOL Program Fragment

```
%Declaring local variables

STRING
  OPEN_TYPE;
BOOLEAN
  RSLT;

%Assigning values to parameters

OPEN_TYPE:="UPDATE";

%Declaring library entry point

LIBRARY DMI;
BOOLEAN PROCEDURE DBOPEN (OPENTYPE);
  STRING OPENTYPE;
  LIBRARY DMI (ACTUALNAME="ALGOLOPEN");

%Invoking entry point

RSLT:=DBOPEN(OPEN_TYPE);
IF RSLT
  THEN EXCEPTIONHANDLER;
```

### COBOL74 or COBOL85 Program Fragment

*Note: In the following example, the variable RESULT is declared as a COMP item, making this example a COBOL74 or COBOL85 program fragment. If RESULT were declared as a COMP-2 item, this fragment could be used in a COBOL program.*

\*\*\*Declaring variables

```
Ø1 OPEN-TYPE    PIC X(6) VALUE "UPDATE".
Ø1 RESULT       PIC 9(1) COMP VALUE IS Ø.
```

\*\*\*Invoking entry point

```
CALL "DBOPEN OF DMINTERPRETER"
    USING OPEN-TYPE
    GIVING RESULT.
```

### FORTRAN77 Program Fragment

\*\*\*Library declaration for entry point

```
LOGICAL FUNCTION DBOPEN (Otype)
  CHARACTER Otype
  IN LIBRARY DMI (ACTUALNAME = "FORTRAN77OPEN")
END
```

\*\*\*Setting variables to initial values

```
CHARACTER UPDATE    *6    /"UPDATE"/
```

\*\*\*Declaring local variables

```
LOGICAL  RSLT
```

\*\*\*Invoking entry point

```
RSLT = DBOPEN(UPDATE)
IF (RSLT) CALL EXCEPT
```

## Creating Reserve Space for New Records

Adding a new record is a three-step operation:

1. Reserve space in the user work area for the new record (using either ALGOLCREATE, DBCREATE, or FORTRAN77CREATE). DMSII initializes the work area with null values for each item in the record.
2. Move values from program-declared variables into the record using DBPUT data transfer entry points. (See Section 4, "Transferring Data.")
3. Store the record into the database. (See "Storing Records" later in this section.)

You can create new records both in and out of transaction state. You can transfer values both in and out of transaction state. However, your program must be in transaction state to store the record. For faster processing time, create new records and transfer values outside of transaction state. Then enter transaction state to store the record.

### Passing Parameters

The standard entry point used to reserve space in the user work area has two required parameters: the data set qualified name and the record type.

Parameter	Explanation
<data set qualified name>	Identifies the data set for the new record.
<record type>	Determines whether the record has a variable or a fixed format. (The <i>DMSII Technical Overview</i> has a detailed explanation of variable and fixed-format records.)

Table 3-3 gives the exported names of the entry points, along with the parameters and the data type of each parameter.

Table 3-3. Creating Reserve Space for New Records

Exported Name of Entry Point	Parameter	Data Type of Parameter
ALGOLCREATE	<data set qualified name>	STRING
	<record type>	STRING
DBCREATE	<data set qualified name>	DISPLAY
	<record type>	DISPLAY
FORTRAN77CREATE	<data set qualified name>	CHARACTER
	<record type>	CHARACTER

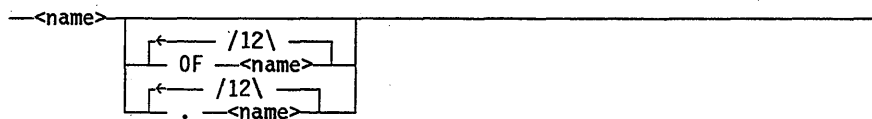
## Creating Reserve Space for New Records (cont.)

---

### Data Set Qualified Name

The data set name must be unique. The following syntax diagram shows you can use a maximum of 12 qualifying parameters to uniquely identify the data set:

<data set qualified name>



The variable element <name> of the data set name is explained as follows:

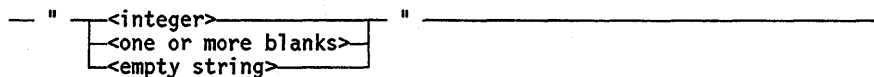
Element	Explanation
<name>	Identifies the data set for the new record.  The name can consist of from 1 to 17 letters, digits, and—depending on the language—either underscores ( _ ) or hyphens (-). The first character must be a letter. The last character must be either a letter or a digit.

For example, if your program accesses two databases (DB1 and DB2) and each database has a data set named CLASS, you must qualify the data set name by naming the database.

### Record Type

Use the following syntax to specify the record type for the new record:

<record type>



The variable elements of the record type are explained as follows:

Element	Explanation
<integer>	Creates a variable-format record. An integer is a whole number (0123456789) that consists of 1 to 12 digits and whose maximum value can be 549755813887.  A variable format consists of two parts: a fixed part (header) and a variable part (trailer). <ul style="list-style-type: none"><li>• Enter a nonzero integer to create both parts of a variable-format record.</li><li>• Enter a zero (0) to create only the fixed part of a variable-format record..</li></ul>
one or more blanks	Creates a fixed-format record.
empty string	Creates a fixed-format record.

### Programming Examples

The following program fragments create a new fixed-format record for the data set EMP. The data set name is unique and does not need any qualifying parameters.

#### ALGOL Program Fragment

```
%Declaring local variables

STRING
  DATA_SET_NAME, SPACE_1;
BOOLEAN
  RSLT;

%Assigning values to parameters

DATA_SET_NAME:="EMP";
SPACE_1:=" ";

%Declaring library entry point

LIBRARY DMI;
BOOLEAN PROCEDURE DBCREATE(STRUCTURE,RECORDTYPE);
STRING STRUCTURE,RECORDTYPE;
LIBRARY DMI (ACTUALNAME="ALGOLCREATE");

%Invoking entry point

RSLT:=DBCREATE(DATA_SET_NAME,SPACE_1);
IF RSLT
  THEN EXCEPTIONHANDLER;
```

#### COBOL74 or COBOL85 Program Fragment

**Note:** *In the following example, the variable RESULT is declared as a COMP item, making this example a COBOL74 or COBOL85 program fragment. If RESULT were declared as a COMP-2 item, this fragment could be used in a COBOL program.*

```
***Declaring variables

Ø1 DATA-SET-NAME          PIC X(17) VALUE IS "EMP".
Ø1 SPACE-1                PIC X(1) VALUE IS " ".
Ø1 RESULT                  PIC 9(1) COMP VALUE IS Ø.

***Invoking entry point

CALL "DBCREATE OF DMINTERPRETER"
  USING DATA-SET-NAME, SPACE-1
  GIVING RESULT.
```

## Creating Reserve Space for New Records (cont.)

---

### FORTRAN77 Program Fragment

\*\*\*Declaring library entry point

```
LOGICAL FUNCTION DBCR (STR, RECTYP)
  CHARACTER STR, RECTYP
  IN LIBRARY DMI (ACTUALNAME = "FORTRAN77CREATE")
  END
```

\*\*\*Setting variables to initial values

```
CHARACTER DSNAME      *4      /"EMP"/,
*      SPACE          *1      /" "/
```

\*\*\*Declaring local variables

```
LOGICAL  RSLT
```

\*\*\*Invoking entry point

```
RSLT = DBCR(DSNAME,SPACE)
IF (RSLT) CALL EXCEPT
```

## Setting the Current Path

As your program invokes a structure, the DMSII software creates and maintains a current path for that structure. The current path points to the record in that structure your program is currently using. This record is known as the current record.

You can position, or set, the current path of a data set to either the beginning or ending of the structure. The beginning of the structure is one record position before the first physical record. The end of the structure is one record position after the last physical record.

You also can set the current path of an index set with the ALGOLSET, DBSET, and FORTRAN77SET entry points. The current point of an index set is positioned to the current record within the spanned data set.

### Passing Parameters

The standard entry point used to set a current path has two required parameters: the name of a structure and a target location.

Parameter	Explanation
<structure>	Identifies either the data set or index set whose current path will be reset.
<target>	Identifies the location where the current path should point: the beginning of the structure, the end of the structure, or, for an index set, the current record in a particular data set.

Table 3-4 gives the exported names of the entry points, along with the parameters and the data type of each parameter.

Table 3-4. Setting the Current Path

Exported Name of Entry Point	Parameter	Data Type of Parameter
ALGOLSET	<structure>	STRING
	<target>	STRING
DBSET	<structure>	DISPLAY
	<target>	DISPLAY
FORTRAN77SET	<structure>	CHARACTER
	<target>	CHARACTER

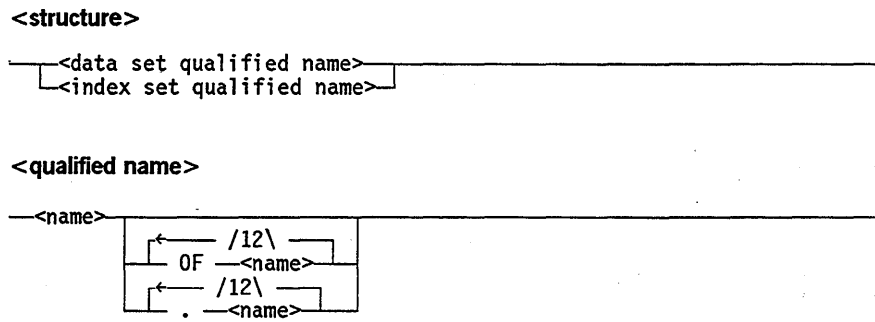


## Setting the Current Path (cont.)

---

### Structure

The structure can be either a data set or an index set. The name of the set must be unique. The following syntax diagram shows you can use a maximum of 12 qualifying parameters to uniquely identify the data set or index set:



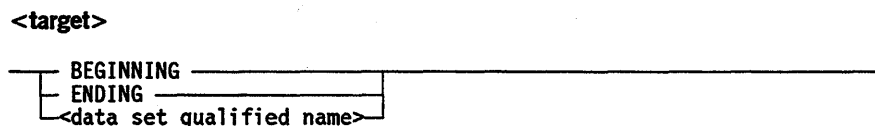
The variable elements of the structure parameter are explained as follows:

Element	Explanation
<data set qualified name>	Identifies the data set whose current path is repositioned at run time.
<index set qualified name>	Identifies the index set whose current path is repositioned at run time.
<name>	Identifies the name of the data set or index set. A name can consist of from 1 to 17 letters, digits, and—depending on the language—either underscores ( ) or hyphens (-). The first character must be a letter. The last character must be either a letter or a digit.

For example, if your program accesses two databases (DB1 and DB2) and each database has a data set named CLASS, you must qualify the data set name by naming the database.

### Target

Use the following syntax to set the location target for the pointer:



The variable elements of the target parameter are explained as follows:

Element	Explanation
BEGINNING	Sets the current path to the start of the file at run time.
ENDING	Sets the current path to the end of the file at run time.
<data set qualified name>	Identifies the data set spanned by the index set and repositions the pointer of the index set to the current record within the spanned data set. Valid only for index sets.

**Programming Examples**

In the program fragments, the data set EMP is set to its beginning point.

**ALGOL Program Fragment**

```
%Declaring local variables

STRING
  DATA_SET_NAME;
  BEGIN_1;
BOOLEAN
  RSLT;

%Assigning values to parameters

DATA_SET_NAME:="EMP";
BEGIN_1:="BEGINNING";

%Declaring library entry point

LIBRARY DMI;
BOOLEAN PROCEDURE DBSET(STRUCTURE,TARGET);
  STRING STRUCTURE, TARGET;
  LIBRARY DMI (ACTUALNAME="ALGOLSET");

%Invoking entry point

RSLT:=DBSET(DATA_SET_NAME,BEGIN_1);
IF RSLT
  THEN EXCEPTIONHANDLER;
```

**COBOL74 or COBOL85 Program Fragment**

*Note: In this example, the variable RESULT is declared as a COMP item, making this example a COBOL74 or COBOL85 program fragment. If RESULT were declared as a COMP item, this fragment could be used as a COBOL program.*

```
***Declaring variables

Ø1 DATA-SET-NAME      PIC X(17) VALUE IS "EMP".
Ø1 BEGIN-1            PIC X(9)  VALUE IS "BEGINNING".
Ø1 RESULT              PIC 9(1)  COMP VALUE IS Ø.

***Invoking entry point

CALL "DBSET OF DMINTERPRETER"
  USING DATA-SET-NAME, BEGIN-1
  GIVING RESULT.
```

## Setting the Current Path (cont.)

---

### FORTRAN77 Program Fragment

\*\*\*Declaring library entry point

```
LOGICAL FUNCTION DBSET (STR,TARGET)
  CHARACTER STR, TARGET
  IN LIBRARY DMI (ACTUALNAME = "FORTRAN77SET")
END
```

\*\*\*Setting variables to initial values

```
CHARACTER DSNAME      *4      /"EMP"/
CHARACTER BEGIN       *4      /"BEGINNING"/
```

\*\*\*Declaring local variables

```
LOGICAL  RSLT
```

\*\*\*Invoking entry point

```
RSLT = DBSET(DSNAME,BEGIN)
IF (RSLT) CALL EXCEPT
```

## Explicitly Freeing Records

When your program locks a record, DMSII guarantees you exclusive use of that record until your program frees (unlocks) the record. Freeing a record releases the record from all lock or secure restrictions so that the record is available to other users and programs.

Depending on which DASDL options are set for the database and whether or not the program is in transaction state, several data management operations implicitly free a record by repositioning the current path. However, if your program is outside of transaction state, you can explicitly free a record by executing a free operation. If there is no concurrency control for the database, you can explicitly free a record within transaction state by executing a free operation.

*Note: If a record is part of a locked or secured structure, you cannot explicitly free the record. The entire structure must be explicitly freed in order for the record to be freed.*

For more details on explicitly and implicitly freeing records, read the *DMSII Application Programming Guide*. For more information on explicitly freeing a structure, read “Explicitly Freeing Structures” later in this section.

### Passing a Parameter

The standard entry point used to explicitly free a record has one required parameter – the structure where the record is stored.

Parameter	Explanation
<structure>	Identifies either a data set (to free a data set record) or a database (to free a global data record).

Note that DMSII frees the current record in the named structure. Therefore, if the structure is a data set, DMSII frees the current data set record. If the structure is a database, DMSII frees the global data record.

Table 3-5 gives the exported names of the entry points, along with the parameter and the data type of the parameter.

Table 3-5. Explicitly Freeing Records

Exported Name of Entry Point	Parameter	Data Type of Parameter
ALGOLFREE	<structure>	STRING
DBFREE	<structure>	DISPLAY
FORTTRAN77FREE	<structure>	CHARACTER

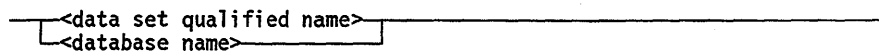
You can explicitly free either a data set record or a global data record. The name of the structure must be unique. The following syntax diagram shows you can use a maximum

## Explicitly Freeing Records (cont.)

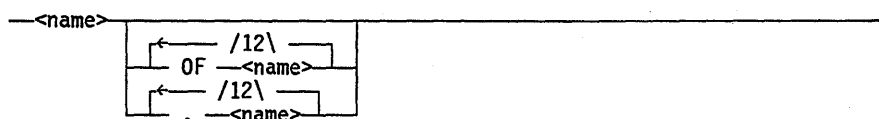
---

of 12 qualifying parameters to uniquely identify the data set name. The database name cannot be qualified.

### <structure>



### <data set qualified name>



The variable elements of the structure parameter are explained as follows:

Element	Explanation
<data set qualified name>	Identifies the data set that is accessed at run time. This element is used only to free a data set record.
<database name>	Identifies the database for which the DMINTERPRETER library was compiled. This element is used only to free a global data record.
<name>	Identifies either the data set or the database. A name can consist of from 1 to 17 letters, digits, and—depending on the language—either underscores (_) or hyphens (-). The first character must be a letter. The last character must be either a letter or a digit.

For example, if your program accesses two databases (DB1 and DB2) and each database has a data set named CLASS, you must qualify the data set name by naming the database.

### Programming Examples

The following program fragments show how to unlock a record in the data set EMP. Because the data set name EMP is unique, you need not qualify it.

**ALGOL Program Fragment**

```

%Declaring local variables

STRING
  DATA_SET_NAME;
BOOLEAN
  RSLT;

%Assigning values to parameters

DATA_SET_NAME:="EMP";

%Declaring library entry point

LIBRARY DMI;
BOOLEAN PROCEDURE DBFREE(STRUCTURE);
  STRING STRUCTURE;
  LIBRARY DMI (ACTUALNAME="ALGOLFREE");

%Invoking entry point

RSLT:=DBFREE(DATA_SET_NAME);
IF RSLT
  THEN EXCEPTIONHANDLER;

```

**COBOL74 or COBOL85 Program Fragment**

*Note: In the following example, the variable RESULT is declared as a COMP item, making this example a COBOL74 or COBOL85 program fragment. If RESULT is declared as a COMP item, this fragment could be used in a COBOL program.*

```

***Declaring variables

Ø1 DATA-SET-NAME      PIC X(17) VALUE IS "EMP".
Ø1 RESULT              PIC 9(1) COMP VALUE IS Ø.

***Invoking entry point

CALL "DBFREE OF DMINTERPRETER"
  USING DATA-SET-NAME
  GIVING RESULT.

```

## Explicitly Freeing Records (cont.)

---

### FORTRAN77 Program Fragment

\*\*\*Declaring library entry point

```
LOGICAL FUNCTION DBFREE (STR)
  CHARACTER STR
  IN LIBRARY DMI (ACTUALNAME = "FORTRAN77FREE")
END
```

\*\*\*Setting variables to initial values

```
CHARACTER DSNAME      *4      /"EMP"/
```

\*\*\*Declaring local variables

```
LOGICAL  RSLT
```

\*\*\*Invoking entry point

```
RSLT = DBFREE(DSNAME)
IF (RSLT) CALL EXCEPT
```

## Explicitly Freeing Structures

When your program locks a structure, DMSII guarantees you exclusive use of that structure and all its records until your program frees (unlocks) the structure. Freeing a structure releases the structure from all lock or secure restrictions so that the structure is available to other users and programs. All locked or secured records in the structure are freed.

You must explicitly free a structure. You cannot implicitly free a structure.

*Note: If a record is part of a locked or secured structure, you cannot explicitly free the record. The entire structure must be explicitly freed in order for the record to be freed.*

### Passing a Parameter

The standard entry point used to explicitly free a record has one required parameter – the name of the data set in which the structure is stored.

Parameter	Explanation
<data set qualified name>	Identifies the data set that is accessed at run time.

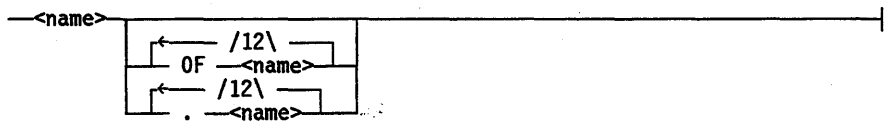
Table 3-6 gives the exported names of the entry points, along with the parameter and the data type of the parameter.

Table 3-6. Explicitly Freeing Structures

Exported Name of Entry Point	Parameter	Data Type of Parameter
ALGOLFREESTR	<data set qualified name>	STRING
DBFREE	<data set qualified name>	DISPLAY
FORTTRAN77FREE	<data set qualified name>	CHARACTER

The name must be unique. You can use a maximum of 12 qualifying parameters to uniquely identify the data set name.

<data set qualified name>





## Explicitly Freeing Structures (cont.)

---

The variable elements of the data set parameter are explained as follows:

Element	Explanation
<data set qualified name>	Identifies the data set that is accessed at run time. This element is used only to free a data set record.
<name>	Identifies the data set. A name can consist of from 1 to 17 letters, digits, and — depending on the language — either underscores (_) or hyphens (-). The first character must be a letter. The last character must be either a letter or a digit.

### Programming Examples

The following program fragments show how to free a structure in the data set EMP. Because the data set name EMP is unique, you need not qualify it.

#### ALGOL Program Fragment

```
%Declaring local variables

STRING
  DATA_SET_NAME;
BOOLEAN
  RSLT;

%Assigning values to parameters

DATA_SET_NAME:="EMP";

%Declaring library entry point

LIBRARY DMI;
BOOLEAN PROCEDURE DBFREESTR(STRUCTURE);
  STRING STRUCTURE;
  LIBRARY DMI (ACTUALNAME="ALGOLFREESTR");

%Invoking entry point

RSLT:=DBFREESTR(DATA_SET_NAME);
IF RSLT
  THEN EXCEPTIONHANDLER;
```

**COBOL74 or COBOL85 Program Fragment**

*Note: In the following example, the variable RESULT is declared as a COMP item, making this example a COBOL74 or COBOL85 program fragment. If RESULT is declared as a COMP item, this fragment could be used in a COBOL program.*

\*\*\*Declaring variables

```
Ø1 DATA-SET-NAME      PIC X(17) VALUE IS "EMP".
Ø1 RESULT              PIC 9(1) COMP VALUE IS Ø.
```

\*\*\*Invoking entry point

```
CALL "DBFREESTR OF DMINTERPRETER"
      USING DATA-SET-NAME
      GIVING RESULT.
```

**FORTRAN77 Program Fragment**

\*\*\*Declaring library entry point

```
LOGICAL FUNCTION DBFSTR (STR)
  CHARACTER STR
  IN LIBRARY DMI (ACTUALNAME = "FORTRAN77FREESTR")
END
```

\*\*\*Setting variables to initial values

```
CHARACTER DSNAME      *4      /"EMP"/
```

\*\*\*Declaring local variables

```
LOGICAL  RSLT
```

\*\*\*Invoking entry point

```
RSLT = DBFSTR(DSNAME)
IF (RSLT) CALL EXCEPT
```

### Finding Records

The find operation retrieves a data record from the database, using a serial search, a random search, or a combined search. A serial search retrieves records using the physical sequence of records. A random search retrieves records based on selection criteria. You also can search for a record using both the physical location and selection criteria. For example, you can retrieve the next record in a data set where the last name is the same as the current record.

The find operation does not lock or secure the record.

*Note: This entry point corresponds to the language extension FIND statement. However, the interpretive interface entry point is more sophisticated. Several restrictions have been eased, and some algorithmic enhancements have been made. These improvements—as well as a discussion of search and retrieval techniques, current records and paths, and user work areas—are detailed in the DMSII Application Programming Guide.*

#### Passing Parameters

The standard entry point used to find a record has three parameters: the direction of the retrieval, the name of the structure, and any condition to be used in the search for the record.

Parameter	Explanation
<direction>	Identifies the physical sequence for a sequential search.
<structure>	Identifies the structure to be searched. It can be either a data set, an index set, or a database.
<condition>	Identifies restrictions for a random search.

Only the name of structure is required; the other parameters are optional. If only the name is given, the current record is moved to the user work area.

Table 3-7 gives the exported names of the entry points, along with the parameters and the data type of each parameter.

Table 3-7. Finding Records

Exported Name of Entry Point	Parameter	Data Type of Parameter
ALGOLFIND	<direction>	STRING
	<structure>	STRING
	<condition>	STRING
DBFIND	<direction>	DISPLAY
	<structure>	DISPLAY
	<condition>	DISPLAY
FORTRAN77FIND	<direction>	CHARACTER
	<structure>	CHARACTER
	<condition>	CHARACTER

**Direction**

The direction parameter is part of the serial search. Use the following syntax to identify the direction of a sequential search:

<direction>

FIRST
NEXT
PRIOR
LAST

The variable elements of the direction parameter are explained as follows:

Element	Explanation
empty or blanks	(Default) Indicates that the find operation does not use a sequential search.  <i>Note: If you leave both the direction and condition parameters blank, DMSII transfers the current record in the specified structure to the user work area.</i>
FIRST	Transfers the first record in the specified structure to the user work area.
NEXT	Transfers the next record in the specified structure to the user work area.
PRIOR	Transfers the prior record in the specified structure to the user work area.
LAST	Transfers the last record in the specified structure to the user work area.

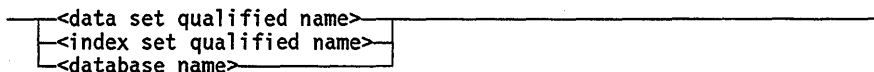
## Finding Records (cont.)

---

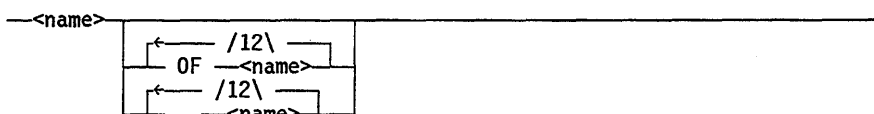
### Structure

The structure to be searched can be either a data set, an index set, or a database. The following syntax diagram shows if the name of the data set or index is not unique, you can use a maximum of 12 qualifying parameters to uniquely identify the data set or index set. The database name cannot be qualified.

<structure>



<qualified name>



The variable elements of the structure parameter are explained as follows:

Element	Explanation
<data set qualified name>	Identifies the data set to be searched at run time.
<index set qualified name>	Identifies the index set to be searched at run time.
<database name>	Identifies the database for which the DMINTERPRETER library was compiled.
<name>	Identifies either a data set, an index set, or a database. A name can consist of from 1 to 17 letters, digits, and—depending on the language—either underscores ( _ ) or hyphens ( - ). The first character must be a letter. The last character must be either a letter or a digit.

For example, if your program accesses two databases (DB1 and DB2) and each database has a data set named CLASS, you must qualify the data set name by naming the database.

### Conditions

Use the following syntax to identify any conditions governing a random search:

<condition>



The variable elements of the condition parameter are explained as follows:

Element	Explanation
empty or blanks	(Default) Indicates that the search does not use any selection criteria.  <i>Note: If you leave both the condition and direction parameters blank, DMSII transfers the current record in the specified structure to the user work area.</i>
<selection expression>	Establishes criteria for the search during the find operation. (For languages that also support language extensions, consult the reference manual for selection expression syntax. Read the <i>DMSII Application Programming Guide</i> for a detailed explanation of a selection expression.)
<link>	Identifies the database structure or data set item that links the current record to a record in another data set.

### Programming Examples

In the following program fragments, the program retrieves the first record in which EMP-NO is equal to 11 in the data set EMP.

#### ALGOL Program Fragment

```

%Declaring local variables

STRING
  FIRST_1,
  DATA_SET_NAME,
  CONDITION_1;
BOOLEAN
  RSLT

%Assigning values to parameters

FIRST_1:="FIRST";
DATA_SET_NAME:="EMP";
CONDITION_1:="EMP-NO=11";

%Declaring library entry point

LIBRARY DMI;
BOOLEAN PROCEDURE DBFIND(DIRECTION,STRUCTURE,CONDITION);
  STRING DIRECTION, STRUCTURE, CONDITION;
  LIBRARY DMI (ACTUALNAME="ALGOLFIND");

%Invoking entry point

RSLT:=DBFIND(FIRST_1,DATA_SET_NAME,CONDITION_1);
IF RSLT
  THEN EXCEPTIONHANDLER;

```

## Finding Records (cont.)

---

### COBOL74 or COBOL85 Program Fragment

*Note: In this example, the variable RESULT is declared as a COMP item, making this example a COBOL74 or COBOL85 program fragment. If RESULT were declared as a COMP-2 item, this fragment could be used in a COBOL program.*

\*\*\*Declaring variables

```
Ø1 FIRST-1          PIC X(5) VALUE IS "FIRST".
Ø1 DATA-SET-NAME   PIC X(17) VALUE IS "EMP".
Ø1 CONDITION-1     PIC X(9) VALUE IS "EMP-NO=11".
Ø1 RESULT          PIC 9(1) COMP VALUE IS Ø.
```

\*\*\*Invoking entry point

```
CALL "DBFIND OF DMINTERPRETER"
      USING FIRST-1, DATA-SET-NAME, CONDITION-1
      GIVING RESULT.
```

### FORTRAN77 Program Fragment

\*\*\*Declaring library entry point

```
LOGICAL FUNCTION DBFIND (STR,ITEM,WHERE)
      CHARACTER STR, ITEM, WHERE
      IN LIBRARY DMI (ACTUALNAME = "FORTRAN77FIND")
END
```

\*\*\*Setting variables to initial values

```
CHARACTER FIRST      *1Ø    /"FIRST"/
*      DSNAME         *4     /"EMP"/
*      COND           *9     /"EMP-NO=11"/
```

\*\*\*Declaring local variables

```
LOGICAL  RSLT
```

\*\*\*Invoking entry point

```
RSLT = DBFIND(FIRST,DSNAME,COND)
IF (RSLT) CALL EXCEPT
```

## Locking Records

The lock operation has two main objectives: to find a designated record and to lock the record. Locking the record prevents other users from simultaneously modifying the record. A program might lock a record (and check the content) before deleting the record or changing the values in a field.

*Note: This entry point functions in the same manner as does the find operation entry point except that a record is locked after it is found. The entry point can locate a record using a random, sequential, or combination search. Refer to "Finding Records" in this section.*

### Passing Parameters

The standard entry point used to find and lock a record has three parameters: the direction of the retrieval, the name of the structure, and any condition to be used in the search for the record.

Parameter	Explanation
<direction>	Identifies the physical sequence for a sequential search.
<structure>	Identifies the structure to be searched. It can be either a data set, an index set, or a database.
<condition>	Identifies restrictions for a random search.

Only the name of structure is required; the other parameters are optional. If only the name is given, the current record is moved to the user work area and locked.

Table 3-8 gives the exported names of the entry points, along with the parameters and the data type of each parameter.

Table 3-8. Locking Records

Exported Name of Entry Point	Parameter	Data Type of Parameter
ALGOLLOCK	<direction>	STRING
	<structure>	STRING
	<condition>	STRING
DBLOCK	<direction>	DISPLAY
	<structure>	DISPLAY
	<condition>	DISPLAY
FORTRAN77LOCK	<direction>	CHARACTER
	<structure>	CHARACTER
	<condition>	CHARACTER



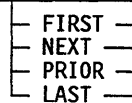
## Locking Records (cont.)

---

### Direction

The direction parameter is part of the serial search. Use the following syntax to identify the direction of a sequential search:

<direction>



The variable elements of the direction parameter are explained as follows:

Element	Explanation
empty or blanks	(Default) Indicates that the lock operation does not use a sequential search.  <i>Note: If you leave both the direction and condition parameters blank, DMSII transfers the current record in the specified structure to the user work area.</i>
FIRST	Transfers the first record in the specified structure to the user work area and locks it.
NEXT	Transfers the next record in the specified structure to the user work area and locks it.
PRIOR	Transfers the prior record in the specified structure to the user work area and locks it.
LAST	Transfers the last record in the specified structure to the user work area and locks it.

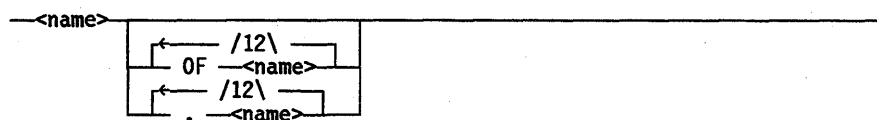
### Structure

The structure to be searched can be either a data set, an index set, or a database. The following syntax diagram shows if the name of the data set or index set is not unique, you can use a maximum of 12 qualifying parameters to uniquely identify the data set or index set. The database name cannot be qualified.

<structure>



<qualified name>



The variable elements of the structure parameter are explained as follows:

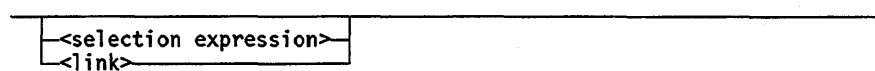
Element	Explanation
<data set qualified name>	Identifies the data set that is searched at run time.
<index set qualified name>	Identifies the index set that is searched at run time.
<database name>	Identifies the database for which the DMINTERPRETER library was compiled.
<name>	Identifies either a data set, an index set, or a database. A name can consist of from 1 to 17 letters, digits, and—depending on the language—either underscores (_) or hyphens (-). The first character must be a letter. The last character must be either a letter or a digit.

For example, if your program accesses two databases (DB1 and DB2) and each database has a data set named CLASS, you must qualify the data set name by naming the database.

#### Conditions

Use the following syntax to identify any conditions governing a random search:

<condition>



The variable elements of the condition parameter are explained as follows:

Element	Explanation
empty or blanks	(Default) Indicates that the search does not use any selection criteria.  <b>Note:</b> <i>If you leave both the direction and condition parameters blank, DMSII transfers the current record in the specified structure to the user work area.</i>
<selection expression>	Establishes criteria for the search during the lock operation. (For languages that also support language extensions, consult the reference manual for selection expression syntax. Read the <i>DMSII Application Programming Guide</i> for a detailed explanation of a selection expression.)
<link>	Identifies the database structure or data set item that links the current record to a record in another data set.

### Programming Examples

In the following program fragments, the program retrieves the first record in which EMP-NO is equal to 11 in the data set EMP. The record is then locked.

#### ALGOL Program Fragment

%Declaring local variables

```
STRING
  FIRST_1,
  DATA_SET_NAME,
  CONDITION_1;
BOOLEAN
  RSLT
```

%Assigning values to parameters

```
FIRST_1:="FIRST";
DATA_SET_NAME:="EMP";
CONDITION_1:="EMP-NO=11";
```

%Declaring library entry point

```
LIBRARY DMI;
BOOLEAN PROCEDURE DBLOCK(DIRECTION,STRUCTURE,CONDITION);
  STRING DIRECTION, STRUCTURE, CONDITION;
  LIBRARY DMI (ACTUALNAME="ALGOLLOCK");
```

%Invoking entry point

```
RSLT:=DBLOCK(FIRST_1,DATA_SET_NAME,CONDITION_1);
IF RSLT
  THEN EXCEPTIONHANDLER;
```

**COBOL74 or COBOL85 Program Fragment**

*Note: In the following example, the variable RESULT is declared as a COMP item, making this example a COBOL74 or COBOL85 program fragment. If RESULT were declared as a COMP-2 item, this fragment could be used in a COBOL program.*

\*\*\*Declaring variables

```

Ø1 FIRST-1          PIC X(5)  VALUE IS "FIRST".
Ø1 DATA-SET-NAME   PIC X(17) VALUE IS "EMP".
Ø1 CONDITION-1      PIC X(9)  VALUE IS "EMP-NO=11".
Ø1 RESULT           PIC 9(1)  COMP VALUE IS Ø.
    
```

\*\*\*Invoking entry point

```

CALL "DBLOCK OF DMINTERPRETER"
      USING FIRST-1, DATA-SET-NAME, CONDITION-1
      GIVING RESULT.
    
```

**FORTRAN77 Program Fragment**

\*\*\*Declaring library entry point

```

LOGICAL FUNCTION DBLOCK (STR,ITEM,WHERE)
      CHARACTER STR, ITEM, WHERE
      IN LIBRARY DMI (ACTUALNAME = "FORTRAN77LOCK")
END
    
```

\*\*\*Setting variables to initial values

```

CHARACTER FIRST      *1Ø    /"FIRST"/
*      DSNAME        *4     /"EMP"/
*      COND          *9     /"EMP-NO=11"/
    
```

\*\*\*Declaring local variables

```

LOGICAL  RSLT
    
```

\*\*\*Invoking entry point

```

RSLT = DBLOCK(FIRST,DSNAME,COND)
IF (RSLT) CALL EXCEPT
    
```

## Locking Structures

The lock structure operation has two main objectives: to find the designated structure and to lock the structure. Locking a structure prevents other users from simultaneously modifying the structure.

All records in the structure are locked. The records are freed only when the structure is explicitly freed or the database is closed. The records are not implicitly freed after the DMSII executes an end transaction operation.

Read "Freeing Structures" in this section for more information on explicitly freeing a structure. Read "Closing a Database" in this section for more details on closing a database.

### Passing a Parameter

The standard entry point used to find and lock a structure has one parameter — the name of the data set that is accessed at run time.

Parameter	Explanation
<data set qualified name>	Identifies the data set to be accessed at run time.

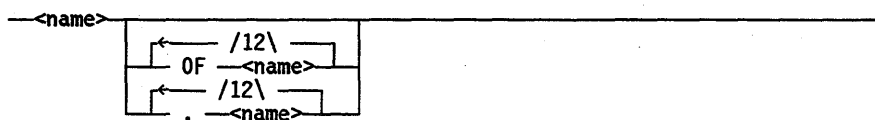
Table 3-9 gives the exported names of the entry points, along with the parameters and the data type of each parameter.

Table 3-9. Locking Structures

Exported Name of Entry Point	Parameter	Data Type of Parameter
ALGOLLOCKSTR	<data set qualified name>	STRING
DBLOCKSTR	<data set qualified name>	DISPLAY
FORTTRAN77LOCKSTR	<data set qualified name>	CHARACTER

The data set must be uniquely identified. You can use a maximum of 12 qualifying parameters to uniquely identify the data set.

<data set qualified name>



The variable elements of the data set parameter are explained as follows:

Element	Explanation
<data set qualified name>	Identifies the data set that is searched at run time.
<name>	Identifies the data set. A name can consist of from 1 to 17 letters, digits, and — depending on the language — either underscores ( _ ) or hyphens ( - ). The first character must be a letter. The last character must be either a letter or a digit.

### Programming Examples

In the following program fragments, the program retrieves and locks the data set `DATA_SET_NAME`. All records in the data set are locked. The entire structure must be explicitly freed in order to free any of the locked records.

#### ALGOL Program Fragment

```
%Declaring local variables

STRING
  DATA_SET_NAME,
BOOLEAN
  RSLT

%Assigning values to parameters

DATA_SET_NAME:="EMP";

%Declaring library entry point

LIBRARY DMI;
BOOLEAN PROCEDURE DBLOCKSTR(STRUCTURE);
  STRING STRUCTURE
  LIBRARY DMI (ACTUALNAME="ALGOLLOCKSTR");

%Invoking entry point

RSLT:=DBLOCKSTR(DATA_SET_NAME);
IF RSLT
  THEN EXCEPTIONHANDLER;
```

### COBOL74 or COBOL85 Program Fragment

*Note: In the following example, the variable RESULT is declared as a COMP item, making this example a COBOL74 or COBOL85 program fragment. If RESULT were declared as a COMP-2 item, this fragment could be used in a COBOL program.*

\*\*\*Declaring variables

```
Ø1 DATA-SET-NAME      PIC X(17) VALUE IS "EMP".
Ø1 RESULT              PIC 9(1)  COMP VALUE IS Ø.
```

\*\*\*Invoking entry point

```
CALL "DBLOCKSTR OF DMINTERPRETER"
      USING DATA-SET-NAME
      GIVING RESULT.
```

### FORTRAN77 Program Fragment

\*\*\*Declaring library entry point

```
LOGICAL FUNCTION DBSTR (ITEM)
  CHARACTER ITEM
  IN LIBRARY DMI (ACTUALNAME = "FORTRAN77LOCKSTR")
END
```

\*\*\*Setting variables to initial values

```
CHARACTER DSNAME      *4      /"EMP"/
```

\*\*\*Declaring local variables

```
LOGICAL  RSLT
```

\*\*\*Invoking entry point

```
RSLT = DBSTR(DSNAME)
IF (RSLT) CALL EXCEPT
```

## Securing Records

The secure operation has two main objectives: to find a designated record and to secure the record. Securing a record guarantees no other user can modify the record but allows other users to access the record and retrieve information.

*Note: This entry point functions in the same manner as does the find operation entry point except that a record is secured after it is found. The entry point can locate a record using a random, sequential, or combination search. Refer to "Finding Records" in this section.*

### Passing Parameters

The standard entry point used to find and secure a record has three parameters: the direction of the retrieval, the name of the structure, and any condition to be used in the search for the record.

Parameter	Explanation
<direction>	Identifies the physical sequence for a sequential search.
<structure>	Identifies the structure to be searched. It can be either a data set, an index set, or a database.
<condition>	Identifies restrictions for a random search.

Only the name of structure is required; the other parameters are optional. If only the name is given, the current record is moved to the user work area and secured.

Table 3-10 gives the exported names of the entry points, along with the parameters and the data type of each parameter.

**Table 3-10. Securing Records**

Exported Name of Entry Point	Parameter	Data Type of Parameter
ALGOLSECURE	<direction>	STRING
	<structure>	STRING
	<condition>	STRING
DBSECURE	<direction>	DISPLAY
	<structure>	DISPLAY
	<condition>	DISPLAY
FORTRAN77SECURE	<direction>	CHARACTER
	<structure>	CHARACTER
	<condition>	CHARACTER



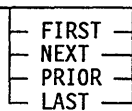
## Securing Records (cont.)

---

### Direction

The direction parameter is part of the serial search. Use the following syntax to identify the direction of a sequential search:

<direction>



The variable elements of the direction parameter are explained as follows:

Element	Explanation
empty or blanks	(Default) Indicates that the secure operation does not use a sequential search.  <i>Note: If you leave both the direction and condition parameters blank, DMSII transfers the current record in the specified structure to the user work area.</i>
FIRST	Transfers the first record in the specified structure to the user work area and secures it.
NEXT	Transfers the next record in the specified structure to the user work area and secures it.
PRIOR	Transfers the prior record in the specified structure to the user work area and secures it.
LAST	Transfers the last record in the specified structure to the user work area and secures it.

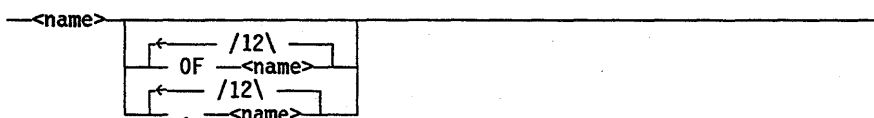
### Structure

The structure to be searched can be either a data set, an index set, or a database. The following syntax diagram shows if the name of the data set or index set is not unique, you can use a maximum of 12 qualifying parameters to uniquely identify the data set or index set. The database name cannot be qualified.

<structure>



<qualified name>



The variable elements of the structure parameter are explained as follows:

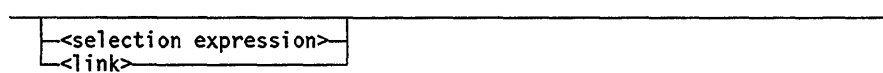
Element	Explanation
<data set qualified name>	Identifies the data set that is searched at run time.
<index set qualified name>	Identifies the index set that is searched at run time.
<database name>	Identifies the database for which the DMINTERPRETER library was compiled.
<name>	Identifies either a data set, an index set, or a database. A name can consist of from 1 to 17 letters, digits, and—depending on the language—either underscores (_) or hyphens (-). The first character must be a letter. The last character must be either a letter or a digit.

For example, if your program accesses two databases (DB1 and DB2) and each database has a data set named CLASS, you must qualify the data set name by naming the database.

**Conditions**

Use the following syntax to identify any conditions governing a random search:

<condition>



The variable elements of the condition parameter are explained as follows:

Element	Explanation
empty or blanks	(Default) Indicates that the search does not use any selection criteria.  <b>Note:</b> <i>If you leave both the direction and condition parameters blank, DMSII transfers the current record in the specified structure to the user work area.</i>
<selection expression>	Establishes criteria for the search during the secure operation. (For languages that also support language extensions, consult the reference manual for selection expression syntax. Read the <i>DMSII Application Programming Guide</i> for a detailed explanation of a selection expression.)
<link>	Identifies the database structure or data set item that links the current record to a record in another data set.

### Programming Examples

In the following program fragments, the program retrieves the first record in which EMP-NO is equal to 11 in the data set EMP. The record is then secured.

#### ALGOL Program Fragment

```
%Declaring local variables
```

```
STRING  
    FIRST_1,  
    DATA_SET_NAME,  
    CONDITION_1;  
BOOLEAN  
    RSLT
```

```
%Assigning values to parameters
```

```
FIRST_1:="FIRST";  
DATA_SET_NAME:="EMP";  
CONDITION_1:="EMP-NO=11";
```

```
%Declaring library entry point
```

```
LIBRARY DMI;  
BOOLEAN PROCEDURE DBSECURE (DIRECTION,STRUCTURE,CONDITION);  
    STRING DIRECTION, STRUCTURE, CONDITION;  
    LIBRARY DMI (ACTUALNAME="ALGOLSECURE");
```

```
%Invoking entry point
```

```
RSLT:=DBSECURE(FIRST_1,DATA_SET_NAME,CONDITION_1);  
IF RSLT  
    THEN EXCEPTIONHANDLER;
```

**COBOL74 or COBOL85 Program Fragment**

*Note: In the following example, the variable RESULT is declared as a COMP item, making this example a COBOL74 or COBOL85 program fragment. If RESULT were declared as a COMP-2 item, this fragment could be used in a COBOL program.*

\*\*\*Declaring variables

```

Ø1 FIRST-1          PIC X(5) VALUE IS "FIRST".
Ø1 DATA-SET-NAME  PIC X(17) VALUE IS "EMP".
Ø1 CONDITION-1     PIC X(9) VALUE IS "EMP-NO=11".
Ø1 RESULT          PIC 9(1) COMP VALUE IS Ø.
    
```

\*\*\*Invoking entry point

```

CALL "DBSECURE OF DMINTERPRETER"
     USING FIRST-1, DATA-SET-NAME, CONDITION-1
     GIVING RESULT.
    
```

**FORTRAN77 Program Fragment**

\*\*\*Declaring library entry point

```

LOGICAL FUNCTION DBSEC (STR,ITEM,WHERE)
  CHARACTER STR, ITEM, WHERE
  IN LIBRARY DMI (ACTUALNAME = "FORTRAN77SECURE")
END
    
```

\*\*\*Setting variables to initial values

```

CHARACTER FIRST      *1Ø    /"FIRST"/
  *      DSNAME      *4     /"EMP"/
  *      COND        *9     /"EMP-NO=11"/
    
```

\*\*\*Declaring local variables

```

LOGICAL  RSLT
    
```

\*\*\*Invoking entry point

```

RSLT = DBSEC(FIRST,DSNAME,COND)
IF (RSLT) CALL EXCEPT
    
```

## Securing Structures

The secure structure operation has two main objectives: to find the designated structure and to secure the structure. Securing a structure prevents other users from simultaneously modifying any records in the structure. However, other users can access and retrieve information from records in the structure.

All records in the structure are secured. The records are freed only when the structure is explicitly freed or the database is closed. The records are not implicitly freed after the DMSII executes an end transaction operation.

Read "Freeing Structures" in this section for more information on explicitly freeing a structure. Read "Closing a Database" in this section for more details on closing a database.

### Passing a Parameter

The standard entry point used to find and secure a structure has one parameter – the name of the data set that is accessed at run time.

Parameter	Explanation
<data set qualified name>	Identifies the data set to be accessed at run time.

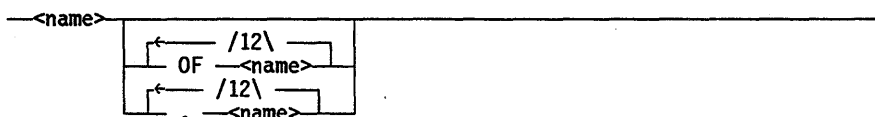
Table 3-11 gives the exported names of the entry points, along with the parameters and the data type of each parameter.

Table 3-11. Securing Structures

Exported Name of Entry Point	Parameter	Data Type of Parameter
ALGOLSECURESTR	<data set qualified name>	STRING
DBSECURESTR	<data set qualified name>	DISPLAY
FORTTRAN77SECURESTR	<data set qualified name>	CHARACTER

The data set must be uniquely identified. You can use a maximum of 12 qualifying parameters to uniquely identify the data set.

<data set qualified name>



The variable elements of the data set parameter are explained as follows:

Element	Explanation
<data set qualified name>	Identifies the data set that is searched at run time.
<name>	Identifies the data set. A name can consist of from 1 to 17 letters, digits, and —depending on the language—either underscores ( _ ) or hyphens (-). The first character must be a letter. The last character must be either a letter or a digit.

### Programming Examples

In the following program fragments, the program retrieves and secures the data set `DATA_SET_NAME`. All records in the data set are secured: other users can access information but cannot modify information.

#### ALGOL Program Fragment

```

%Declaring local variables

STRING
  DATA_SET_NAME,
BOOLEAN
  RSLT

%Assigning values to parameters

DATA_SET_NAME:="EMP";

%Declaring library entry point

LIBRARY DMI;
BOOLEAN PROCEDURE DBSECURESTR(STRUCTURE);
  STRING STRUCTURE
  LIBRARY DMI (ACTUALNAME="ALGOLSECURESTR");

%Invoking entry point

RSLT:=DBSECURESTR(DATA_SET_NAME);
IF RSLT
  THEN EXCEPTIONHANDLER;

```

**COBOL74 or COBOL85 Program Fragment**

*Note: In the following example, the variable RESULT is declared as a COMP item, making this example a COBOL74 or COBOL85 program fragment. If RESULT were declared as a COMP-2 item, this fragment could be used in a COBOL program.*

\*\*\*Declaring variables

```
Ø1 DATA-SET-NAME      PIC X(17) VALUE IS "EMP".
Ø1 RESULT              PIC 9(1)  COMP VALUE IS Ø.
```

\*\*\*Invoking entry point

```
CALL "DBSECURESTR OF DMINTERPRETER"
      USING DATA-SET-NAME
      GIVING RESULT.
```

**FORTRAN77 Program Fragment**

\*\*\*Declaring library entry point

```
LOGICAL FUNCTION DBSSTR (ITEM)
  CHARACTER ITEM
  IN LIBRARY DMI (ACTUALNAME = "FORTRAN77SECURESTR")
END
```

\*\*\*Setting variables to initial values

```
CHARACTER DSNAME      *4      /"EMP"/
```

\*\*\*Declaring local variables

```
LOGICAL  RSLT
```

\*\*\*Invoking entry point

```
RSLT = DBSSTR(DSNAME)
IF (RSLT) CALL EXCEPT
```

## Begin Transaction – Entering Transaction State

For each audited database, DMSII maintains a log (audit trail) of all activities that update the database. For example, the audit trail notes all records that are deleted or added and all data changes. This audit trail is used by the recovery software to prevent database corruption.

When a program opens an audited database to update it, the database can be in one of two states: protected or transaction. In a protected state, no update activity can be performed. By default, an audited database is opened in a protected state.

An application program must specifically place the audited database in the transaction state in order to perform any update activities, such as storing or deleting records.

**Notes:**

- *DMSII ignores invocations to begin or end transaction state for unaudited databases.*
- *Read the DMSII Application Programming Guide for transaction and recovery guidelines.*

### Passing a Parameter

The standard entry point used to place an opened, audited database in transaction state has one required parameter – the audit parameter.

Parameter	Explanation
<audit>	Specifies whether an image of the current restart data set record is to be audited. (The restart data set is used for restart routines. Consult the <i>DMSII DASDL Reference Manual</i> for details about the content of the restart data set. Consult the <i>DMSII Application Programming Guide</i> for programming considerations.)

Table 3-12 gives the exported names of the entry points, along with the parameter and the data type of the parameter.

**Table 3–12. Begin Transaction – Entering Transaction State**

Exported Name of Entry Point	Parameter	Data Type of Parameter
ALGOLBEGINTRANSACTION	<audit>	STRING
DBBEGINTRANSACTION	<audit>	DISPLAY
FORTRAN77BEGINTRANSACTION	<audit>	CHARACTER



## Begin Transaction – Entering Transaction State (cont.)

---

Use the following syntax to begin transaction state:

<audit>

```
┌── AUDIT ───┐
└── NOAUDIT ─┘
```

*Note:* In order for your program to enter transaction state, you must set the value of the audit parameter to *AUDIT*.

The variable element of the audit parameter is explained as follows:

Element	Explanation
AUDIT	DMSII audits an image of the current restart data set record.
NOAUDIT	DMSII does not perform an audit.

### Programming Examples

The following program fragments place an opened, audited database in transaction state. DMSII audits an image of the current restart data set record.

#### ALGOL Program Fragment

```
%Declaring local variables

STRING
  P_AUDIT;
BOOLEAN
  RSLT;

%Assigning values to parameters

P_AUDIT:="AUDIT";

%Declaring library entry point

LIBRARY DMI;
BOOLEAN PROCEDURE DBBEGINTRANSACTION (AUDIT);
  STRING AUDIT;
  LIBRARY DMI (ACTUALNAME="ALGOLBEGINTRANSACTION")

%Invoking entry point

RSLT:=DBBEGINTRANSACTION(P_AUDIT);
IF RSLT
  THEN EXCEPTIONHANDLER;
```

**COBOL74 or COBOL85 Program Fragment**

*Note: In the following example, the variable RESULT is declared as a COMP item, making this example a COBOL74 or COBOL85 program fragment. If RESULT were declared as a COMP-2 item, this fragment could be used in a COBOL program.*

\*\*\*Declaring variables

```
Ø1 P-AUDIT          PIC X(5) VALUE IS "AUDIT".
Ø1 RESULT          PIC 9(1) COMP VALUE IS Ø.
```

\*\*\*Invoking entry point

```
CALL "DBBEGINTRANSACTION OF DMINTERPRETER"
      USING P-AUDIT
      GIVING RESULT.
```

**FORTRAN77 Program Fragment**

\*\*\*Declaring library entry point

```
LOGICAL FUNCTION DBBTR (AUDIT)
  CHARACTER AUDIT
  IN LIBRARY DMI (ACTUALNAME = "FORTRAN77BEGINTRANSACTION")
END
```

\*\*\*Setting variables to initial values

```
CHARACTER AUDIT      *1Ø      /"AUDIT"/
```

\*\*\*Declaring local variables

```
LOGICAL  RSLT
```

\*\*\*Invoking entry point

```
RSLT = DBBTR(AUDIT)
IF (RSLT) CALL EXCEPT
```

# Aborting Transactions

An abort transaction operation discards all updates made in a transaction and brings the database back to the point immediately before the begin transaction operation that initiated the update activity. The program and all concurrent programs continue running. All other transactions are processed.

### Notes:

- *If the DASDL INDEPENDENTTRANS option is not set for the database, any attempt to abort a transaction is ignored.*
- *DMSII ignores invocations to abort transactions for unaudited databases.*
- *For audited databases, the database must be in transaction state to perform an abort operation.*
- *Read the DMSII Application Programming Guide for transaction and recovery guidelines.*

### Passing Parameters

The standard entry point used to abort a transaction has no parameters. Table 3-13 gives the exported names of the entry points.

Table 3-13. Export Names of Entry Points Used to Abort a Transaction

Export Names
ALGOLABORTTRANSACTION
DBABORTTRANSACTION
FORTTRAN77ABORTTRANSACTION

### Programming Examples

The following program fragments abort a transaction. The audited database must be in transaction state.

**ALGOL Program Fragment**

```
%Declaring local variables

BOOLEAN
    RSLT;

%Declaring library entry point

LIBRARY DMI;
BOOLEAN PROCEDURE DBABORTTRANSACTION;
    LIBRARY DMI (ACTUALNAME="ALGOLBEGINTRANSACTION")

%Invoking entry point

RSLT:=DBABORTTRANSACTION;
IF RSLT
    THEN EXCEPTIONHANDLER;
```

**COBOL74 or COBOL85 Program Fragment**

*Note: In the following example, the variable RESULT is declared as a COMP item, making this example a COBOL74 or COBOL85 program fragment. If RESULT were declared as a COMP-2 item, this fragment could be used in a COBOL program.*

```
***Declaring variables

Ø1 RESULT          PIC 9(1) COMP VALUE IS Ø.

***Invoking entry point

CALL "DBABORTTRANSACTION OF DMINTERPRETER"
    GIVING RESULT.
```

**FORTRAN77 Program Fragment**

```
***Declaring library entry point

LOGICAL FUNCTION DBATR
    IN LIBRARY DMI (ACTUALNAME = "FORTRAN77ABORTTRANSACTION")
END

***Declaring local variables

LOGICAL RSLT

***Invoking entry point

RSLT = DBATR
IF (RSLT) CALL EXCEPT
```

# Saving Transaction Points

A save transaction point operation writes a savepoint record to the audit trail. The savepoint applies only to the current transaction and does not affect hold/load recovery. You can write more than one savepoint record to the audit trail within a transaction.

### Notes:

- *If the DASDL INDEPENDENTTRANS option is not set for the database, any attempt to create a savepoint is ignored.*
- *DMSII ignores invocations to save transaction points for unaudited databases.*
- *For audited databases, the database must be in transaction state to perform a save transaction point operation.*
- *Read the DMSII Application Programming Guide for transaction and recovery guidelines.*

### Passing a Parameter

The standard entry point used to save a transaction point has one required parameter—a unique identifier for the savepoint.

Parameter	Explanation
<savepoint>	Specifies the unique number assigned to the savepoint.

Table 3-14 gives the exported names of the entry points, along with the parameter and the data type of the parameter.

Table 3-14. Saving Transaction Points

Exported Name of Entry Point	Parameter	Data Type of Parameter
ALGOLSAVETRPOINT	<savepoint>	INTEGER
DBSAVETRPOINT	<savepoint>	DISPLAY
FORTRAN77SAVETRPOINT	<savepoint>	INTEGER

Use the following syntax to create a savepoint record:

<savepoint>

— <integer> —————|

The variable element of the savepoint parameter is explained as follows:

Element	Explanation
<integer>	A user-assigned whole number that uniquely identifies the savepoint record. The number must be enclosed within parentheses.

### Programming Examples

The following program fragments write a savepoint record to the audit trail. The database must be in transaction state.

#### ALGOL Program Fragment

```
%Declaring local variables

INTEGER
  SAVE_POINT;
BOOLEAN
  RSLT;

%Assigning values to parameters

SAVE_POINT:="872";

%Declaring library entry point

LIBRARY DMI;
BOOLEAN PROCEDURE DBSAVETRPOINT (SAVE_POINT);
  INTEGER SAVE_POINT;
  LIBRARY DMI (ACTUALNAME="ALGOLSAVETRPOINT")

%Invoking entry point

RSLT:=DBSAVETRPOINT(SAVE_POINT);
IF RSLT
  THEN EXCEPTIONHANDLER;
```

## Saving Transaction Points (cont.)

---

### COBOL74 or COBOL85 Program Fragment

*Note: In the following example, the variable RESULT is declared as a COMP item, making this example a COBOL74 or COBOL85 program fragment. If RESULT were declared as a COMP-2 item, this fragment could be used in a COBOL program.*

\*\*\*Declaring variables

```
Ø1 S-POINT          PIC X(1) VALUE IS "1".
Ø1 RESULT           PIC 9(1) COMP VALUE IS Ø.
```

\*\*\*Invoking entry point

```
CALL "DBSAVETRPOINT OF DMINTERPRETER"
      USING S-POINT
      GIVING RESULT.
```

### FORTRAN77 Program Fragment

\*\*\*Declaring library entry point

```
LOGICAL FUNCTION DBSAVE (SPOINT)
  STRING SPOINT
  IN LIBRARY DMI (ACTUALNAME = "FORTRAN77SAVETRPOINT")
END
```

\*\*\*Setting variables to initial values

```
INTEGER SPOINT      *1Ø    /"1"/
```

\*\*\*Declaring local variables

```
LOGICAL  RSLT
```

\*\*\*Invoking entry point

```
RSLT = DBSAVE(SPOINT)
IF (RSLT) CALL EXCEPT
```

## Canceling Transactions Back to Savepoints

Cancel transaction point operations abort the current transaction back to a specified savepoint. The program and all concurrent programs continue running. All other transactions are processed.

**Notes:**

- *If the DASDL INDEPENDENTTRANS option is not set for the database, any attempt to cancel a transaction is ignored.*
- *DMSII ignores invocations to cancel transactions for unaudited databases.*
- *For audited databases, the database must be in transaction state to perform a cancel transaction point operation.*
- *Read the DMSII Application Programming Guide for transaction and recovery guidelines.*

### Passing a Parameter

The standard entry point used to cancel a transaction point has one required parameter—a unique identifier for the savepoint.

Parameter	Explanation
<savepoint>	Specifies the unique number assigned to the savepoint.

Table 3-15 gives the exported names of the entry points, along with the parameter and the data type of the parameter.

**Table 3–15. Canceling Transactions Back to Savepoints**

Exported Name of Entry Point	Parameter	Data Type of Parameter
ALGOLCANCELTRPOINT	<savepoint>	INTEGER
DBCANCELTRPOINT	<savepoint>	DISPLAY
FORTRAN77CANCELTRPOINT	<savepoint>	INTEGER

Use the following syntax to identify a savepoint record:

```
<savepoint>
— <integer> _____|
```



## Canceling Transactions Back to Savepoints (cont.)

---

The variable element of the savepoint parameter is explained as follows:

Element	Explanation
<integer>	A user-assigned whole number that uniquely identifies the savepoint record. The number must be enclosed within parentheses.

If no corresponding savepoint record is found or if the savepoint parameter is not entered, all updates performed during the current transaction start are aborted. The cancel transaction point operation is executed as if it were an abort transaction operation.

### Programming Examples

The following program fragments write a savepoint record to the audit trail. The database must be in transaction state.

#### ALGOL Program Fragment

```
%Declaring local variables

INTEGER
  CANCEL_POINT;
BOOLEAN
  RSLT;

%Assigning values to parameters

CANCEL_POINT:="872";

%Declaring library entry point

LIBRARY DMI;
BOOLEAN PROCEDURE DBCANCELTRPOINT (CANCEL_POINT);
  INTEGER CANCEL_POINT;
  LIBRARY DMI (ACTUALNAME="ALGOLCANCELTRPOINT")

%Invoking entry point

RSLT:=DBCANCELTRPOINT(CANCEL_POINT);
IF RSLT
  THEN EXCEPTIONHANDLER;
```

**COBOL74 or COBOL85 Program Fragment**

*Note: In the following example, the variable RESULT is declared as a COMP item, making this example a COBOL74 or COBOL85 program fragment. If RESULT were declared as a COMP-2 item, this fragment could be used in a COBOL program.*

\*\*\*Declaring variables

```
Ø1 C-POINT          PIC X(1) VALUE IS "1".
Ø1 RESULT           PIC 9(1) COMP VALUE IS Ø.
```

\*\*\*Invoking entry point

```
CALL "DBCANCELTRPOINT OF DMINTERPRETER"
      USING S-POINT
      GIVING RESULT.
```

**FORTRAN77 Program Fragment**

\*\*\*Declaring library entry point

```
LOGICAL FUNCTION DBCNCL (CPOINT)
  STRING CPOINT
  IN LIBRARY DMI (ACTUALNAME = "FORTRAN77CANCELTRPOINT")
END
```

\*\*\*Setting variables to initial values

```
INTEGER CPOINT      *1Ø      /"1"/
```

\*\*\*Declaring local variables

```
LOGICAL RSLT
```

\*\*\*Invoking entry point

```
RSLT = DBCNCL(CPOINT)
IF (RSLT) CALL EXCEPT
```

# Deleting Data Records

The deletion operation has three main objectives: to find a designated record, lock the record, and then delete the record. However, you cannot use the deletion operation to delete a global data record.

For audited databases, the database must be in transaction state to perform a deletion operation.

### *Notes:*

- *This entry point functions in the same manner as does the find operation entry point except that a record is locked and deleted after it is found. The entry point can locate the record using a random, sequential, or combination search. Refer to "Finding Records" in this section.*
- *The deletion operation leaves an image of the deleted record in the user work area. This image is used by the re-create operation. See "Re-creating Records" later in this section.*

### Passing Parameters

The standard entry point used to delete a record has three parameters: the direction of the retrieval, the name of the data set, and any condition to be used in the search for the record.

Parameter	Explanation
<direction>	Identifies the physical sequence for a sequential search.
<structure>	Identifies the structure to be searched. It can be either a data set, an index set, or a database.
<condition>	Gives the special criteria for a random search.

Only the name of structure is required; the other parameters are optional. If only the name is given, the current record is moved to the user work area and deleted.

Table 3-16 gives the exported names of the entry points, along with the parameters and the data type of each parameter.

Table 3-16. Deleting a Data Record

Exported Name of Entry Point	Parameter	Data Type of Parameter
ALGOLDELETE	<direction>	STRING
	<structure>	STRING
	<condition>	STRING
DBDELETE	<direction>	DISPLAY
	<structure>	DISPLAY
	<condition>	DISPLAY
FORTRAN77DELETE	<direction>	CHARACTER
	<structure>	CHARACTER
	<condition>	CHARACTER

**Direction**

The direction parameter is part of the serial search. Use the following syntax to identify the direction of a sequential search:

<direction>

—	FIRST	—
—	NEXT	—
—	PRIOR	—
—	LAST	—

The variable elements of the direction parameter are explained as follows:

Element	Explanation
empty or blanks	(Default) Indicates that the deletion operation does not use a sequential search.  <i>Note: If you leave both the direction and condition parameters blank, DMSII transfers the current record in the specified structure to the user work area.</i>
FIRST	Transfers the first record in the specified structure to the user work area before deleting the record.
NEXT	Transfers the next record in the specified structure to the user work area before deleting the record.
PRIOR	Transfers the prior record in the specified structure to the user work area before deleting the record.
LAST	Transfers the last record in the specified structure to the user work area before deleting the record.

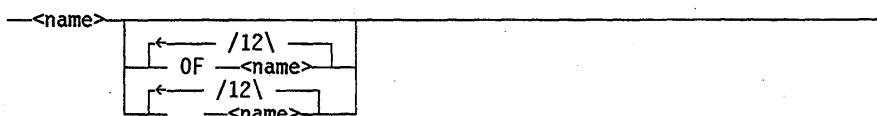
## Deleting Data Records (cont.)

---

### Structure

The record must be in a data set. The data set name must be unique. The following syntax diagram shows you can use a maximum of 12 qualifying parameters to uniquely identify the data set:

<data set qualified name>



The variable element of the data set name is explained as follows:

Element	Explanation
<name>	Identifies the data set that is searched for the record at run time. The name can consist of from 1 to 17 letters, digits, and—depending on the language—either underscores ( _ ) or hyphens (-). The first character must be a letter. The last character must be either a letter or a digit.

For example, if your program accesses two databases (DB1 and DB2) and each database has a data set named CLASS, you must qualify the data set name by naming the database.

### Conditions

Use the following syntax to identify any conditions governing a random search:

<condition>



The variable elements of the condition parameter are explained as follows:

Element	Explanation
empty or blanks	(Default) Indicates that the search does not use any selection criteria.  <b>Note:</b> <i>If you leave both the direction and condition parameters blank, DMSII transfers the current record in the specified structure to the user work area.</i>
<selection expression>	Establishes criteria for the search during the deletion operation. (For languages that also support language extensions, consult the reference manual for selection expression syntax. Read the <i>DMSII Application Programming Guide</i> for a detailed explanation of a selection expression.)
<link>	Identifies the database structure or data set item that links the current record to a record in another data set.

### Programming Examples

In these program fragments, the entry point is used to find the first record in the data set EMP in which the EMP-NO is 11. Unless that record is a global data record, it is deleted.

#### ALGOL Program Fragment

```
%Declaring local variables

STRING
    FIRST_1,
    DATA_SET_NAME,
    CONDITION_1;
BOOLEAN
    RSLT

%Assigning values to parameters

FIRST_1:="FIRST";
DATA_SET_NAME:="EMP";
CONDITION_1:="EMP-NO=11";

%Declaring library entry point

LIBRARY DMI;
BOOLEAN PROCEDURE DBDELETE(DIRECTION,STRUCTURE,CONDITION);
    STRING DIRECTION, STRUCTURE, CONDITION;
    LIBRARY DMI (ACTUALNAME="ALGOLDELETE");

%Invoking entry point

RSLT:=DBDELETE(FIRST_1,DATA_SET_NAME,CONDITION_1);
IF RSLT
    THEN EXCEPTIONHANDLER;
```

## Deleting Data Records (cont.)

---

### COBOL74 or COBOL85 Program Fragment

*Note: In the following example, the variable RESULT is declared as a COMP item, making this example a COBOL74 or COBOL85 program fragment. If RESULT were declared as a COMP-2 item, this fragment could be used in a COBOL program.*

\*\*\*Declaring variables

```
Ø1 FIRST-1          PIC X(5) VALUE IS "FIRST".
Ø1 DATA-SET-NAME   PIC X(17) VALUE IS "EMP".
Ø1 CONDITION-1     PIC X(9) VALUE IS "EMP-NO=11".
Ø1 RESULT          PIC 9(1) COMP VALUE IS Ø.
```

\*\*\*Invoking entry point

```
CALL "DBDELETE OF DMINTERPRETER"
      USING FIRST-1, DATA-SET-NAME, CONDITION-1
      GIVING RESULT.
```

### FORTRAN77 Program Fragment

\*\*\*Declaring library entry point

```
LOGICAL FUNCTION DBDEL (STR,ITEM,WHERE)
      CHARACTER STR, ITEM, WHERE
      IN LIBRARY DMI (ACTUALNAME = "FORTRAN77DELETE")
END
```

\*\*\*Setting variables to initial values

```
CHARACTER FIRST      *1Ø    /"FIRST"/
*      DSNAME        *4     /"EMP"/
*      COND          *9     /"EMP-NO=11"/
```

\*\*\*Declaring local variables

```
LOGICAL  RSLT
```

\*\*\*Invoking entry point

```
RSLT = DBDEL(FIRST,DSNAME,COND)
IF (RSLT) CALL EXCEPT
```

## Re-creating Records

In existing records, if an item is not required, you can change the value of the item by finding and locking the record, moving data from the application program directly into the item, and storing the record back in the database. Any item in the original record that is not specifically changed remains the same.

However, if a required item is part of a set in which duplicates are not allowed or if the DASDL *KEYCHANGEOK* option is not set, you must reconstruct the entire record and specifically enter the values for all modified items. To modify a required item, your program must do the following:

1. Delete the original record.
2. Immediately re-create the record format.
3. Use the DBPUT data transfer entry points to place values into the re-created record. (See Section 4, "Transferring Data," for details of DBPUT entry points.) Any item that is not specifically changed remains as it was in the original (deleted) record.
4. Store the newly re-created record in the database while the database is in transaction state.

The re-create operation allows the program to access the user work area in which a previously deleted record image resides and to reuse the record. Re-creating can be done either in or out of transaction state. However, since a record can be deleted and stored only in transaction state, re-creating records in transaction state is more efficient.

### Caution

When you delete a record, an image of the record format and values remains in the user work area. The re-create operation uses this image to re-create the record. If your program performs any operation that affects the user work area after the deletion operation and before the re-create operation, you will not be able to re-create the record.

### Passing Parameters

The standard entry point used to re-create a record has two required parameters: the data set qualified name and the record type.

Parameter	Explanation
<data set qualified name>	Identifies the data set in which the record resides.
<record type>	Determines whether the duplicated record has a variable or a fixed format. (The <i>DMSII Technical Overview</i> has a detailed explanation of variable and fixed-format records.)



## Re-creating Records (cont.)

Table 3-17 gives the exported names of the entry points, along with the parameters and the data type of each parameter.

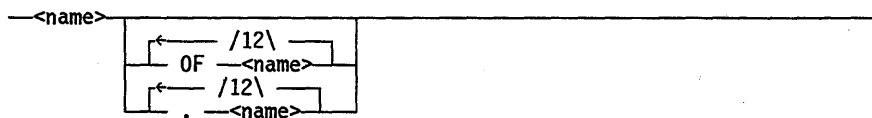
**Table 3-17. Re-creating a Record**

Exported Name of Entry Point	Parameter	Data Type of Parameter
ALGOLRECREATE	<data set qualified name>	STRING
	<record type>	STRING
DBRECREATE	<data set qualified name>	DISPLAY
	<record type>	DISPLAY
FORTRAN77RECREATE	<data set qualified name>	CHARACTER
	<record type>	CHARACTER

### Data Set Name

The data set name must be unique. The following syntax diagram shows you can use up to 12 qualifying parameters to uniquely identify the data set:

<data set qualified name>



The variable element of the data set name is explained as follows:

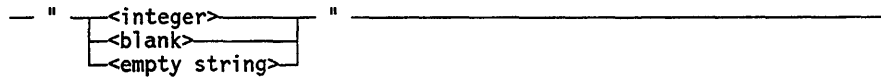
Element	Explanation
<data set qualified name>	Identifies the data set in which the existing record resides. The name can consist of from 1 to 17 letters, digits, and—depending on the language—either underscores ( _ ) or hyphens (-). The first character must be a letter. The last character must be either a letter or a digit.

For example, if your program accesses two databases (DB1 and DB2) and each database has a data set named CLASS, you must qualify the data set name by naming the database.

**Record Type**

Use the following syntax to identify the record type:

<record type>



The variable elements of the record type parameter are explained as follows:

Element	Explanation
<integer>	Creates a variable-format record. An integer is a whole number (0123456789) that consists of from 1 to 12 digits and whose maximum value can be 549755813887.  A variable format consists of two parts: a fixed part (header) and a variable part (trailer): <ul style="list-style-type: none"> <li>• Entry of a nonzero integer creates both parts of a variable-format record.</li> <li>• Entry of a zero (0) creates only the fixed part of a variable-format record.</li> </ul>
blank	Creates a fixed-format record.
empty string	Creates a fixed-format record.

**Programming Examples**

The following program fragments re-create a fixed format record in the data set EMP.

## Re-creating Records (cont.)

---

### ALGOL Program Fragment

%Declaring local variables

```
STRING
  DATA_SET_NAME, SPACE_1;
BOOLEAN
  RSLT;
```

%Assigning values to parameters

```
DATA_SET_NAME:="EMP";
SPACE_1:=" ";
```

%Declaring library entry point

```
LIBRARY DMI;
BOOLEAN PROCEDURE DBRECREATE (STRUCTURE,RECORDTYPE);
  STRING STRUCTURE, RECORDTYPE;
  LIBRARY DMI (ACTUALNAME="ALGOLRECREATE");
```

%Invoking entry point

```
RSLT:=DBRECREATE(DATA_SET_NAME,SPACE_1);
IF RSLT
  THEN EXCEPTIONHANDLER;
```

### COBOL74 or COBOL85 Program Fragment

*Note: In the following example, the variable RESULT is declared as a COMP item, making this example a COBOL74 or COBOL85 program fragment. If RESULT were declared as a COMP-2 item, this fragment could be used in a COBOL program.*

\*\*\*Declaring variables

```
Ø1 DATA-SET-NAME          PIC X(17) VALUE IS "EMP".
Ø1 SPACE-1                 PIC X(1) VALUE IS " ".
Ø1 RESULT                  PIC 9(1) COMP VALUE IS Ø.
```

\*\*\*Invoking entry point

```
CALL "DBRECREATE OF DMINTERPRETER"
  USING DATA-SET-NAME, SPACE-1
  GIVING RESULT.
```

**FORTRAN77 Program Fragment**

```
***Declaring library entry point
<<icom>
LOGICAL FUNCTION DBRCR (STR, RECTYP)
  CHARACTER STR, RECTYP
  IN LIBRARY DMI (ACTUALNAME = "FORTRAN77RECREATE")
END

***Setting variables to initial values

CHARACTER DSNAME      *4      /"EMP"/,
  *      SPACE        *1      /" "/

***Declaring local variables

LOGICAL  RSLT

***Invoking entry point

RSLT = DBRCR(DSNAME,SPACE)
IF (RSLT) CALL EXCEPT
```

## Storing Records

You must use a store operation to enter a newly created, re-created, or modified record into a specified structure. The store operation also makes the appropriate entries into each set associated with the data set.

Only records that have been placed into the work area of the program by a lock, create, re-create, or store operation can be stored.

*Note:* You can store a record in an audited database only when the program is in transaction state.

### Passing a Parameter

The standard entry point used to store a record has one required parameter – the name of the data set or database.

Parameter	Explanation
<structure>	Identifies either the data set or the database where the record is to be stored.

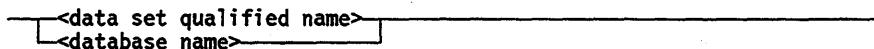
Table 3-18 gives the exported names of the entry points, along with the parameter and the data type of the parameter.

Table 3-18. Storing Records

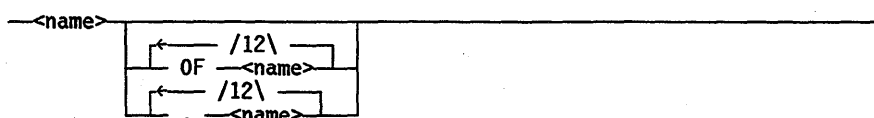
Exported Name of Entry Point	Parameter	Data Type of Parameter
ALGOLSTORE	<structure>	STRING
DBSTORE	<structure>	DISPLAY
FORTTRAN77STORE	<structure>	CHARACTER

The structure can be either a data set or a database. The following syntax diagrams show if the name of the data set is not unique, you can use up to 12 qualifying parameters to uniquely identify the data set. A database name cannot be qualified.

<structure>



<data set qualified name>



The variable element of the structure parameter is explained as follows:

Element	Explanation
<data set qualified name>	Identifies the data set in which the record will be stored. The data set must be defined as a data set within the specified database.
<database name>	Identifies the database for which the DMINTERPRETER library was compiled. The stored information is stored in the global record for the database.
<name>	Identifies the name of the data set or database. A name can consist of from 1 to 17 letters, digits, and—depending on the language—either underscores (_) or hyphens (-). The first character must be a letter. The last character must be either a letter or a digit.

For example, if your program accesses two databases (DB1 and DB2) and each database has a data set named CLASS, you must qualify the data set name by naming the database.

### Programming Examples

The following program fragments show how to store a record in a data set named EMP.

#### ALGOL Program Fragment

```

%Declaring local variables

STRING
  DATA_SET_NAME;
BOOLEAN
  RSLT;

%Assigning values to parameters

DATA_SET_NAME:="EMP";

%Declaring library entry point

LIBRARY DMI;
BOOLEAN PROCEDURE DBSTORE (STRUCTURE);
  STRING STRUCTURE;
  LIBRARY DMI (ACTUALNAME="ALGOLSTORE");

%Invoking entry point

RSLT:=DBSTORE(DATA_SET_NAME);
IF RSLT
  THEN EXCEPTIONHANDLER;

```

## Storing Records (cont.)

---

### COBOL74 or COBOL85 Program Fragment

*Note: In the following example, the variable RESULT is declared as a COMP item, making this example a COBOL74 or COBOL85 program fragment. If RESULT were declared as a COMP-2 item, this fragment could be used in a COBOL program.*

\*\*\*Declaring variables

```
Ø1 DATA-SET-NAME      PIC X(17) VALUE IS "EMP".
Ø1 RESULT               PIC 9(1) COMP VALUE IS Ø.
```

\*\*\*Invoking entry point

```
CALL "DBSTORE OF DMINTERPRETER"
      USING DATA-SET-NAME
      GIVING RESULT.
```

### FORTRAN77 Program Fragment

\*\*\*Declaring library entry point

```
LOGICAL FUNCTION DBSTORE (STR)
  CHARACTER STR
  IN LIBRARY DMI (ACTUALNAME = "FORTRAN77STORE")
END
```

\*\*\*Setting variables to initial values

```
CHARACTER DSNAME      *4      /"EMP"/
```

\*\*\*Declaring local variables

```
LOGICAL RSLT
```

\*\*\*Invoking entry point

```
RSLT = DBSTORE(DSNAME)
IF (RSLT) CALL EXCEPT
```

## End Transaction – Leaving Transaction State

When transactions are completed, the end transaction operation takes the audited database out of transaction state and placed the database a protected state. In the protected state, no update activity can be performed.

*Note: DMSII ignores invocations to begin or end transaction state for unaudited databases.*

### Passing a Parameter

The standard entry point used to leave transaction state has one required parameter – the audit syncpoint parameter.

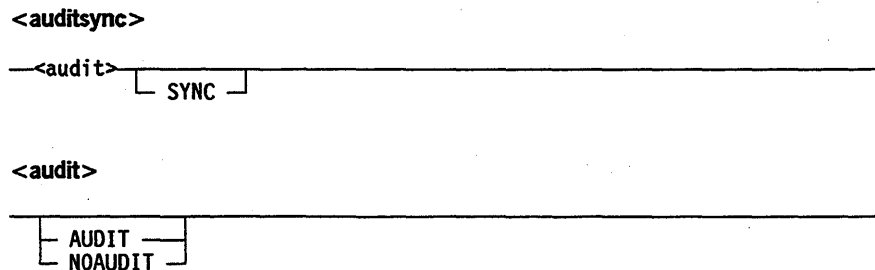
Parameter	Explanation
<auditsync>	Specifies whether an image of the current restart data set is to be audited and can force a syncpoint.  The restart data set is used for restart routines. A syncpoint minimizes application recovery time by controlling how often the database is forced to be synchronized with the audit trail. (Consult the <i>DMSII Technical Overview</i> , the <i>DMSII DASDL Reference Manual</i> , and the <i>DMSII Application Programming Guide</i> for details about restart data sets and syncpoints.)

Table 3-19 gives the exported names of the entry points, along with the parameter and the data type of the parameter.

Table 3–19. End Transaction – Leaving Transaction State

Exported Name of Entry Point	Parameter	Data Type of Parameter
ALGOLENDTRANSACTION	<auditsync>	STRING
DBENDTRANSACTION	<auditsync>	DISPLAY
FORTRAN77ENDTRANSACTION	<auditsync>	CHARACTER

Use the following syntax to end transaction state:





## End Transaction — Leaving Transaction State (cont.)

---

The elements of the audit syncpoint parameter are explained as follows:

Element	Explanation
empty or blanks	(Default) NOAUDIT assumed.
AUDIT	Audits an image of the current restart data set record.
NOAUDIT	No auditing occurs.
SYNC	Forces a syncpoint after the transaction is successfully completed.

Use the SYNC option in an end transaction operation to override the syncpoint option set in the DASDL description of the database. The end transaction operation should be immediately followed by recovery code.

You can use the SYNC option with either the AUDIT or NOAUDIT option; both AUDIT SYNC and NOAUDIT SYNC are valid.

### Programming Examples

The following program fragments cause a program to exit transaction state with the current restart data set record being audited. In these instances, your program must set a variable to AUDIT or AUDIT SYNC before invoking the entry point.

#### ALGOL Program Fragment

```
%Declaring local variables

STRING
  P_AUDIT;
BOOLEAN
  RSLT;

%Assigning values to parameters

P_AUDIT:="AUDIT";

%Declaring library entry point

LIBRARY DMI;
BOOLEAN PROCEDURE DBENDTRANSACTION (AUDITSYNC);
STRING AUDITSYNC;
LIBRARY DMI (ACTUALNAME="ALGOLENDTRANSACTION");

%Invoking entry point

RSLT:=DBENDTRANSACTION(P_AUDIT);
IF RSLT
  THEN EXCEPTIONHANDLER;
```

**COBOL74 or COBOL85 Program Fragment**

*Note: In the following example, the variable RESULT is declared as a COMP item, making this example a COBOL74 or COBOL85 program fragment. If RESULT were declared as a COMP-2 item, this fragment could be used in a COBOL program.*

\*\*\*Declaring variables

```
Ø1 P-AUDIT      PIC X(5) VALUE IS "AUDIT".  
Ø1 RESULT      PIC 9(1) COMP VALUE IS Ø.
```

\*\*\*Invoking entry point

```
CALL "DBENDTRANSACTION OF DMINTERPRETER"  
  USING P-AUDIT  
  GIVING RESULT.
```

**FORTRAN77 Program Fragment**

\*\*\*Declaring library entry point

```
LOGICAL FUNCTION DBETR (AUDIT)  
  CHARACTER AUDIT  
  IN LIBRARY DMI (ACTUALNAME = "FORTRAN77ENDTRANSACTION")  
END
```

\*\*\*Setting variables to initial values

```
CHARACTER AUDIT *1Ø /"AUDIT"/
```

\*\*\*Declaring local variables

```
LOGICAL RSLT
```

\*\*\*Invoking entry point

```
RSLT = DBETR(AUDIT)  
IF (RSLT) CALL EXCEPT
```

# Closing a DMSII Database

After closing a DMSII database, your application program can no longer access any of the files associated with the database. Although the database is implicitly closed when the library execution is complete, explicitly closing the database is a recovery safeguard.

*Note: Do not close a database when the database is in transaction state. The transaction state should be explicitly exited before the database is closed. See "End Transaction—Leaving Transaction State" in this section.*

### Passing Parameters

The standard entry point that explicitly closes the database has no required parameters. Table 3-20 gives the exported names of the entry points.

Table 3-20. Export Names of Entry Points Used to Close a Database

Export Names
ALGOLCLOSE
DBCLOSE
FORTAN77CLOSE

### Programming Examples

The following program fragments close a database.

#### ALGOL Program Fragment

*Note: You do not need to assign ALGOLCLOSE values to parameters in order to use the ALGOLCLOSE entry point.*

```
%Declaring local variables

BOOLEAN
  RSLT;

%Declaring library entry point

LIBRARY DMI;
BOOLEAN PROCEDURE DBCLOSE;
  LIBRARY DMI (ACTUALNAME="ALGOLCLOSE");

%Invoking entry point

RSLT:=DBCLOSE;
IF RSLT
  THEN EXCEPTIONHANDLER;
```

**COBOL74 or COBOL85 Program Fragment**

*Note: In the following example, the variable RESULT is declared as a COMP item, making this example a COBOL74 or COBOL85 program fragment. If RESULT were declared as a COMP-2 item, this fragment could be used in a COBOL program.*

\*\*\*Declaring variables

```
Ø1 RESULT      PIC 9(1) COMP VALUE IS Ø.
```

\*\*\*Invoking entry point

```
CALL "DBCLOSE OF DMINTERPRETER"  
      GIVING RESULT.
```

**FORTRAN77 Program Fragment**

*Note: You do not need to set FORTRAN77CLOSE variables to initial values in order to use the FORTRAN77CLOSE entry point.*

\*\*\*Declaring library entry point

```
LOGICAL FUNCTION DBCLOSE()  
      IN LIBRARY DMI (ACTUALNAME = "FORTRAN77CLOSE")  
END
```

\*\*\*Declaring local variables

```
LOGICAL  RSLT
```

\*\*\*Invoking entry point

```
RSLT = DBCLOSE( )  
IF (RSLT) CALL EXCEPT
```

## Executing Language Extensions

You can execute language extensions through a standard entry point. The entry point allows flexible, run-time definition of DMSII operations and provides users with an easy transition from DMSII language extensions. However, because the input to this entry point must be completely parsed, the entry point runs at higher processor cost.

### Passing a Parameter

As seen in Table 3-21, there is one required parameter – the language extension statement. This parameter must evaluate to a complete statement. Exception actions and input and output mapping are not allowed.

*Note: FORTRAN77 does not support the DMSII language extensions interface. However, you can use the FORTRAN77VERB entry point.*

Table 3-21. Executing Language Extensions

Exported Name of Entry Point	Parameter
ALGOLVERB	<language extension statement>
DBVERB	<language extension statement>
FORTAN77VERB	<language extension statement>

The following DMSII user language constructs can be used as the required parameter:

- Database statement
- Create statement
- Transaction statement
- Record statement
- Set statement
- Search statement
- Status statement

Brief descriptions of these statements are included here. If your application language supports the DMSII language extensions, consult the appropriate language-specific reference manual for more information.

### Database Statement

A database statement performs one of two functions:

- Closes the database
- Opens the database for either inquiry or inquiry and update

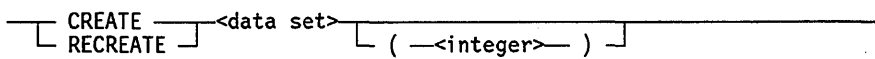
<database statement>



**Create Statement**

A create statement initializes the user work area for a new or re-created image of a data set record.

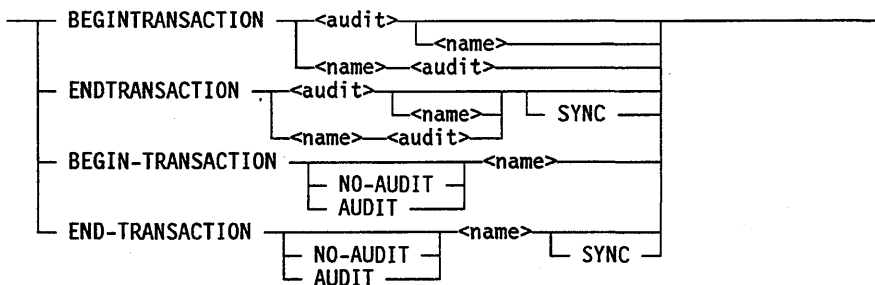
<create statement>



**Transaction Statement**

A transaction statement causes a program to either enter or exit transaction state, with or without auditing an image of the current restart data set and forcing a syncpoint. If a name is specified, it is the restart data set.

<transaction statement>

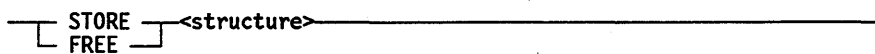


**Record Statement**

A record statement performs one of two functions:

- Stores a new or modified record in a specified structure
- Unlocks a record residing in a specified structure

<record statement>



**Set Statement**

A set statement sets the current path of a specified structure to the start or end of the structure or, if it is an index set, to a particular data set.

## Executing Language Extensions (cont.)

---

<set statement>

— SET —<structure> TO —<target>—————|

### Search Statement

A search statement finds a selected record in a specified structure and locks the record for modification or deletion.

<search statement>

— FIND —| —<structure> —| —<expression> —|  
— LOCK —| — FIRST —| — WHERE —|  
— MODIFY —| — LAST —| — AT —|  
— DELETE —| — NEXT —|  
— —| — PRIOR —|  
—<data set> — VIA —<link>—————|

### Status Statement

A status statement allows the user to find out whether a prior database operation was successful.

<status statement>

— DMSTATUS —————|

### Notes:

- *The COBOL, COBOL74, and COBOL85 languages cannot use the DMSTATUS form of a call on DBVERB. These languages cannot pass an exception parameter. Instead, use a direct call on the DBSTATUS entry point.*
- *The ALGOL and FORTRAN77 languages, when using the status statement construct, pass only the string DMSTATUS as a parameter. In all other cases, ALGOL programs use complete user language interface statements.*

### Programming Examples

The following program fragments show how the entry point can be called to execute the set, search, and status language extensions.

#### ALGOL Program Fragment

*Note: You do not need to declare local variables or assign values to parameters that pertain to the DBVERB when you use the DBVERB to execute set, search status language extensions in ALGOL.*

```
%Declaring library entry point

LIBRARY DMI;
BOOLEAN PROCEDURE DBVERB(HOSTSTATEMENT);
STRING HOSTSTATEMENT;
LIBRARY DMI (ACTUALNAME="ALGOLVERB");

%Invoking entry point
%
% USING DBVERB ON A SET STATEMENT
%
RSLT:=DBVERB("SET EMP TO BEGINNING");
IF RSLT
    THEN EXCEPTIONHANDLER;
%
% USING DBVERB ON A SEARCH STATEMENT
%
RSLT:=DBVERB("FIND FIRST EMP AT EMP-NO");
IF RSLT
    THEN EXCEPTIONHANDLER;
%
% USING DBVERB ON A STATUS STATEMENT
%
DBVERB("FIND PRIOR EMP");
RSLT:=DBVERB("DMSTATUS");
```



## Executing Language Extensions (cont.)

---

### COBOL, COBOL74, or COBOL85 Program Fragment

\*\*\*Declaring variables

```
Ø1 DATA-REQUEST          PIC X(4Ø) VALUE IS SPACES.
```

\*\*\*Invoking entry point

\*\*

\*\* USING DBVERB ON A SET STATEMENT

\*\*

```
MOVE "SET EMP TO BEGINNING" TO DATA-REQUEST.
```

```
CALL "DBVERB OF DMINTERPRETER"
```

```
    USING DATA-REQUEST
```

```
    GIVING RESULT.
```

```
IF RESULT = 1
```

```
    PERFORM EXCEPTION-HANDLER.
```

\*\*

\*\* USING DBVERB ON A SEARCH STATEMENT

\*\*

```
MOVE "FIND FIRST EMP AT EMP-NO=11" TO DATA-REQUEST.
```

```
CALL "DBVERB OF DMINTERPRETER"
```

```
    USING DATA-REQUEST
```

```
    GIVING RESULT.
```

```
IF RESULT = 1
```

```
    PERFORM EXCEPTION-HANDLER.
```

**FORTRAN77 Program Fragment**

*Note: You do not have to set any variables that pertain to the DBVERB to initial values when you use the DBVERB to execute set, search status language extensions in FORTRAN77.*

\*\*\*Declaring library entry point

```
LOGICAL FUNCTION DBVERB (STMT)
  CHARACTER STMT
  IN LIBRARY DMI (ACTUALNAME = "FORTRAN77VERB")
END
```

\*\*\*Declaring local variables

```
LOGICAL  RSLT
```

\*\*\*Invoking entry point

```
      *      USING DBVERB ON A SET STATEMENT
```

```
RSLT = DBVERB("SET EMP TO BEGINNING")
IF (RSLT) CALL EXCEPT
```

```
      *      USING DBVERB ON A SEARCH STATEMENT
```

```
RSLT = DBVERB("FIND FIRST EMP AT EMP-NO=11")
IF (RSLT) CALL EXCEPT
```

```
      *      USING DBVERB ON A STATUS STATEMENT
```

```
RSLT = DBVERB("SET EMP TO BEGINNING")
IF (RSLT) CALL EXCEPT
RSLT = DBVERB("FIND PRIOR EMP")
RSLT = DBVERB("DMSTATUS")
CALL DBEXCN(RSLT,MSG)
CALL DBEXCT(RSLT,MSG)
```



# Section 4

## Transferring Data

With the DMSII language extensions interface, any program written in a supported COBOL language can directly access data in the database. All other languages must use input and output mapping statements. Direct access is available because the COBOL compilers can access the DASDL database description during program compilation and can enforce data-type checking. In contrast, any compiler in which formal data structures are not directly supported cannot enforce data-type checking.

With the interpretive interface, direct access to data is not possible through any language compiler for these reasons:

- The compiler might not be able to enforce data-type checking in all situations.
- The format of a user work area might not be known during program compilation.

Instead of accessing the data directly, the interpretive interface uses data transfer entry points, provided through the DMINTERPRETER library, to access and move data. The DBGET and DBPUT data transfer entry points move data into and out of a user work area.

Each data transfer task, with its corresponding entry points, are listed in Table 4-1. Use the heading for each listed data transfer task to locate the pages that detail the entry point. For example, a description of the data transfer entry points ALGOLGETSTRING, DBGETDISPLAY, and FORTRAN77GETCHARACTER is found under, "Moving Character Strings to Variables." The tasks are grouped; all DBGET data entry points are presented before all DBPUT data entry points. Constructing data transfers during program execution is explained as the last data entry point. General guidelines for using the data entry points precede the task descriptions.

**Table 4-1. Data Entry Points to the DMINTERPRETER Library by Task**

<b>Task</b>	<b>Entry Points</b>
<b>Moving character strings to variables</b>	ALGOLGETSTRING
	DBGETDISPLAY
	FORTRAN77GETCHARACTER
<b>Moving Kanji alpha character strings to variables</b>	ALGOLGETKANJI
	DBGETKANJI
	FORTRAN77GETKANJI

continued

## Transferring Data

---

**Table 4-1. Data Entry Points to the DMINTERPRETER Library by Task (cont.)**

<b>Task</b>	<b>Entry Points</b>
<b>Moving numeric values to variables</b>	ALGOLGETREAL DBGETREAL FORTRAN77GETREAL
<b>Moving double-precision values to variables</b>	ALGOLGETDOUBLE DBGETDOUBLE FORTRAN77GETDOUBLE
<b>Retrieving Boolean values</b>	ALGOLGETBOOLEAN DBGETBOOLEAN FORTRAN77GETLOGICAL
<b>Placing strings into data items</b>	ALGOLPUTSTRING DBPUTDISPLAY FORTRAN77PUTCHARACTER
<b>Placing strings into Kanji alpha items</b>	ALGOLPUTKANJI DBPUTKANJI FORTRAN77PUTKANJI
<b>Placing numeric values into data items</b>	ALGOLPUTREAL DBPUTREAL FORTRAN77PUTCHARACTER
<b>Placing double-precision values into data items</b>	ALGOLPUTDOUBLE DBPUTDOUBLE FORTRAN77PUTDOUBLE
<b>Setting data items to Boolean values</b>	ALGOLPUTBOOLEAN DBPUTBOOLEAN FORTRAN77PUTLOGICAL
<b>Setting data items to a null value</b>	ALGOLPUTNUL DBPUTNULL FORTRAN77PUTNULL
<b>Constructing data transfers during program execution</b>	ALGOLDATA DBDATA FORTRAN77DATA

The discussion of each task includes the following:

- A brief explanation of the function of the entry point
- Syntax diagrams and semantics that describe required parameters
- Program fragments that illustrate
  - Declaring variables for the entry points
  - Invoking the entry point
  - Returning results from the DMINTERPRETER library

Unless specifically noted, COBOL, COBOL74, and COBOL85 have the same requirements.

## Guidelines for Using Data Transfer Entry Points

DBGET and DBPUT data transfer entry points operate on data in the following manner:

- DBGET entry points move data from the user's work area to user-declared variables. The DBGET entry points return an exception if the declared variable is too short to hold the specified data item.
- DBPUT entry points move data from user-declared variables to user work areas. The DBPUT entry points return an exception if the value to be put into the database does not fit into the specified data item.

As illustrated in Figure 4-1, modifying the database with the interpretive interface resembles the following three-step process:

1. Prepare the user work area in one of two ways:
  - Use a find, delete, lock or secure operation to retrieve and move data from an existing record to the user work area.
  - Use a create or re-create operation to reserve space in the user work area.
2. Use the DBPUT and DBGET data transfer entry points to transfer values between the program and the work area.
3. Use the store operation to transfer data from the user work area back to the database.

## Transferring Data

---

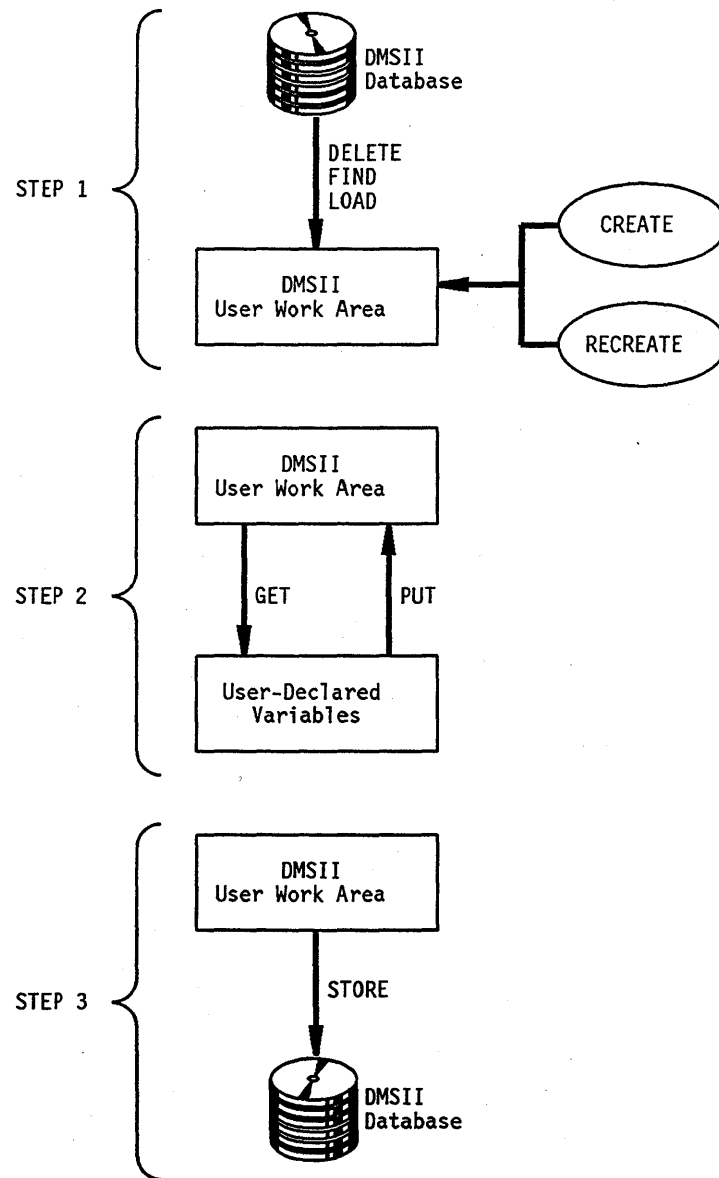


Figure 4-1. Three Steps in Transferring Data

The DMINTERPRETER library checks the parameters of data transfer entry points to ensure that the requested operation is compatible with the DASDL database description. For example, items declared in the DASDL description with an OCCURS clause must be subscripted.

For more information read the following sections:

- Section 2, “Accessing the Interpretive Interface,” details how each language accesses the interpretive interface and how each language invokes an entry point.
- Section 3, “Manipulating the Database,” explains how to find, lock, create, re-create, and store records.
- Section 5, “Handling Exceptions,” contains detailed information on exception handling.
- Appendix B, “DMSII Exceptions and Errors,” contains the specific messages for the DBGET and DBPUT exceptions.



# Moving Character Strings to Variables

You can move the content of an alpha or a group item character string to a program-declared variable, using a DBGET data transfer entry point.

### Passing Parameters

The data transfer entry points that move character strings to variables have three required parameters: the name of the structure, the name of the item, and the name of the variable.

Parameter	Explanation
<structure>	Identifies either the data set (for an alpha item) or the database (for a global item) in which the item is located.
<item name>	Identifies the data set item whose string value is to be moved to the user-declared variable.
<variable name>	Identifies the user-declared variable by its name in the program. The variable must be able to contain a string.

Table 4-2 gives the exported names of the entry points, as well as the parameters and the data type of each parameter.

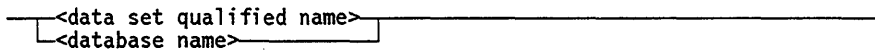
Table 4-2. Moving Character Strings to Variables

Exported Name of Entry Point	Parameter	Data Type of Parameter
ALGOLGETSTRING	<structure>	STRING
	<item name>	STRING
	<variable name>	STRING
DBGETDISPLAY	<structure>	DISPLAY
	<item name>	DISPLAY
	<variable name>	DISPLAY
FORTRAN77GETCHARACTER	<structure>	CHARACTER
	<item name>	CHARACTER
	<variable name>	CHARACTER

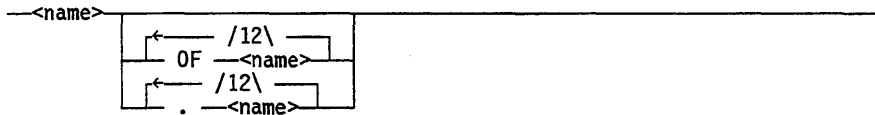
### Structure

The structure can be either a data set or a database. The following syntax diagrams show that if the name of the data set is not unique, you can use up to 12 qualifying parameters to uniquely identify the data set. A database name cannot be qualified.

**<structure>**



**<data set qualified name>**



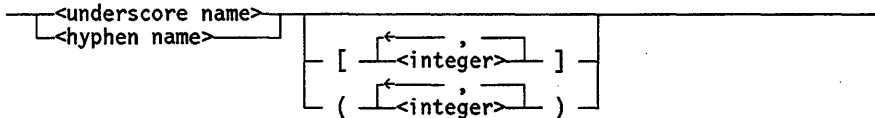
The variable elements of the structure parameter are explained as follows:

Element	Explanation
<data set qualified name>	Identifies the data set in which the alpha item is located.
<database name>	Identifies the database in which the global data item is located.
<name>	Identifies the name of the data set or database. A name can consist of from 1 to 17 letters, digits, and—depending on the language—either underscores ( ) or hyphens (-). The first character must be a letter. The last character must be either a letter or a digit.

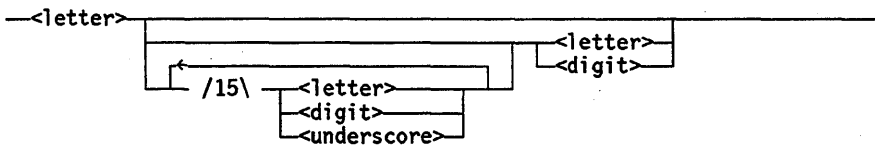
**Item Name**

The following syntax diagrams show the valid combinations of characters that can be used to form an item name:

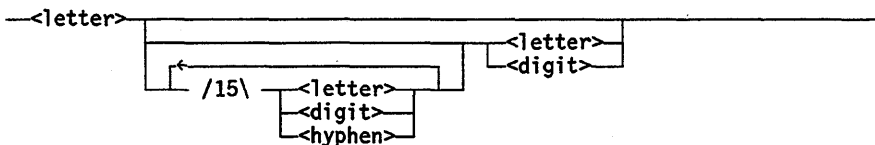
**<item name>**



**<underscore name>**



**<hyphen name>**



## Moving Character Strings to Variables (cont.)

---

The variable elements of an item name are explained as follows:

Element	Explanation
<underscore name>	A string of from 1 to 17 letters, digits, and underscores (_), starting with a letter and ending with either a letter or a digit.
<hyphen name>	A string of from 1 to 17 letters, digits, and hyphens (-), starting with a letter and ending with either a letter or a digit.
<integer>	A subscript for an OCCURS clause item. An integer is a whole number (0123456789) that consists of from 1 to 12 digits and that has a maximum value of 549755813887.

### Variable Name

Each language has a different way of identifying the variable that contains or evaluates to the string. Consult the appropriate language reference manual for more information on variables that can contain a string.

### Program Examples

The following program fragments use data transfer entry points to move character strings from the user work area to variables declared in the application program.

#### ALGOL Program Fragment

In the ALGOL example, the data set name (the structure) is initialized as EMP at the beginning of the program. The item name is set to EMP\_FNAME before the entry point is invoked. The variable is set to EXPR\_3 in body of the program.

```

%Declaring local variables

STRING
  DATA_SET_NAME, ITEM_NAME, EXPR_3;
BOOLEAN
  RSLT;

%Assigning values to parameters

DATA_SET_NAME:="EMP";

%Declaring library entry point

LIBRARY DMI;
BOOLEAN PROCEDURE DBGETSTRING(STRUCTURE,ITEM,VARIABLE);
  STRING STRUCTURE, ITEM;
  REAL VARIABLE;
  LIBRARY DMI (ACTUALNAME="ALGOLGETSTRING");

%Invoking entry point

ITEM_NAME:="EMP_FNAME";
RSLT:=DBGETSTRING(DATA_SET_NAME,ITEM_NAME,EXPR_3);
IF RSLT
  THEN EXCEPTIONHANDLER;

```

**COBOL74 or COBOL85 Program Fragment**

In the following example, the structure is declared as EMP at the beginning of the program. The item name is set to EMP-FNAME before the entry point is invoked. The variable is set to VAR-3 in body of the program.

*Note: In this example, the variable RESULT is declared as a COMP item, making this example a COBOL74 or COBOL85 program fragment. If RESULT were declared as a COMP-2 item, this fragment could be used in a COBOL program.*

```

***Declaring variables

Ø1 DATA-SET-NAME          PIC X(17) VALUE IS "EMP".
Ø1 ITEM-NAME              PIC X(17) VALUE IS SPACES.
Ø1 VAR-3                  PIC X(1Ø) VALUE IS SPACES.
Ø1 RESULT                 PIC 9(1) COMP VALUE IS Ø.

***Invoking entry point

MOVE "EMP-FNAME" TO ITEM-NAME.
CALL "DBGETDISPLAY OF DMINTERPRETER"
  USING DATA-SET-NAME, ITEM-NAME, VAR-3
  GIVING RESULT.

```

## Moving Character Strings to Variables (cont.)

---

### FORTRAN77 Program Fragment

In this FORTRAN77 example, the structure is set to EMP through the variable DSNAME. The item name is assigned before the entry point is invoked. The variable is set to EXPR3 in the body of the program.

\*\*\*Declaring library entry point

```
LOGICAL FUNCTION DBGETC (STR, ITEM, VAR)
  CHARACTER STR, ITEM, VAR
  IN LIBRARY DMI (ACTUALNAME = "FORTRAN77GETCHARACTER")
END
```

\*\*\*Setting variables to initial values

```
CHARACTER DSNAME      *4      /"EMP"/,
*      ITMNAM         *13     ,
*      EXPR3          *22
```

\*\*\*Declaring local variables

```
LOGICAL  RSLT
```

\*\*\*Invoking entry point

```
ITMNAM = "EMP-FNAME"
RSLT = DBGETC(DSNAME,ITMNAM,EXPR3)
IF (RSLT) CALL EXCEPT
```

## Moving Kanji Alpha Character Strings to Variables

You can move the content of a Kanji alpha character string to a program-declared variable, using a GETKANJI data transfer entry point.

### Passing Parameters

The GETKANJI entry points have three required parameters: the name of the structure, the name of the item, and the name of the variable.

Parameter	Explanation
<structure>	Identifies the data set or the database in which the item is located.
<item name>	Identifies the data set item whose Kanji string value is to be moved to the user-declared variable.
<variable name>	Identifies the user-declared variable by its name in the program. The variable must be able to contain a Kanji string.

Table 4-3 gives the exported names of the entry points, as well as the parameters and the data type of each parameter.

Table 4-3. Moving Kanji Character Strings to Variables

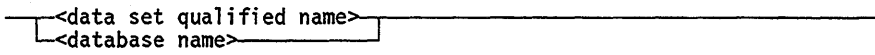
Exported Name of Entry Point	Parameter	Data Type of Parameter
ALGOLGETKANJI	<structure>	STRING
	<item name>	STRING
	<variable name>	STRING
DBGETKANJI	<structure>	DISPLAY
	<item name>	DISPLAY
	<variable name>	DISPLAY or KANJI
FORTRAN77GETKANJI	<structure>	CHARACTER
	<item name>	CHARACTER
	<variable name>	CHARACTER

### Structure

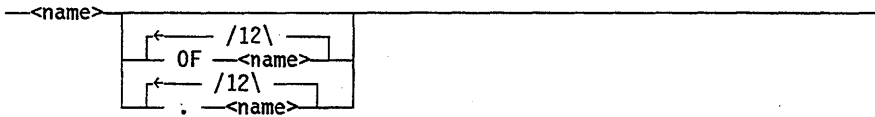
The structure can be either a data set or a database. The following syntax diagrams show that if the name of the data set is not unique, you can use up to 12 qualifying parameters to uniquely identify the data set. A database name cannot be qualified.

## Moving Kanji Alpha Character Strings to Variables (cont.)

**<structure>**



**<data set qualified name>**



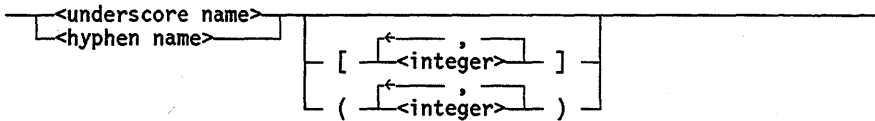
The variable elements of the structure parameter are explained as follows:

Element	Explanation
<data set qualified name>	Identifies the data set that is to be accessed at run time.
<database name>	Identifies the database for which the DMINTERPRETER library was compiled.
<name>	Identifies the name of the data set or database. A name can consist of from 1 to 17 letters, digits, and—depending on the language—either underscores ( ) or hyphens (-). The first character must be a letter. The last character must be either a letter or a digit.

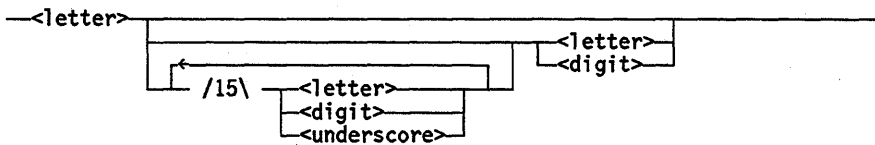
### Item Name

The following syntax diagrams show the valid combinations of characters that can be used to form an item name:

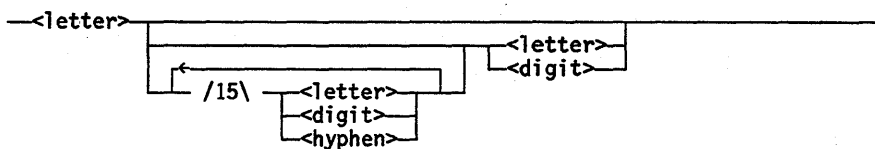
**<item name>**



**<underscore name>**



**<hyphen name>**



The variable elements of a name are explained as follows:

Element	Explanation
<underscore name>	A string of from 1 to 17 letters, digits, and underscores (_), starting with a letter and ending with either a letter or a digit.
<hyphen name>	A string of from 1 to 17 letters, digits, and hyphens (-), starting with a letter and ending with either a letter or a digit.
<integer>	A subscript for an OCCURS clause item. An integer is a whole number (0123456789) that consists of from 1 to 12 digits and that has a maximum value of 549755813887.

### Variable Name

Each language has a different way of identifying the variable that contains or evaluates to a Kanji string. Consult the appropriate language reference manual for more information on variables that can contain a string.

*Note: COBOL does not support Kanji characters. In COBOL74 and COBOL85, where Kanji is supported, the variable can be defined as PIC X(2n) or PIC X(n) USAGE IS KANJI.*

### Program Examples

The following program fragments use GETKANJI data transfer entry points to move Kanji items from the user work area to variables declared in the application program.

#### ALGOL Program Fragment

In the ALGOL example, the structure is EMP, the item name is EMP\_KNAME, and the variable is EXPR\_3.



## Moving Kanji Alpha Character Strings to Variables (cont.)

---

%Declaring local variables

```
STRING
  DATA_SET_NAME, ITEM_NAME, EXPR_3;
BOOLEAN
  RSLT;
```

%Assigning values to parameters

```
DATA_SET_NAME:="EMP";
```

%Declaring library entry point

```
LIBRARY DMI;
BOOLEAN PROCEDURE DBGETKANJI(STRUCTURE,ITEM,VARIABLE);
  STRING STRUCTURE, ITEM;
  REAL VARIABLE;
  LIBRARY DMI (ACTUALNAME="ALGOLGETKANJI");
```

%Invoking entry point

```
ITEM_NAME:="EMP_KNAME";
RSLT:=DBGETKANJI(DATA_SET_NAME,ITEM_NAME,EXPR_3);
IF RSLT
  THEN EXCEPTIONHANDLER;
```

### COBOL74 or COBOL85 Program Fragment

In this example, the structure is EMP, the item name is EMP-KNAME, and the variable is VAR-1. The variable RESULT is declared as a COMP item, making this example a COBOL74 or COBOL85 program fragment.

*Note: COBOL does not support Kanji characters.*

\*\*\*Declaring variables

```
Ø1 DATA-SET-NAME          PIC X(17) VALUE IS "EMP".
Ø1 ITEM-NAME               PIC X(17) VALUE IS SPACES.
Ø1 VAR-1                   PIC X(1Ø) USAGE IS KANJI.
                           VALUE IS SPACES.
Ø1 RESULT                  PIC 9(1) COMP VALUE IS Ø.
```

\*\*\*Invoking entry point

```
MOVE "EMP-KNAME" TO ITEM-NAME.
CALL "DBGETKANJI OF DMINTERPRETER"
  USING DATA-SET-NAME, ITEM-NAME, VAR-1
  GIVING RESULT.
```

**FORTTRAN77 Program Fragment**

In this FORTRAN77 example, the structure is EMP, the item name is EMP-KNAME, and the variable is EXPR3.

\*\*\*Declaring library entry point

```
LOGICAL FUNCTION DBGETK (STR, ITEM, VAR)
  CHARACTER STR, ITEM, VAR
  IN LIBRARY DMI (ACTUALNAME = "FORTRAN77GETKANJI")
END
```

\*\*\*Setting variables to initial values

```
CHARACTER DSNAME      *4      /"EMP"/,
*      ITMNAM         *13     ,
*      EXPR3          *22
```

\*\*\*Declaring local variables

```
LOGICAL  RSLT
```

\*\*\*Invoking entry point

```
ITMNAM = "EMP-KNAME"
RSLT = DBGETK(DSNAME,ITMNAM,EXPR3)
IF (RSLT) CALL EXCEPT
```

# Moving Numeric Values to Variables

You can transfer the numeric value of a number, real, field, count, record-type, or population item to a program-declared variable, using a GETREAL data transfer entry point.

### Passing Parameters

The GETREAL entry points have three required parameters: the name of the structure, the name of the item, and the name of the variable.

Parameter	Explanation
<structure>	Identifies the data set or the database in which the item is located.
<item name>	Identifies the data set item whose numeric value is moved to the user-declared variable.
<variable name>	Identifies the user-declared variable by its name in the program. The variable must be able to contain a real value.

### Notes:

- *If the declared variable is too small to hold the specified data item, the variable is set to HIGH-VALUE (the largest integer that can be represented) and an appropriate exception is returned. Consult the DMSII Technical Overview for an explanation of HIGH-VALUE. Read Appendix B, "DMSII Exceptions and Errors," for more information on exceptions.*
- *If the numeric items that are too large to be transferred by ALGOLGETREAL, DBGETREAL, or FORTRAN77GETREAL, use the ALGOLGETDOUBLE, DBGETREAL, or FORTRAN77GETDOUBLE entry point, respectively.*

Table 4-4 gives the exported names of the entry points, as well as the parameters and the data type of each parameter.

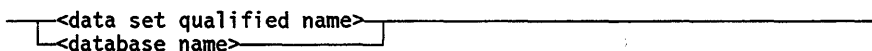
Table 4-4. Moving Numeric Values to Variables

Exported Name of Entry Point	Parameter	Data Type of Parameter
ALGOLGETREAL	<structure>	STRING
	<item name>	STRING
	<variable name>	REAL
DBGETREAL	<structure>	DISPLAY
	<item name>	DISPLAY
	<variable name>	COMP-4 (COBOL)
	<variable name>	REAL (COBOL74 or COBOL85)
FORTRAN77GETREAL	<structure>	CHARACTER
	<item name>	CHARACTER
	<variable name>	REAL

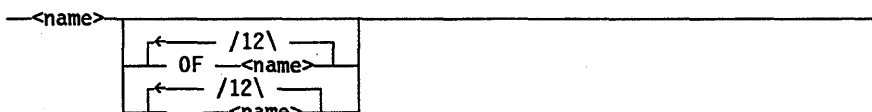
**Structure**

The structure can be either a data set or a database. The following syntax diagrams show that if the name of the data set is not unique, you can use up to 12 qualifying parameters to uniquely identify the data set. A database name cannot be qualified.

**<structure>**



**<data set qualified name>**



The variables elements of the structure parameter are explained as follows:

Element	Explanation
<data set qualified name>	Identifies the data set that is to be accessed at run time.
<database name>	Identifies the database for which the DMINTERPRETER library was compiled.

*continued*

## Moving Numeric Values to Variables (cont.)

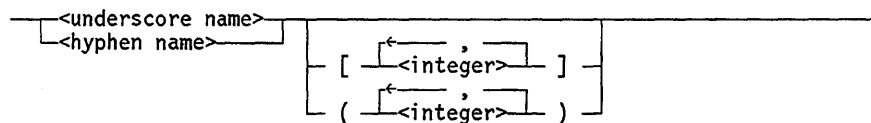
*continued*

Element	Explanation
<name>	Identifies the name of the data set or database. A name can consist of from 1 to 17 letters, digits, and—depending on the language—either underscores (_) or hyphens (-). The first character must be a letter. The last character must be either a letter or a digit.

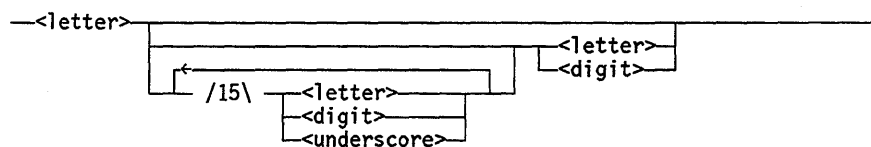
### Item Name

The following syntax diagrams show the valid combinations of characters that can be used to form an item name:

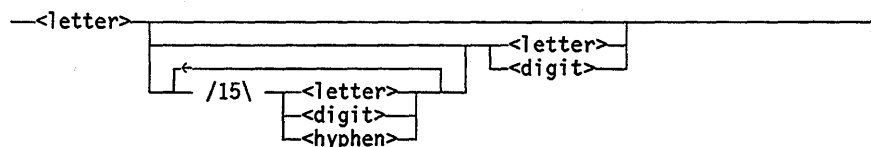
<item name>



<underscore name>



<hyphen name>



The variable elements of an item name are explained as follows:

Element	Explanation
<underscore name>	A string of from 1 to 17 letters, digits, and underscores (_), starting with a letter and ending with either a letter or a digit.
<hyphen name>	A string of from 1 to 17 letters, digits, and hyphens (-), starting with a letter and ending with either a letter or a digit.
<integer>	A subscript for an OCCURS clause item. An integer is a whole number (0123456789) that consists of from 1 to 12 digits and that has a maximum value of 549755813887.

### Variable Name

Each language has a different way of identifying the variable that contains or evaluates to a real numeric value. Consult the appropriate language reference manual for more information on variables that can contain a real numeric value.

*Note: For COBOL, the variable must be declared in the WORKING-STORAGE section of the program as a level 77 COMP-4 item. For COBOL74 and COBOL85, the variable must be declared as a level 77 REAL item.*

### Program Examples

The following program fragments use GETREAL data transfer entry points to move numeric values from the user work area to variables declared in the application program.

#### ALGOL Program Fragment

In the ALGOL example, the data set is EMP, the item is EMP\_NO, and the variable is EXPR\_1.

```
%Declaring local variables

REAL
  EXPR_1;
STRING
  DATA_SET_NAME, ITEM_NAME;
BOOLEAN
  RSLT;

%Assigning values to parameters

DATA_SET_NAME:="EMP";

%Declaring library entry point

LIBRARY DMI;
BOOLEAN PROCEDURE DBGETREAL(STRUCTURE,ITEM,VARIABLE);
  STRING STRUCTURE, ITEM;
  REAL VARIABLE;
  LIBRARY DMI (ACTUALNAME="ALGOLGETREAL");

%Invoking entry point

ITEM_NAME:="EMP_NO";
RSLT:=DBGETREAL(DATA_SET_NAME,ITEM_NAME,EXPR_1);
IF RSLT
  THEN EXCEPTIONHANDLER;
```

#### COBOL74 or COBOL85 Program Fragment

In this example, the data set is EMP, the item is EMP-NO, and the variable is VAR-2. Since VAR-2 is declared as a REAL item and RESULT is declared as a COMP item, this example can be either a COBOL74 or a COBOL85 program fragment.

## Moving Numeric Values to Variables (cont.)

---

*Note: To make this a COBOL program fragment, declare the variable VAR-2 as a COMP-4 item and the variable RESULT as a COMP-2 item.*

\*\*\*Declaring variables

```
Ø1 DATA-SET-NAME      PIC X(17) VALUE IS "EMP".
Ø1 ITEM-NAME           PIC X(17) VALUE IS SPACES.
Ø1 VAR-2               REAL.
Ø1 RESULT              PIC 9(1) COMP VALUE IS Ø.
```

\*\*\*Invoking entry point

```
MOVE "EMP-NO" TO ITEM-NAME.
CALL "DBGETREAL OF DMINTERPRETER"
    USING DATA-SET-NAME, ITEM-NAME, VAR-2
    GIVING RESULT.
```

### **FORTRAN77 Program Fragment**

In this FORTRAN77 example, a value for DSNAME (the structure) is initialized at the beginning of the program; the value of ITMNAM (the item) is assigned in the body of the program. The variable is EXPRI.

\*\*\*Declaring library entry point

```
LOGICAL FUNCTION DBGETR (STR, ITEM, VAR)
  CHARACTER STR, ITEM
  REAL VAR
  IN LIBRARY DMI (ACTUALNAME = "FORTRAN77GETREAL")
END
```

\*\*\*Setting variables to initial values

```
CHARACTER DSNAME      *4    /"EMP"/,
*      ITMNAM         *13
```

```
REAL    EXPRI
```

\*\*\*Declaring local variables

```
LOGICAL  RSLT
```

\*\*\*Invoking entry point

```
ITMNAM = "EMP-NO"
RSLT = DBGETR(DSNAME,ITMNAM,EXPRI)
IF (RSLT) CALL EXCEPT
```

## Moving Double-Precision Values to Variables

Use the GETDOUBLE data transfer entry point to move a numeric value that exceeds the capacity of a real variable from a number, real, field, count, record-type, or population item to a program-declared variable. (See “Moving Numeric Values to Variables” in this section.)

### Passing Parameters

The GETDOUBLE entry points have three required parameters: the name of the structure, the name of the item, and the name of the variable.

Parameter	Explanation
<structure>	Identifies the data set or the database in which the item is located.
<item name>	Identifies the data set item whose double-precision value is to be moved to the user-declared variable.
<variable name>	Identifies the user-declared variable by its name in the program. The variable must be able to contain a double-precision value.

**Note:** *If an exception occurs, the variable is set to HIGH-VALUE (all bits ON), and an appropriate exception is returned. Read Appendix B, “DMSII Exceptions and Errors,” for more information on exceptions. Consult the DMSII Technical Overview for an explanation of HIGH-VALUE.*

Table 4-5 gives the exported names of the entry points, as well as the parameters and the data type of each parameter.

Table 4–5. Moving Double-Precision Values to Variables

Exported Name of Entry Point	Parameter	Data Type of Parameter
ALGOLGETDOUBLE	<structure>	STRING
	<item name>	STRING
	<variable name>	DOUBLE
DBGETDOUBLE	<structure>	DISPLAY
	<item name>	DISPLAY
	<variable name>	COMP-5 (COBOL)
		DOUBLE (COBOL74 or COBOL85)
FORTRAN77GETDOUBLE	<structure>	CHARACTER
	<item name>	CHARACTER
	<variable name>	DOUBLE PRECISION



## Moving Double-Precision Values to Variables (cont.)

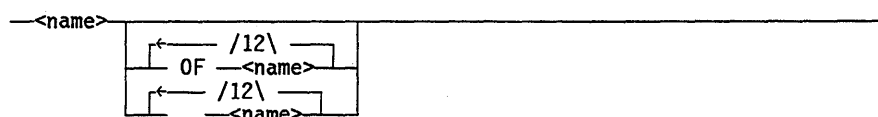
### Structure

The structure can be either a data set or a database. The following syntax diagrams show that if the name of the data set is not unique, you can use up to 12 qualifying parameters to uniquely identify the data set. A database name cannot be qualified.

#### <structure>



#### <data set qualified name>



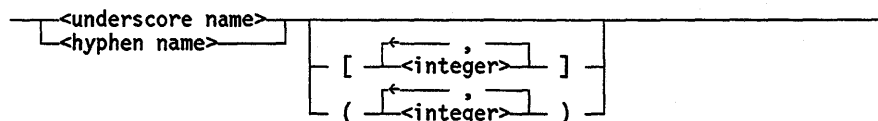
The variable elements of the structure parameter are explained as follows:

Element	Explanation
<data set qualified name>	Identifies the data set that is to be accessed at run time.
<database name>	Identifies the database for which the DMINTERPRETER library was compiled.
<name>	Identifies the name of the data set or database. A name can consist of from 1 to 17 letters, digits, and—depending on the language—either underscores (_) or hyphens (-). The first character must be a letter. The last character must be either a letter or a digit.

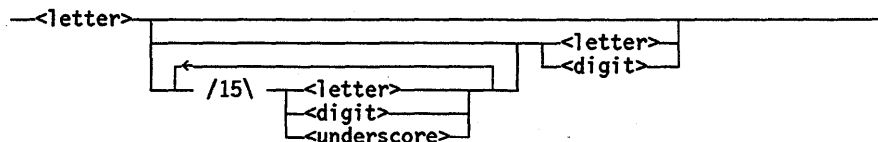
### Item Name

The following syntax diagrams show the valid combinations of characters that can be used to form an item name:

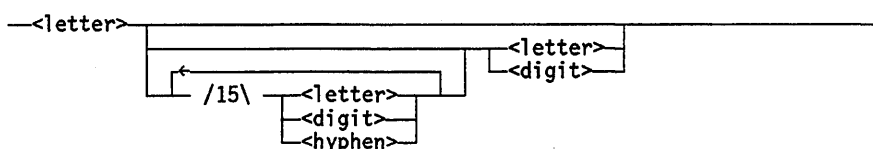
#### <item name>



#### <underscore name>



<hyphen name>



The variable elements of an item name are explained as follows:

Element	Explanation
<underscore name>	A string of from 1 to 17 letters, digits, and underscores ( _ ), starting with a letter and ending with either a letter or a digit.
<hyphen name>	A string of from 1 to 17 letters, digits, and hyphens ( - ), starting with a letter and ending with either a letter or a digit.
<integer>	A subscript for an OCCURS clause item. An integer is a whole number (0123456789) that consists from of 1 to 12 digits and that has a maximum value of 549755813887.

### Variable Name

Each language has a different way of identifying the entity that contains or evaluates to a real, double-precision numeric value. Consult the appropriate language reference manual for more information on variables that can contain a real, double-precision numeric value.

**Note:** For COBOL, the variable must be declared in the *WORKING-STORAGE* section as a level 77 *COMP-5* item. For COBOL74 and COBOL85, the variable must be declared as a level 77 *DOUBLE* item.

### Program Examples

The following program fragments use the DBGETDOUBLE data transfer entry points to move double-precision numeric values from the user work area to variables declared in the application program.

#### ALGOL Program Fragment

In the ALGOL example, the structure is initialized as EMP at the beginning of the program. The item name is EMP\_SSN. The variable is EMPR\_2.

## Moving Double-Precision Values to Variables (cont.)

---

%Declaring local variables

```
DOUBLE
  EXPR_2;
STRING
  DATA_SET_NAME, ITEM_NAME;
BOOLEAN
  RSLT;
```

%Assigning values to parameters

```
DATA_SET_NAME:="EMP";
```

%Declaring library entry point

```
LIBRARY DMI;
BOOLEAN PROCEDURE DBGETDOUBLE(STRUCTURE,ITEM,VARIABLE);
  STRING STRUCTURE, ITEM;
  DOUBLE VARIABLE;
  LIBRARY DMI (ACTUALNAME="ALGOLGETDOUBLE");
```

%Invoking entry point

```
ITEM_NAME:="EMP_SSN";
RSLT:=DBGETDOUBLE(DATA_SET_NAME,ITEM_NAME,EXPR_2);
IF RSLT
  THEN EXCEPTIONHANDLER;
```

### COBOL74 or COBOL85 Program Fragment

In this example, the structure is EMP; the item is EMP-SSN. Since VAR-3 is declared as a DOUBLE item and RESULT is declared as a COMP item, this example can be either a COBOL74 or a COBOL85 program fragment.

*Note: To make this example a COBOL program fragment, declare the variable VAR-3 as a COMP-5 item and the variable RESULT as a COMP-2 item.*

\*\*\*Declaring variables

```
Ø1 DATA-SET-NAME          PIC X(17) VALUE IS "EMP".
Ø1 ITEM-NAME               PIC X(17) VALUE IS SPACES.
Ø1 VAR-3                   DOUBLE.
Ø1 RESULT                  PIC 9(1) COMP VALUE IS Ø.
```

\*\*\*Invoking entry point

```
MOVE "EMP-SSN" TO ITEM-NAME.
CALL "DBGETDOUBLE OF DMINTERPRETER"
  USING DATA-SET-NAME, ITEM-NAME, VAR-3
  GIVING RESULT.
```

**FORTRAN77 Program Fragment**

In this FORTRAN77 example, EMP is the structure. The item name is EMP-SSN. The variable is EXPR2.

\*\*\*Declaring library entry point

```
LOGICAL FUNCTION DBGETD (STR, ITEM, VAR)
  CHARACTER STR, ITEM
  DOUBLE PRECISION VAR
  IN LIBRARY DMI (ACTUALNAME = "FORTRAN77GETDOUBLE")
END
```

\*\*\*Setting variables to initial values

```
CHARACTER      DSNAME      *4      /"EMP"/,
  *             ITMNAM      *13
```

```
DOUBLE PRECISION EXPR2
```

\*\*\*Declaring local variables

```
LOGICAL      RSLT
```

\*\*\*Invoking entry point

```
ITMNAM = "EMP-SSN"
RSLT = DBGETD(DSNAME,ITMNAM,EXPR2)
IF (RSLT) CALL EXCEPT
```

# Retrieving Boolean Values

Use the GETBOOLEAN data transfer entry point to perform two functions:

- Retrieve a Boolean value from either a Boolean item or a bit in a field item.
- Set the value of a variable, as appropriate, to either TRUE or FALSE.

### Passing Parameters

The GETBOOLEAN entry points have three required parameters: the name of the structure, the name of the item, and the name of the variable.

Parameter	Explanation
<structure>	Identifies the data set or the database in which the item is located.
<item name>	Identifies the data set item whose Boolean value is to be moved to the user-declared variable.
<variable name>	Identifies the user-declared variable by its name in the program. The variable must be able to contain a Boolean value (TRUE or FALSE).

**Note:** *If an exception occurs, the value of the variable is set to show a false condition. Consult Appendix B, "DMSII Exceptions and Errors," for more information on exceptions.*

Table 4-6 gives the exported names of the entry points, as well as the parameters and the data type of each parameter.

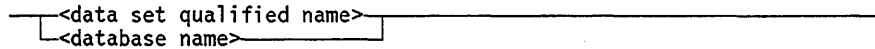
Table 4-6. Retrieving Boolean Values

Exported Name of Entry Point	Parameter	Data Type of Parameter
ALGOLGETBOOLEAN	<structure>	STRING
	<item name>	STRING
	<variable name>	BOOLEAN
DBGETBOOLEAN	<structure>	DISPLAY
	<item name>	DISPLAY
	<variable name>	DISPLAY
FORTRAN77GETBOOLEAN	<structure>	CHARACTER
	<item name>	CHARACTER
	<variable name>	LOGICAL

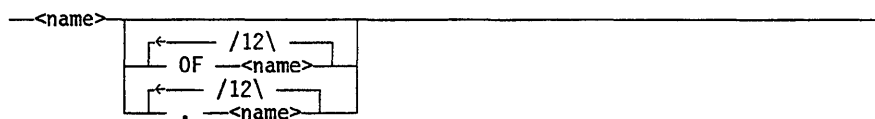
**Structure**

The structure can be either a data set or a database. The following syntax diagrams show that if the name of the data set is not unique, you can use up to 12 qualifying parameters to uniquely identify the data set. A database name cannot be qualified.

**<structure>**



**<data set qualified name>**



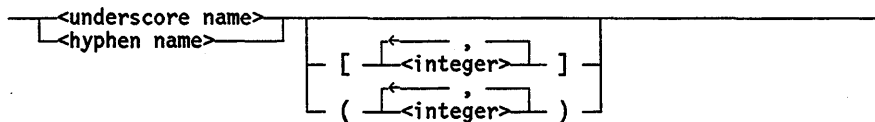
The variable elements of the structure parameter are explained as follows:

Element	Explanation
<data set qualified name>	Identifies the data set that is to be accessed at run time.
<database name>	Identifies the database for which the DMINTERPRETER library was compiled.
<name>	Identifies the name of the data set or database. A name can consist from of 1 to 17 letters, digits, and—depending on the language, either underscores ( ) or hyphens (-). The first character must be a letter. The last character must be either a letter or a digit.

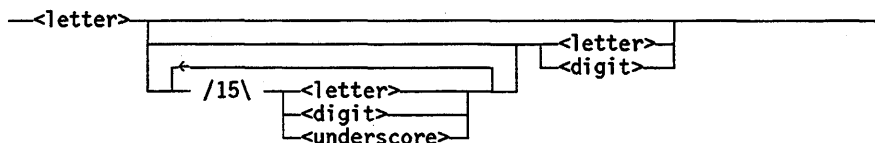
**Item Name**

The following syntax diagrams show the valid combinations of characters that can be used to form an item name:

**<item name>**



**<underscore name>**





```

%Declaring local variables

STRING
  DATA_SET_NAME, ITEM_NAME;
BOOLEAN
  EXPR_4, RSLT;

%Assigning values to parameters

DATA_SET_NAME:="EMP";

%Declaring library entry point

LIBRARY DMI;
BOOLEAN PROCEDURE DBGETBOOLEAN(STRUCTURE,ITEM,VARIABLE);
  STRING STRUCTURE, ITEM;
  BOOLEAN VARIABLE;
  LIBRARY DMI (ACTUALNAME="ALGOLGETBOOLEAN");

%Invoking entry point

ITEM_NAME:="EMP_SEC_CLEAR";
RSLT:=DBGETBOOLEAN(DATA_SET_NAME,ITEM_NAME,EXPR_4);
IF RSLT
  THEN EXCEPTIONHANDLER;

```

**COBOL74 or COBOL85 Program Fragment**

This example uses the structure EMP, the item EMP-SEC-CLEAR, and the variable VAR-1. Because the variable RESULT is declared as a COMP item, this example can be either a COBOL74 or a COBOL85 program.

**Note:** To make this example a COBOL program fragment, declare the variable RESULT as a COMP-2 item.

```

***Declaring variables

Ø1 DATA-SET-NAME          PIC X(17) VALUE IS "EMP".
Ø1 ITEM-NAME              PIC X(17) VALUE IS SPACES.
Ø1 VAR-1                  PIC X(1Ø) VALUE IS SPACES.
Ø1 RESULT                  PIC 9(1) COMP VALUE IS Ø.

***Invoking entry point

MOVE "EMP-SEC-CLEAR" TO ITEM-NAME.
CALL "DBGETBOOLEAN OF DMINTERPRETER"
  USING DATA-SET-NAME, ITEM-NAME, VAR-1
  GIVING RESULT.

```



## Retrieving Boolean Values (cont.)

---

### FORTRAN77 Program Fragment

In this FORTRAN77 example, a value for DSNAME is initialized to EMP at the beginning of the program. The value of EMP-SEC-CLEAR is assigned to ITMNAM in the body of the program. The variable is EXPR4.

\*\*\*Declaring library entry point

```
LOGICAL FUNCTION DBGETL (STR, ITEM, VAR)
  CHARACTER STR, ITEM
  LOGICAL VAR
  IN LIBRARY DMI (ACTUALNAME = "FORTRAN77GETLOGICAL")
END
```

\*\*\*Setting variables to initial values

```
CHARACTER DSNAME      *4      /"EMP"/,
  *      ITMNAM        *13
```

```
LOGICAL  EXPR4
```

\*\*\*Declaring local variables

```
LOGICAL  RSLT
```

\*\*\*Invoking entry point

```
ITMNAM = "EMP-SEC-CLEAR"
RSLT = DBGETL(DSNAME,ITMNAM,EXPR4)
IF (RSLT) CALL EXCEPT
```

## Placing Strings into Data Items

Using a DBPUT data transfer entry point, you can move a string from a program-declared variable into a data item. The data item must be declared in the DASDL description as either an alpha item or a group item.

### Passing Parameters

The data transfer entry points that move a string from a program-declared variable to a data item have three required parameters: the name of the structure, the name of the item, and the expression containing the string.

Parameter	Explanation
<structure>	Identifies either the data set (for an alpha item) or the database (for a global item) that is to be accessed at run time.
<item name>	Identifies the data set item to be modified.
<expression>	Identifies the user-declared variable by its name in the program.

**Note:** *If the string exceeds the defined length of the data item, DMSII returns an exception to the program.*

Table 4-7 gives the exported names of the entry points, as well as the parameters and the data type of each parameter.

Table 4-7. Placing Strings into Data Items

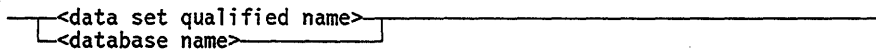
Exported Name of Entry Point	Parameter	Data Type of Parameter
ALGOLPUTSTRING	<structure>	STRING
	<item name>	STRING
	<expression>	STRING
DBPUTDISPLAY	<structure>	DISPLAY
	<item name>	DISPLAY
	<variable>	DISPLAY
FORTRAN77PUTCHARACTER	<structure>	CHARACTER
	<item name>	CHARACTER
	<expression>	CHARACTER

### Structure

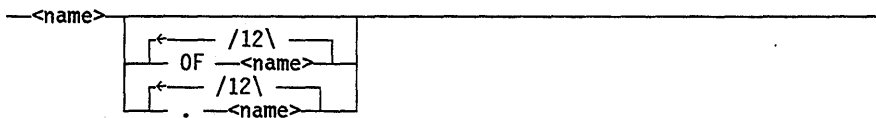
The structure can be either a data set or a database. The following syntax diagrams show that if the name of the data set is not unique, you can use up to 12 qualifying parameters to uniquely identify the data set. A database name cannot be qualified.

## Placing Strings into Data Items (cont.)

<structure>



<data set qualified name>



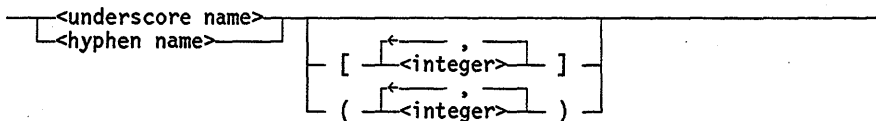
The variable elements of the structure parameter are explained as follows:

Element	Explanation
<data set qualified name>	Identifies the data set that is to be accessed at run time.
<database name>	Identifies the database for which the DMINTERPRETER library was compiled.
<name>	Identifies the name of the data set or database. A name can consist of from 1 to 17 letters, digits, and—depending on the language—either underscores ( ) or hyphens (-). The first character must be a letter. The last character must be either a letter or a digit.

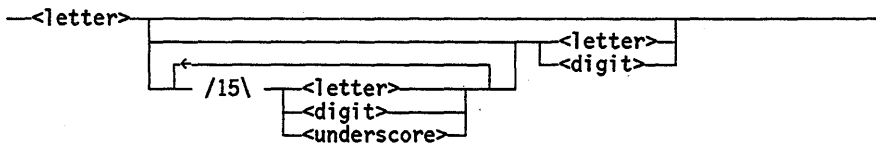
### Item Name

The following syntax diagrams show the valid combinations of characters that can be used to form an item name:

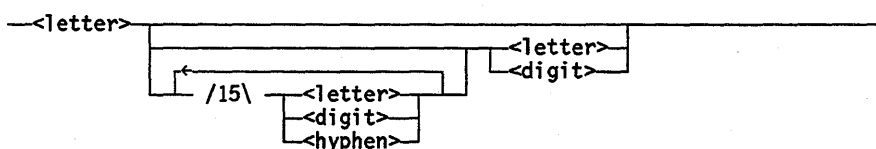
<item name>



<underscore name>



<hyphen name>



The variable elements of a name are explained as follows:

Element	Explanation
<underscore name>	A string of from 1 to 17 letters, digits, and underscores ( _ ), starting with a letter and ending with either a letter or a digit.
<hyphen name>	A string of from 1 to 17 letters, digits, and hyphens ( - ), starting with a letter and ending with either a letter or a digit.
<integer>	A subscript for an OCCURS clause item. An integer is a whole number (0123456789) that consists of from 1 to 12 digits and that has a maximum value of 549755813887.

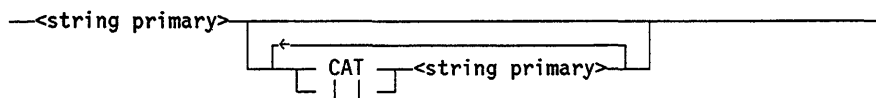
### Expression

Each language has a different way of identifying the expression that contains or evaluates to the string:

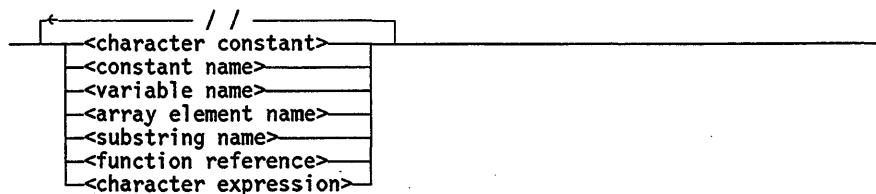
- For ALGOL, it is a string expression.
- For COBOL, it is a DISPLAY variable.
- For FORTRAN77, it is a character expression.

The syntax for the ALGOL string expression and for the FORTRAN77 character expression are shown in the following diagrams:

<string expression>



<character expression>



Consult the appropriate language reference manual for more information on expressions that can contain a string.

### Program Examples

The following program fragments show how to use data transfer entry points to move data item values from string expressions declared in the application program to a DMSII user work area.

## Placing Strings into Data Items (cont.)

---

### ALGOL Program Fragment

In the ALGOL example, the structure is EMP, the item name is EMP\_FNAME, and EXPR\_3 contains the string expression. Concatenation is used to create a string for the employee name being put into the database.

```
%Declaring local variables

STRING
  DATA_SET_NAME, ITEM_NAME, EXPR_3;
BOOLEAN
  RSLT;

%Assigning values to parameters

DATA_SET_NAME:="EMP";

%Declaring library entry point

LIBRARY DMI;
BOOLEAN PROCEDURE DBPUTSTRING(STRUCTURE,ITEM,VARIABLE);
  STRING STRUCTURE, ITEM, VARIABLE;
  LIBRARY DMI (ACTUALNAME="ALGOLPUTSTRING");

%Invoking entry point

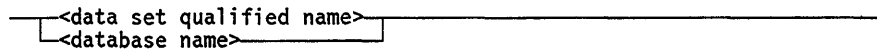
ITEM_NAME:="EMP_FNAME";
RSLT:=DBPUTSTRING(DATA_SET_NAME,ITEM_NAME,EXPR_3 CAT "LINA");
IF RSLT
  THEN EXCEPTIONHANDLER;
```

### COBOL74 or COBOL85 Program Fragment

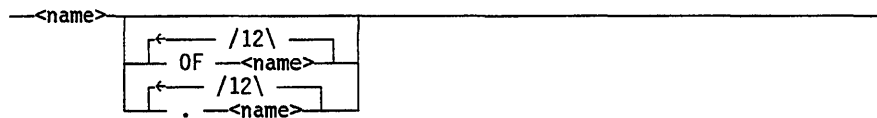
This example declares the structure as EMP, the item as EMP-FNAME, and the variable as VAR-1. Because the variable RESULT is declared as a COMP item, the example could be either a COBOL74 or a COBOL85 program fragment.

*Note: If the variable RESULT is declared as a COMP-2, this example could be used as a COBOL program fragment.*

**<structure>**



**<data set qualified name>**



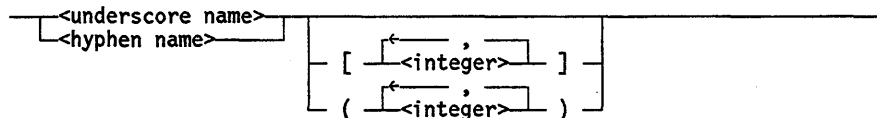
The variable elements of the structure parameter are explained as follows:

Element	Explanation
<data set qualified name>	Identifies the data set that is to be accessed at run time.
<database name>	Identifies the database for which the DMINTERPRETER library was compiled.
<name>	Identifies the name of the data set or database. A name can consist of from 1 to 17 letters, digits, and—depending on the language—either underscores (_) or hyphens (-). The first character must be a letter. The last character must be either a letter or a digit.

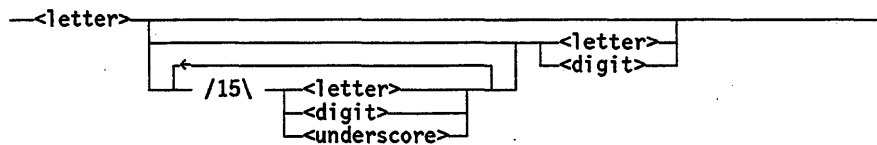
**Item Name**

The following syntax diagrams show the valid combinations of characters that can be used to form an item name:

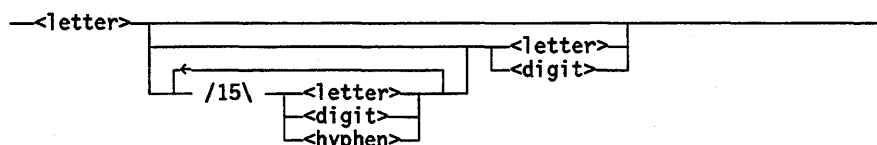
**<item name>**



**<underscore name>**



**<hyphen name>**



## Placing Strings into Kanji Alpha Items (cont.)

The variable elements of an item name are explained as follows:

Element	Explanation
<underscore name>	A string of from 1 to 17 letters, digits, and underscores (_), starting with a letter and ending with either a letter or a digit.
<hyphen name>	A string of from 1 to 17 letters, digits, and hyphens (-), starting with a letter and ending with either a letter or a digit.
<integer>	A subscript for an OCCURS clause item. An integer is a whole number (0123456789) that consists of from 1 to 12 digits and that has a maximum value of 549755813887.

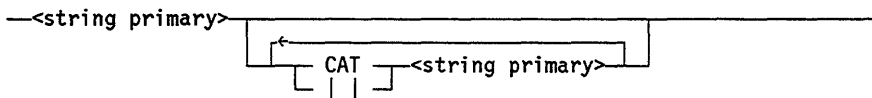
### Expression

Each language has a different way of identifying the variable that contains or evaluates to the string:

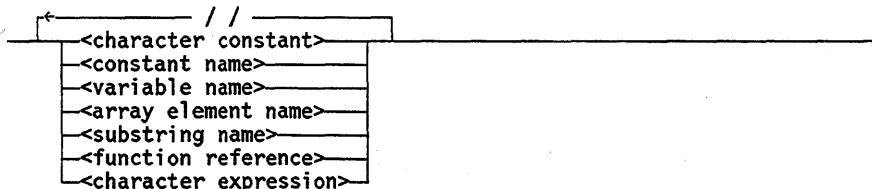
- For ALGOL, it is a string expression.
- For COBOL74 and COBOL85, it is a variable. It can be defined as PIC X(2n) or PIC X(n) USAGE IS KANJI.
- For FORTRAN77, it is a character expression.

*Note: Only COBOL74 and COBOL85 support Kanji characters; COBOL does not.*

### <string expression>



### <character expression>



Consult the appropriate language reference manual for more information on variables that can contain a Kanji string.

### Program Examples

The following program fragments use PUTKANJI entry points to move data item values from string expressions declared in the application program to a DMSII user work area.

\*\*\*Declaring variables

```

Ø1 DATA-SET-NAME      PIC X(17) VALUE IS "EMP".
Ø1 ITEM-NAME          PIC X(17) VALUE IS SPACES.
Ø1 VAR-1              PIC X(1Ø) VALUE IS SPACES.
Ø1 RESULT             PIC 9(1) COMP VALUE IS Ø.
    
```

\*\*\*Invoking entry point

```

MOVE "EMP-FNAME" TO ITEM-NAME.
CALL "DBPUTDISPLAY OF DMINTERPRETER"
    USING DATA-SET-NAME, ITEM-NAME, VAR-1
    GIVING RESULT.
    
```

#### **FORTRAN77 Program Fragment**

In this FORTRAN77 example, DSNNAME contains the structure; ITMNAM is the item name, EMP-FNAME; and EXPR3 contains the character expression. Concatenation is used to create a string for the employee name being put into the database.

\*\*\*Declaring library entry point

```

LOGICAL FUNCTION DBPUTC (STR, ITEM, VAR)
  CHARACTER STR, ITEM, VAR
  IN LIBRARY DMI (ACTUALNAME = "FORTRAN77PUTCHARACTER")
END
    
```

\*\*\*Setting variables to initial values

```

CHARACTER DSNNAME      *4      /"EMP"/,
*      ITMNAM          *13     ,
*      EXPR3           *22
    
```

\*\*\*Declaring local variables

```

LOGICAL  RSLT
    
```

\*\*\*Invoking entry point

```

ITMNAM = "EMP-FNAME"
EXPR3 = "CATHY"
RSLT = DBPUTC(DSNNAME,ITMNAM,EXPR3)
IF (RSLT) CALL EXCEPT
    
```



## Placing Strings into Kanji Alpha Items

Using a PUTKANJI data transfer entry point, you can move a Kanji string into the user work area. The Kanji string must be in an alpha item that was declared in the DASDL description as USAGE IS KANJI.

### Passing Parameters

The PUTKANJI entry points have three required parameters: the name of the structure, the name of the item, and the program-declared expression containing the string.

Parameter	Explanation
<structure>	Identifies either the data set or the database that is to be accessed at run time.
<item name>	Identifies the data set item to be modified.
<expression>	Identifies the user-declared variable by its name in the program.

**Note:** *If the string exceeds the defined length of the data item, DMSII returns an exception to the program.*

Table 4-8 gives the exported names of the entry points, as well as the parameters and the data type of each parameter.

**Table 4-8. Placing Strings into Kanji Alpha Items**

Exported Name of Entry Point	Parameter	Data Type of Parameter
ALGOLPUTKANJI	<structure>	STRING
	<item name>	STRING
	<expression>	STRING
DBPUTKANJI	<structure>	DISPLAY
	<item name>	DISPLAY
	<variable>	DISPLAY or KANJI
FORTRAN77PUTKANJI	<structure>	CHARACTER
	<item name>	CHARACTER
	<expression>	CHARACTER

### Structure

The structure can be either a data set or a database. The following syntax diagrams show that if the name of the data set is not unique, you can use up to 12 qualifying parameters to uniquely identify the data set. A database name cannot be qualified.

### ALGOL Program Fragment

In the ALGOL example, the structure is EMP, the item is EMP\_KNAME, and the variable is EXPR\_3. Concatenation is used to create a string for the employee name before it is placed in the database. KANJI\_STRING is a string item containing Kanji characters that do not have a start-of-Kanji (SOK) or end-of-Kanji (EOK) characters.

%Declaring local variables

```
STRING
  DATA_SET_NAME, ITEM_NAME, EXPR_3;
BOOLEAN
  RSLT;
```

%Assigning values to parameters

```
DATA_SET_NAME:="EMP";
```

%Declaring library entry point

```
LIBRARY DMI;
BOOLEAN PROCEDURE DBPUTKANJI(STRUCTURE,ITEM,VARIABLE);
  STRING STRUCTURE, ITEM, VARIABLE;
  LIBRARY DMI (ACTUALNAME="ALGOLPUTKANJI");
```

%Invoking entry point

```
ITEM_NAME:="EMP_KNAME";
EXPR_3:=KANJI_STRING;
RSLT:=DBPUTKANJI(DATA_SET_NAME,ITEM_NAME,EXPR_3 CAT KANJI_STRING);
IF RSLT
  THEN EXCEPTIONHANDLER;
```

### COBOL74 or COBOL85 Program Fragment

In this example, the structure is EMP, the item is EMP-KNAME, and the variable is VAR-1. The variable RESULT is declared as a COMP item. This program fragment could appear in either a COBOL74 or a COBOL85 program.

*Note: COBOL does not support Kanji characters.*

## Placing Strings into Kanji Alpha Items (cont.)

---

\*\*\*Declaring variables

```
Ø1 DATA-SET-NAME      PIC X(17) VALUE IS "EMP".
Ø1 ITEM-NAME           PIC X(17) VALUE IS SPACES.
Ø1 VAR-1               PIC X(1Ø) USAGE IS KANJI.
                        VALUE IS SPACES.
Ø1 RESULT              PIC 9(1) COMP VALUE IS Ø.
```

\*\*\*Invoking entry point

```
MOVE "EMP-KNAME" TO ITEM-NAME.
CALL "DBPUTKANJI OF DMINTERPRETER"
     USING DATA-SET-NAME, ITEM-NAME, VAR-1
     GIVING RESULT.
```

### FORTRAN77 Program Fragment

In this FORTRAN77 example, a value for DSNAME is initialized at the beginning of the program; the values of ITMNAM and EXPR3 are assigned in the body of the program. KNJSTR is a string item containing Kanji characters without the start-of-Kanji (SOK) or end-of-Kanji (EOK) characters.

\*\*\*Declaring library entry point

```
LOGICAL FUNCTION DBPUTK (STR, ITEM, VAR)
  CHARACTER STR, ITEM, VAR
  IN LIBRARY DMI (ACTUALNAME = "FORTRAN77PUTKANJI")
END
```

\*\*\*Setting variables to initial values

```
CHARACTER DSNAME      *4      /"EMP"/,
*      ITMNAM          *13     ,
*      EXPR3           *22
```

\*\*\*Declaring local variables

```
LOGICAL  RSLT
```

\*\*\*Invoking entry point

```
ITMNAM = "EMP-KNAME"
EXPR3 = KNJSTR
RSLT = DBPUTK(DSNAME,ITMNAM,EXPR3)
IF (RSLT) CALL EXCEPT
```

## Placing Numeric Values into Data Items

Use the PUTREAL data transfer entry point to place a numeric value from a number, real, or field item into the user work area.

### Passing Parameters

The PUTREAL entry points have three required parameters: the name of the structure, the name of the item, and the name of the expression that contains the numeric value.

Parameter	Explanation
<structure>	Identifies the data set or the database that is to be accessed at run time.
<item name>	Identifies the data set item to be modified.
<expression>	Identifies the user-declared variable by its name in the program.

### Notes:

- *If the parameter for the real value produces an incorrect new value, DMSII returns an exception to the program.*
- *If the number of significant digits exceeds the capacity of the data item, use the ALGOLPUTDOUBLE, DBPUTDOUBLE, or FORTRAN77PUTDOUBLE entry point to place the real value into the item.*

Table 4-9 gives the exported names of the entry points, as well as the parameters and the data type of each parameter.

Table 4–9. Placing Numeric Values into Data Items

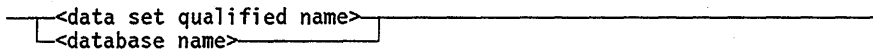
Exported Name of Entry Point	Parameter	Data Type of Parameter
ALGOLPUTREAL	<structure>	STRING
	<item name>	STRING
	<expression>	REAL
DBPUTREAL	<structure>	DISPLAY
	<item name>	DISPLAY
	<variable>	COMP-4 (COBOL)
		REAL (COBOL74 or COBOL85)
FORTRAN77PUTREAL	<structure>	CHARACTER
	<item name>	CHARACTER
	<expression>	REAL

## Placing Numeric Values into Data Items (cont.)

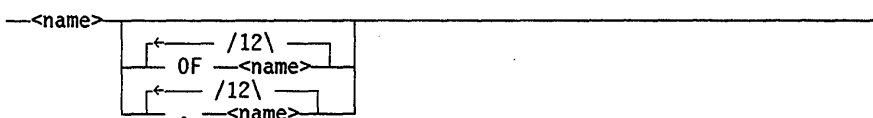
### Structure

The structure can be either a data set or a database. The following syntax diagrams show that if the name of the data set is not unique, you can use up to 12 qualifying parameters to uniquely identify the data set. A database name cannot be qualified.

#### <structure>



#### <data set qualified name>



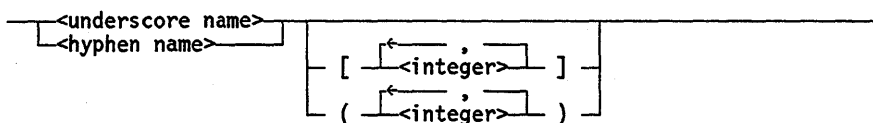
The variable elements of the structure parameter are explained as follows:

Element	Explanation
<data set qualified name>	Identifies the data set that is to be accessed at run time.
<database name>	Identifies the database for which the DMINTERPRETER library was compiled.
<name>	Identifies the name of the data set or database. A name can consist of from 1 to 17 letters, digits, and—depending on the language—either underscores ( _ ) or hyphens ( - ). The first character must be a letter. The last character must be either a letter or a digit.

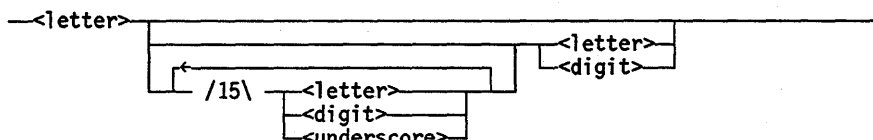
### Item Name

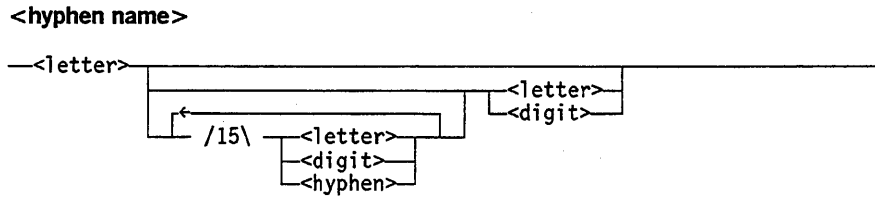
The following syntax diagrams show the valid combinations of characters that can be used to form an item name:

#### <item name>



#### <underscore name>





The variable elements of an item name are explained as follows:

Element	Explanation
<underscore name>	A string of from 1 to 17 letters, digits, and underscores (_), starting with a letter and ending with either a letter or a digit.
<hyphen name>	A string of from 1 to 17 letters, digits, and hyphens (-), starting with a letter and ending with either a letter or a digit.
<integer>	A subscript for an OCCURS clause item. An integer is a whole number (0123456789) that consists of from 1 to 12 digits and that has a maximum value of 549755813887.

**Expression**

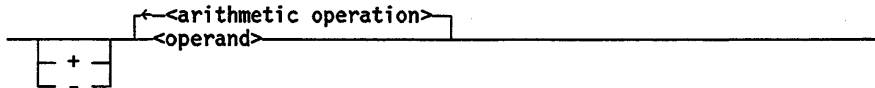
Each language has a different way of identifying the entity that contains or evaluates to a real numeric value:

- For ALGOL, it is an arithmetic expression.
- For COBOL, the variable must be declared in the WORKING-STORAGE section of the program as a level 77 COMP-4 item.
- For COBOL74 and COBOL85, the variable must be declared in the WORKING-STORAGE section of the program as a level 77 REAL item.
- For FORTRAN77, it is an arithmetic expression.

**<arithmetic expression> in ALGOL**



**<arithmetic expression> in FORTRAN77**



Consult the appropriate language reference manual for more information on variables that can contain a real numeric value.

**Program Examples**

The following program fragments use PUTREAL entry points to move numeric values from variables declared in the application program to the user work area.

## Placing Numeric Values into Data Items (cont.)

---

### ALGOL Program Fragment

In the ALGOL example, a value for DATA\_SET\_NAME (the structure) is initialized at the beginning of the program; the value for ITEM\_NAME (item name) and the value for EXPR\_1 (the variable) are assigned in the body of the program.

```
%Declaring local variables

REAL
  EXPR_1;
STRING
  DATA_SET_NAME, ITEM_NAME;
BOOLEAN
  RSLT;

%Assigning values to parameters

DATA_SET_NAME:="EMP";

%Declaring library entry point

LIBRARY DMI;
BOOLEAN PROCEDURE DBPUTREAL(STRUCTURE,ITEM,VARIABLE);
  STRING STRUCTURE, ITEM;
  REAL VARIABLE;
  LIBRARY DMI (ACTUALNAME="ALGOLPUTREAL");

%Invoking entry point

ITEM_NAME:="EMP_NO";
EXPR_1:="11";
RSLT:=DBPUTREAL(DATA_SET_NAME,ITEM_NAME,EXPR_1);
IF RSLT
  THEN EXCEPTIONHANDLER;
```

### COBOL74 or COBOL85 Program Fragment

This example shows either a COBOL74 or a COBOL85 program fragment. The variable VAR-2 is declared as a REAL item and RESULT as a COMP item. The structure is EMP and the item is EMP-NO.

**Note:** *If VAR-2 is declared as a COMP-4 item and RESULT is declared as a COMP-2 item, this example could be used in a COBOL program.*

\*\*\*Declaring variables

```

Ø1 DATA-SET-NAME      PIC X(17) VALUE IS "EMP".
Ø1 ITEM-NAME           PIC X(17) VALUE IS SPACES.
Ø1 VAR-2               REAL.
Ø1 RESULT              PIC 9(1) COMP VALUE IS Ø.
    
```

\*\*\*Invoking entry point

```

MOVE "EMP-NO" TO ITEM-NAME.
MOVE 11 TO VAR-2.
CALL "DBPUTREAL OF DMINTERPRETER"
     USING DATA-SET-NAME, ITEM-NAME, VAR-2
     GIVING RESULT.
    
```

#### **FORTRAN77 Program Fragment**

In this FORTRAN77 example, a value for DSNNAME is initialized at the beginning of the program; the value of ITMNAM and the value of EXPR1 are assigned in the body of the program.

\*\*\*Declaring library entry point

```

LOGICAL FUNCTION DBPUTR (STR, ITEM, VAR)
  CHARACTER STR, ITEM
  REAL VAR
  IN LIBRARY DMI (ACTUALNAME = "FORTRAN77PUTREAL")
END
    
```

\*\*\*Setting variables to initial values

```

CHARACTER DSNNAME      *4      /"EMP"/,
*      ITMNAM          *13
    
```

```

REAL      EXPR1
    
```

\*\*\*Declaring local variables

```

LOGICAL      RSLT
    
```

\*\*\*Invoking entry point

```

ITMNAM = "EMP-NO"
EXPR1 = 11
RSLT = DBPUTR(DSNNAME,ITMNAM,EXPR1)
IF (RSLT) CALL EXCEPT
    
```



## Placing Double-Precision Values into Data Items

Use the PUTDOUBLE data transfer entry points to move a double-precision numeric value from a program-declared variable to a data item. The data item must be defined in the DASDL description as a number, real, or field item. (See “Placing Numeric Values into Data Items” in this section.)

### Passing Parameters

The PUTDOUBLE entry points have three required parameters: the name of the structure, the name of the item, and the name of the expression that contains the numeric value.

Parameter	Explanation
<structure>	Identifies the data set or the database that is to be accessed at run time.
<item name>	Identifies the data set item to be modified.
<expression>	Identifies the user-declared variable by its name in the program.

*Note: If the parameter for the real value produces an incorrect new value, DMSII returns an exception to the program.*

Table 4-10 gives the exported names of the entry points, as well as the parameters and the data type of each parameter.

**Table 4-10. Placing Double-Precision Values into Data Items**

Exported Name of Entry Point	Parameter	Data Type of Parameter
ALGOLPUTDOUBLE	<structure>	STRING
	<item name>	STRING
	<expression>	DOUBLE
DBPUTDOUBLE	<structure>	DISPLAY
	<item name>	DISPLAY
	<variable>	COMP-5 (COBOL) DOUBLE (COBOL74 or COBOL85)
FORTRAN77PUTDOUBLE	<structure>	CHARACTER
	<item name>	CHARACTER
	<expression>	DOUBLE

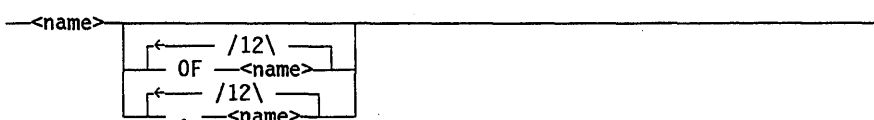
**Structure**

The structure can be either a data set or a database. The following syntax diagrams show that if the name of the data set is not unique, you can use up to 12 qualifying parameters to uniquely identify the data set. A database name cannot be qualified.

**<structure>**



**<data set qualified name>**



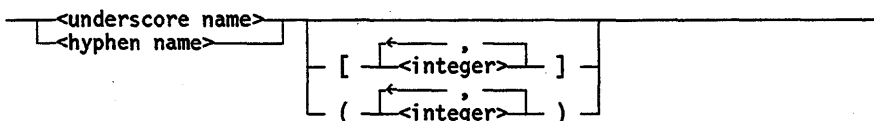
The variable elements of the structure parameter are explained as follows:

Element	Explanation
<data set qualified name>	Identifies the data set that is to be accessed at run time.
<database name>	Identifies the database for which the DMINTERPRETER library was compiled.
<name>	Identifies the name of the data set or database. A name can consist of from 1 to 17 letters, digits, and—depending on the language—either underscores (_) or hyphens (-). The first character must be a letter. The last character must be either a letter or a digit.

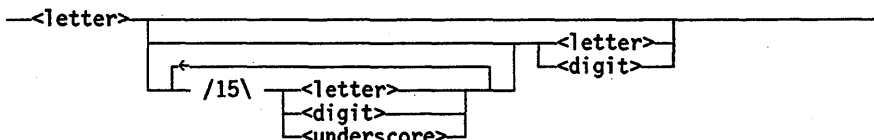
**Item Name**

The following syntax diagrams show the valid combinations of characters that can be used to form an item name:

**<item name>**

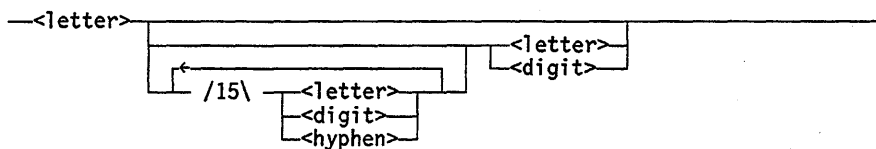


**<underscore name>**



## Placing Double-Precision Values into Data Items (cont.)

<hyphen name>



The variable elements of an item name are explained as follows:

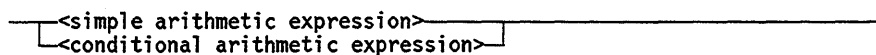
Element	Explanation
<underscore name>	A string of from 1 to 17 letters, digits, and underscores (_), starting with a letter and ending with either a letter or a digit.
<hyphen name>	A string of from 1 to 17 letters, digits, and hyphens (-), starting with a letter and ending with either a letter or a digit.
<integer>	A subscript for an OCCURS clause item. An integer is a whole number (0123456789) that consists of from 1 to 12 digits and that has a maximum value of 549755813887.

### Expression

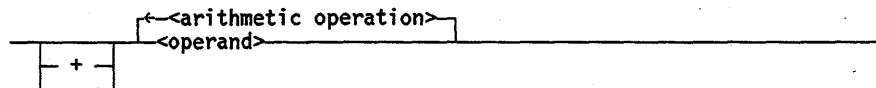
Each language has a different way of identifying the entity that contains or evaluates to a real, double-precision numeric value:

- For ALGOL, it is an arithmetic expression.
- For COBOL, the variable must be declared in the WORKING-STORAGE section of the program as a level 77 COMP-5 item.
- For COBOL74 and COBOL85, the variable must be declared in the WORKING-STORAGE section of the program as a level 77 DOUBLE item.
- For FORTRAN77, it is an arithmetic expression.

### <arithmetic expression> in ALGOL



### <arithmetic expression> in FORTRAN77



Consult the appropriate language reference manual for more information on variables that can contain a real, double-precision, numeric value.

### Program Examples

The following program fragments use PUTDOUBLE entry points to move numeric values from variables declared in the application program to the user work area.

**ALGOL Program Fragment**

In the ALGOL example, the structure is EMP, the item is EMP-SSN, and the arithmetic expression is (EXPR\_2 - 11).

%Declaring local variables

```
DOUBLE
  EXPR_2;
STRING
  DATA_SET_NAME, ITEM_NAME;
BOOLEAN
  RSLT;
```

%Assigning values to parameters

```
DATA_SET_NAME:="EMP";
```

%Declaring library entry point

```
LIBRARY DMI;
BOOLEAN PROCEDURE DBPUTDOUBLE (STRUCTURE,ITEM,VARIABLE);
  STRING STRUCTURE, ITEM;
  DOUBLE VARIABLE;
  LIBRARY DMI (ACTUALNAME="ALGOLPUTDOUBLE");
```

%Invoking entry point

```
ITEM_NAME:="EMP_SSN";
EXPR_2:="573230911";
RSLT:=DBPUTDOUBLE(DATA_SET_NAME,ITEM_NAME,EXPR_2 -11);
IF RSLT
  THEN EXCEPTIONHANDLER;
```

**COBOL74 or COBOL85 Program Fragment**

Because VAR-3 is declared as a DOUBLE item and RESULT is declared as a COMP item, this program fragment can be used in either a COBOL74 or a COBOL85 program.

**Note:** *By declaring the variable VAR-3 a COMP-5 item and RESULT as a COMP-2 item, this example becomes a COBOL program fragment.*

## Placing Double-Precision Values into Data Items (cont.)

---

\*\*\*Declaring variables

```
Ø1 DATA-SET-NAME      PIC X(17) VALUE IS "EMP".
Ø1 ITEM-NAME           PIC X(17) VALUE IS SPACES.
Ø1 VAR-3               DOUBLE.
Ø1 RESULT              PIC 9(1) COMP VALUE IS Ø.
```

\*\*\*Invoking entry point

```
MOVE "EMP-SSN" TO ITEM-NAME.
MOVE 57323Ø911 TO VAR-3.
CALL "DBPUTDOUBLE OF DMINTERPRETER"
     USING DATA-SET-NAME, ITEM-NAME, VAR-3
     GIVING RESULT.
```

### FORTRAN77 Program Fragment

In this FORTRAN77 example, a value for DSNAME is initialized at the beginning of the program; the value of ITMNAM and the value of EXPR2 are assigned in the body of the program. The arithmetic expression is (EXPR2 - 11).

\*\*\*Declaring library entry point

```
LOGICAL FUNCTION DBPUTD (STR, ITEM, VAR)
  CHARACTER STR, ITEM
  DOUBLE PRECISION VAR
  IN LIBRARY DMI (ACTUALNAME = "FORTRAN77PUTDOUBLE")
END
```

\*\*\*Setting variables to initial values

```
CHARACTER      DSNAME      *4      /"EMP"/,
*              ITMNAM      *13

DOUBLE PRECISION  EXPR2
```

\*\*\*Declaring local variables

```
LOGICAL      RSLT
```

\*\*\*Invoking entry point

```
ITMNAM = "EMP-SSN"
EXPR2 = 57323Ø911
RSLT = DBPUTD(DSNAME,ITMNAM,EXPR2 - 11 )
IF (RSLT) CALL EXCEPT
```

## Setting Data Items to Boolean Values

You can use a DBPUT data transfer entry point to place a Boolean value into a BOOLEAN item.

### Passing Parameters

The data transfer entry points that place Boolean values into Boolean items have three required parameters: the name of the structure, the name of the item, and the expression containing the Boolean value.

Parameter	Explanation
<structure>	Identifies the data set or the database that is to be accessed at run time.
<item name>	Identifies the data set item to be modified.
<expression>	Identifies the user-declared variable by its name in the program. In all cases, the expression must evaluate to either TRUE or FALSE.

Table 4-11 gives the exported names of the entry points, as well as the parameters and the data type of each parameter.

Table 4-11. Setting Data Items to Boolean Values

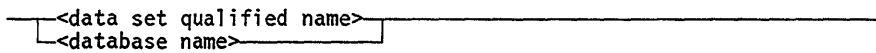
Exported Name of Entry Point	Parameter	Data Type of Parameter
ALGOLPUTBOOLEAN	<structure>	STRING
	<item name>	STRING
	<expression>	BOOLEAN
DBPUTBOOLEAN	<structure>	DISPLAY
	<item name>	DISPLAY
	<variable>	DISPLAY
FORTRAN77PUTLOGICAL	<structure>	CHARACTER
	<item name>	CHARACTER
	<expression>	LOGICAL

### Structure

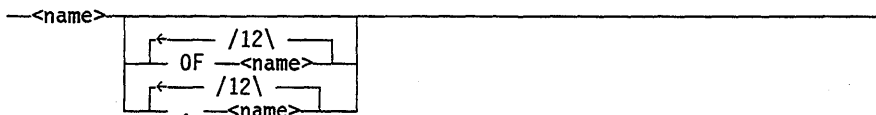
The structure can be either a data set or a database. The following syntax diagrams show that if the name of the data set is not unique, you can use up to 12 qualifying parameters to uniquely identify the data set. A database name cannot be qualified.

## Setting Data Items to Boolean Values (cont.)

### <structure>



### <data set qualified name>



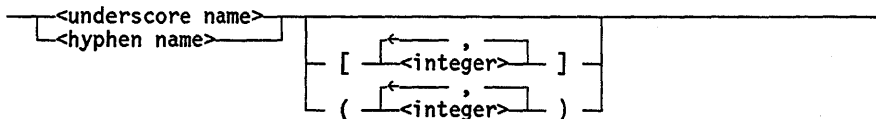
The variable elements of the structure parameter are explained as follows:

Element	Explanation
<data set qualified name>	Identifies the data set that is to be accessed at run time.
<database name>	Identifies the database for which the DMINTERPRETER library was compiled.
<name>	Identifies the name of the data set or database. A name can consist of from 1 to 17 letters, digits, and—depending on the language—either underscores (_) or hyphens (-). The first character must be a letter. The last character must be either a letter or a digit.

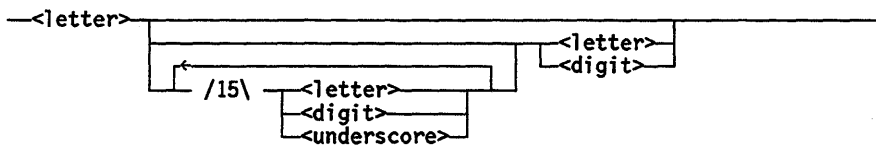
### Item Name

The following syntax diagrams show the valid combinations of characters that can be used to form an item name:

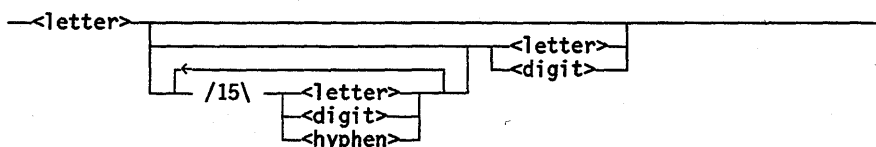
### <item name>



### <underscore name>



### <hyphen name>



The variable elements of an item name are explained as follows:

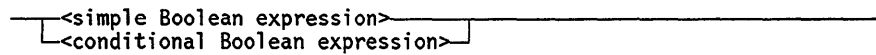
Element	Explanation
<underscore name>	A string of from 1 to 17 letters, digits, and underscores (_), starting with a letter and ending with either a letter or a digit.
<hyphen name>	A string of from 1 to 17 letters, digits, and hyphens (-), starting with a letter and ending with either a letter or a digit.
<integer>	A subscript for an OCCURS clause item. An integer is a whole number (0123456789) that consists of from 1 to 12 digits and that has a maximum value of 549755813887.

**Expression**

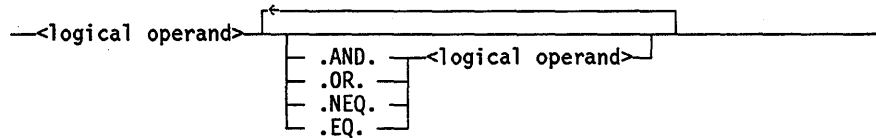
The Boolean parameter must evaluate to either the value TRUE or FALSE:

- For ALGOL, it is a Boolean expression.
- For COBOL, it is a DISPLAY variable that holds the Boolean value.
- For FORTRAN77, it is a logical expression and must evaluate to either *.TRUE* or *.FALSE*. (Note that the period preceding the keyword is required.)

**<Boolean expression>**



**<logical expression>**



Consult the appropriate language reference manual for more information on Boolean expressions.

**Program Examples**

The following program fragments use data transfer entry points to move a value of FALSE from a Boolean item in the program to a user work area.

**ALGOL Program Fragment**

In the ALGOL example, DATA\_SET\_NAME is the structure, ITEM\_NAME is the item, and EXPR\_4 contains the Boolean expression.



## Setting Data Items to Boolean Values (cont.)

---

%Declaring local variables

```
STRING
  DATA_SET_NAME, ITEM_NAME;
BOOLEAN
  EXPR_4, RSLT;
```

%Assigning values to parameters

```
DATA_SET_NAME:="EMP";
```

%Declaring library entry point

```
LIBRARY DMI;
BOOLEAN PROCEDURE DBPUTBOOLEAN(STRUCTURE,ITEM,VARIABLE);
  STRING STRUCTURE, ITEM;
  BOOLEAN VARIABLE;
  LIBRARY DMI (ACTUALNAME="ALGOLPUTBOOLEAN");
```

%Invoking entry point

```
ITEM_NAME:="EMP_SEC_CLEAR";
EXPR_4:=TRUE;
RSLT:=DBPUTBOOLEAN(DATA_SET_NAME,ITEM_NAME,EXPR_4);
IF RSLT
  THEN EXCEPTIONHANDLER;
```

### COBOL74 or COBOL85 Program Fragment

In this example, EMP is the structure, EMP-SEC-CLEAR is the item, and VAR-1 is the variable. The variable RESULT is declared as a COMP item, making this example a COBOL74 or a COBOL85 program fragment.

*Note:* To make this a COBOL program fragment, declare the variable *RESULT* as a COMP-2 item.

\*\*\*Declaring variables

```
Ø1 DATA-SET-NAME          PIC X(17) VALUE IS "EMP".
Ø1 ITEM-NAME                PIC X(17) VALUE IS SPACES.
Ø1 VAR-1                    PIC X(1Ø) VALUE IS SPACES.
Ø1 RESULT                   PIC 9(1) COMP VALUE IS Ø.
```

\*\*\*Invoking entry point

```
MOVE "EMP-SEC-CLEAR" TO ITEM-NAME.
MOVE "TRUE" TO VAR-1.
CALL "DBPUTBOOLEAN OF DMINTERPRETER"
  USING DATA-SET-NAME, ITEM-NAME, VAR-1
  GIVING RESULT.
```

**FORTRAN77 Program Fragment**

In this FORTRAN77 example, a value for DSNAME is initialized at the beginning of the program; the value of ITMNAM and the value of EXPR4 are assigned in the body of the program.

\*\*\*Declaring library entry point

```
LOGICAL FUNCTION DBPUTL (STR, ITEM, VAR)
  CHARACTER STR, ITEM
  LOGICAL VAR
  IN LIBRARY DMI (ACTUALNAME = "FORTRAN77PUTLOGICAL")
END
```

\*\*\*Setting variables to initial values

```
CHARACTER DSNAME      *4      /"EMP"/,
  *      ITMNAM      *13
```

```
LOGICAL  EXPR4
```

\*\*\*Declaring local variables

```
LOGICAL  RSLT
```

\*\*\*Invoking entry point

```
ITMNAM = "EMP-SEC-CLEAR"
EXPR4 = .TRUE
RSLT = DBPUTL(DSNAME,ITMNAM,EXPR4)
IF (RSLT) CALL EXCEPT
```

## Setting Data Items to Null Values

Use PUTNULL entry points to set a data item to its null value. The data item cannot be defined in the DASDL description as either a logical, a count, a record-type, or a population item.

### Passing Parameters

PUTNULL entry points have two required parameters: the name of the structure and the name of the item.

Parameter	Explanation
<structure>	Identifies the data set or the database that is to be accessed at run time.
<item name>	Identifies the data set item to be modified.

Table 4-12 gives the exported names of the entry points, as well as the parameters and the data type of each parameter.

Table 4-12. Setting Data Items to Null Values

Exported Name of Entry Point	Parameter	Data Type of Parameter
ALGOLPUTNULL	<structure>	STRING
	<item name>	STRING
DBPUTNULL	<structure>	DISPLAY
	<item name>	DISPLAY
FORTRAN77PUTNULL	<structure>	CHARACTER
	<item name>	CHARACTER

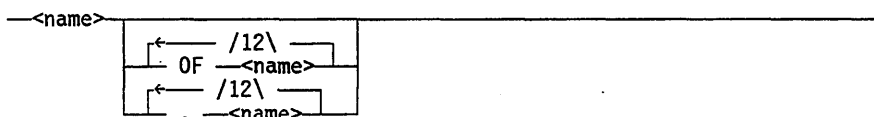
### Structure

The structure can be either a data set or a database. The following syntax diagrams show that if the name of the data set is not unique, you can use up to 12 qualifying parameters to uniquely identify the data set. A database name cannot be qualified.

<structure>



**<data set qualified name>**



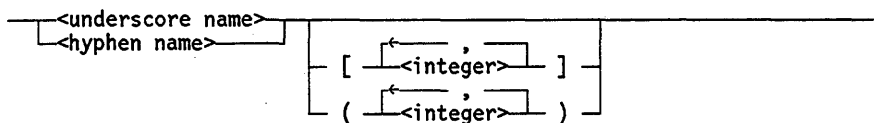
The variable elements of the structure parameter are explained as follows:

Element	Explanation
<data set qualified name>	Identifies the data set that is to be accessed at run time.
<database name>	Identifies the database for which the DMINTERPRETER library was compiled.
<name>	Identifies the name of the data set or database. A name can consist of from 1 to 17 letters, digits, and—depending on the language—either underscores ( ) or hyphens (-). The first character must be a letter. The last character must be either a letter or a digit.

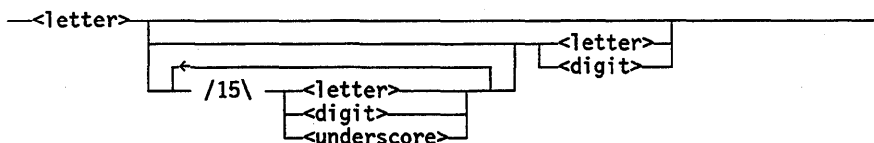
**Item Name**

The following syntax diagrams show the valid combinations of characters that can be used to form an item name:

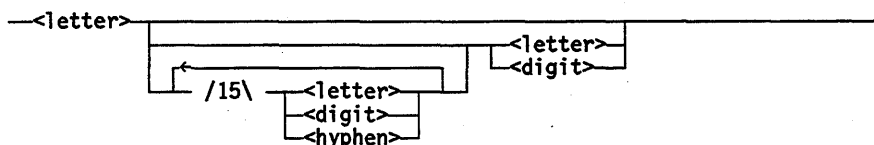
**<item name>**



**<underscore name>**



**<hyphen name>**



## Setting Data Items to Null Values (cont.)

---

The variable elements of an item name are explained as follows:

Element	Explanation
<underscore name>	A string of from 1 to 17 letters, digits, and underscores ( _ ), starting with a letter and ending with either a letter or a digit.
<hyphen name>	A string of from 1 to 17 letters, digits, and hyphens ( - ), starting with a letter and ending with either a letter or a digit.
<integer>	A subscript for an OCCURS clause item. An integer is a whole number (0123456789) that consists of from 1 to 12 digits and that has a maximum value of 549755813887.

### Program Examples

The following program fragments use the PUTNULL entry points to set data items to their null values. The structure is EMP.

#### ALGOL Program Fragment

In the ALGOL example, the data item value is EMP\_SALARY.

```
%Declaring local variables

STRING
  DATA_SET_NAME, ITEM_NAME;
BOOLEAN
  RSLT;

%Assigning values to parameters

DATA_SET_NAME:="EMP";

%Declaring library entry point

LIBRARY DMI;
BOOLEAN PROCEDURE DBPUTNULL(STRUCTURE,ITEM);
  STRING STRUCTURE, ITEM;
  LIBRARY DMI (ACTUALNAME="ALGOLPUTNULL");

%Invoking entry point

ITEM_NAME:="EMP_SALARY";
RSLT:=DBPUTBOOLEAN(DATA_SET_NAME,ITEM_NAME);
IF RSLT
  THEN EXCEPTIONHANDLER;
```

**COBOL74 or COBOL85 Program Fragment**

In the example, the data item value is EMP-SALARY. Because the variable RESULT is declared as a COMP item, this fragment can be used in a COBOL74 or a COBOL85 program.

*Note: To make this example a COBOL program fragment, declare RESULT as a COMP-2 item.*

\*\*\*Declaring variables

```
Ø1 DATA-SET-NAME      PIC X(17) VALUE IS "EMP".
Ø1 ITEM-NAME           PIC X(17) VALUE IS SPACES.
Ø1 RESULT              PIC 9(1) COMP VALUE IS Ø.
```

\*\*\*Invoking entry point

```
MOVE "EMP-SALARY" TO ITEM-NAME.
CALL "DBPUTNULL OF DMINTERPRETER"
    USING DATA-SET-NAME, ITEM-NAME
    GIVING RESULT.
```

**FORTRAN77 Program Fragment**

In the FORTRAN77 example, the data item value is EMP-SALARY.

\*\*\*Declaring library entry point

```
LOGICAL FUNCTION DBPUTN (STR, ITEM)
  CHARACTER STR, ITEM
  IN LIBRARY DMI (ACTUALNAME = "FORTRAN77PUTNULL")
END
```

\*\*\*Setting variables to initial values

```
CHARACTER DSNAME      *4    /"EMP"/,
*      ITMNAM         *13
```

\*\*\*Declaring local variables

```
LOGICAL RSLT
```

\*\*\*Invoking entry point

```
ITMNAM = "EMP-SALARY"
RSLT = DBPUTN(DSNAME,ITMNAM)
IF (RSLT) CALL EXCEPT
```

## Constructing Data Transfers during Program Execution

You can dynamically construct DBGET and DBPUT operations during program execution through a DATA data transfer entry point.

### Passing Parameters

The DATA entry points have two required parameters: the data request parameter and the data parameter.

Parameter	Explanation
<data request>	Specifies the type of operation (DBGET or DBPUT), the data type of the item to be transferred, and the structure and item involved in the operation.
<data>	For DBPUT operations, specifies the actual data that is moved to the user work area from the program.  For DBGET operations, specifies the actual data that is moved from the user work area to the program.

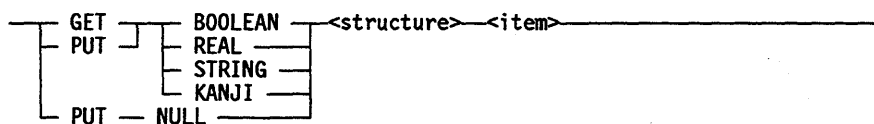
Table 4-13 gives the exported names of the entry points, as well as the parameters and the data type of each parameter.

Table 4-13. Constructing Transfers during Program Execution

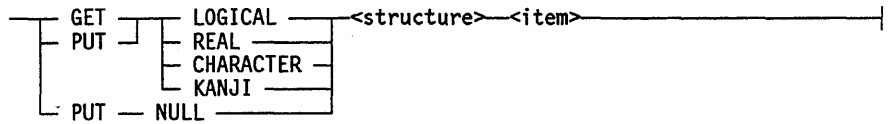
Exported Name of Entry Point	Parameter	Data Type of Parameter
ALGOLDATA	<data request>	STRING
	<data>	STRING
DBDATA	<data request>	DISPLAY
	<data>	DISPLAY
FORTRAN77DATA	<data request>	CHARACTER
	<data>	CHARACTER

There are two syntaxes for the data request parameter: one for ALGOL and COBOL, and one for FORTRAN77. All languages use the same syntax for the data parameter.

#### ALGOL and COBOL <data request>

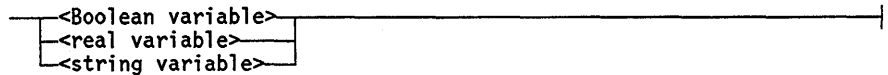


**FORTRAN77 <data request>**



Element	Explanation
GET <data type> <structure> <item>	<p>For ALGOL, COBOL, COBOL74, and COBOL85 programs, moves a data value of type Boolean, real, string, or Kanji to a variable declared in the program. The user work area is searched for the specified structure and data set item. The structure can be either a data set or a database.</p> <p>For FORTRAN77 programs, moves a data value of type logical, real, character, or Kanji to a variable declared in the program. The user work area is searched for the specified structure and data set item. The structure can be either a data set or a database.</p>
PUT <data type> <structure> <item>	<p>For ALGOL, COBOL, COBOL74, and COBOL85 programs, moves a data value of type Boolean, real, string, Kanji, or null from a variable declared in the program to the work area. During a store operation, the value will be stored in the specified structure and data set item. The structure can be either a data set or a database.</p> <p>For FORTRAN77, moves a data value of type logical, real, character, Kanji, or null from a variable declared in the program to the specified data set item in the structure. The structure can be either a data set or a database.</p>

**<data>**



Element	Explanation
<data>	A Boolean, real, or string variable declared in the program. The variable must meet the appropriate language requirements for a variable of that type. Consult the appropriate language reference manual.



### Program Examples

The following program fragments use DATA entry points to set data items to their null values.

#### ALGOL Program Fragment

%Declaring local variables

```
STRING
  DATA_REQUEST, DATA_BUFFER;
BOOLEAN
  RSLT;
```

%Assigning values to parameters

%Values are assigned to the declared variables within the body of the  
%program.

%Declaring library entry point

```
LIBRARY DMI;
BOOLEAN PROCEDURE DBDATA(REQUEST,DATA);
  STRING REQUEST, DATA;
  LIBRARY DMI (ACTUALNAME="ALGOLDATA");
```

%Invoking entry point

```
DATA_REQUEST:="PUT REAL EMP EMP-DATE-HIRED";
DATA_BUFFER:="811223";
RSLT:=DBDATA(DATA_REQUEST,DATA_BUFFER);
IF RSLT
  THEN EXCEPTIONHANDLER;
```

#### COBOL74 or COBOL85 Program Fragment

**Note:** *The variable RESULT is declared as a COMP item, making this example either a COBOL74 or a COBOL85 program fragment. If it had been declared as COMP-2 item, the fragment could be used in a COBOL program.*

\*\*\*Declaring variables

```
Ø1 DATA-REQUEST      PIC X(4Ø) VALUE IS SPACES.
Ø1 DATA-BUFFER       PIC X(1Ø) VALUE IS SPACES.
Ø1 RESULT              PIC 9(1) COMP-2 VALUE IS Ø.
```

\*\*\*Invoking entry point

```
MOVE "GET REAL EMP EMP-NO" TO DATA_REQUEST.
CALL "DBDATA OF DMINTERPRETER"
    USING DATA-REQUEST, DATA-BUFFER
    GIVING RESULT.
```

#### **FORTRAN77 Program Fragment**

\*\*\*Declaring library entry point

```
LOGICAL FUNCTION DBDATA (REQUEST, DATUM)
  CHARACTER REQUEST, DATUM
  IN LIBRARY DMI (ACTUALNAME = "FORTRAN77DATA")
END
```

\*\*\*Setting variables to initial values

```
CHARACTER DATREQ      *4Ø
*      DATBUF         *22
```

\*\*\*Declaring local variables

```
LOGICAL  RSLT
```

\*\*\*Invoking entry point

```
DATREQ = "GET REAL EMP EMP-SSN"
RSLT = DBDATA(DATREQ,DATBUF)
IF (RSLT) CALL EXCEPT
```

```
DATREQ = "GET REAL EMP EMP-DATE-HIRED"
RSLT = DBDATA(DATREQ,DATBUF)
IF (RSLT) CALL EXCEPT
```

```
DATREQ = "GET STRING EMP EMP-LNAME"
RSLT = DBDATA(DATREQ,DATBUF)
IF (RSLT) CALL EXCEPT
```



# Section 5

## Handling Exceptions

Whenever a call to a DMINTERPRETER library standard entry point, data transfer entry point, or attribute setting entry point cannot be successfully or correctly completed, the DMSII software returns an exception word to the program.

The exception word, also called a result descriptor, contains three items of information:

- A value that can be used to determine if an error or exception occurred. (An *exception* affects only the application program that detected the exception. An *error* is caused by a problem within the database. An error affects all programs that request similar data management operations.)
- The exception category and subcategory.
- The structure number involved.

The exception handling entry points allow you to return the exception word, the text of the exception message, or the name of the exception category to your program. Your program locates the text of the message and the name (type) of the category by calling the DMSUPPORT library.

*Note: All the exceptions and errors that can be returned by the language extensions can also be returned by the interpretive interface. In addition, the interpretive interface can return a category of exceptions called INTLIBERROR. These errors are valid only for the interpretive interface. For more information about exceptions that might occur, refer to Appendix B, "DMSII Exceptions and Errors."*

This section details the exception handling entry points by task. Each exception handling task with its corresponding entry points are listed in Table 5-1. Use the heading for each listed task to locate the pages that details the entry point. For example, a description of the exception handling entry points ALGOLSTATUS, DBSTATUS, and FORTRAN77STATUS is found under "Returning the Exception Word."

## Handling Exceptions

---

Table 5–1. Exception Handling Entry Points to the DMINTERPRETER Library by Task

Task	Entry Points
Returning the exception word	ALGOLSTATUS DBSTATUS FORTRAN77STATUS
Returning the text of an exception message	ALGOLEXCEPTIONTEXT DBEXCEPTIONTEXT FORTRAN77EXCEPTIONTEXT
Identifying the type of exception	ALGOLEXCEPTIONNAME DBEXCEPTIONNAME FORTRAN77EXCEPTIONNAME

The discussion of each task includes the following:

- A brief explanation of the function of the entry point
- Syntax diagrams and semantics that describe required parameters
- Program fragments that illustrate
  - Declaring variables for the entry points
  - Invoking the entry point
  - Returning results from the DMINTERPRETER library
  - Handling exceptions returned by the entry point

Unless otherwise noted, COBOL, COBOL74, and COBOL85 have the same requirements.

Refer to Section 3, “Manipulating the Database,” Section 4, “Transferring Data,” and Section 6, “Restricting Calls to the Accessroutines,” for information on standard entry points, data transfer entry points, and attribute setting entry points, respectively.

**Note:** *Until the database is explicitly opened, any call on a standard entry point, a data transfer entry point, or an attribute setting entry point results in an error. However, your program can call the three exception handling entry points and perform operations on the intrinsic data set DBSTRUCTURE whether the database is opened or closed.*

## Returning the Exception Word

The status entry point returns the exception word for the last called entry point. The exception word contains three items:

- A value that can be used to determine if an error or exception occurred
- The exception category and subcategory
- The structure number involved

The exception word can be passed to the exception text entry point (to retrieve the text of the exception message) or to the exception name entry point (to retrieve the name of the exception category).

### Passing a Parameter

For ALGOL and FORTRAN77, the status exception handling entry point has no required parameter.

For COBOL, COBOL74, and COBOL85, your program needs an exception variable.

Parameter	Explanation
<exception>	A variable containing a value that signifies whether an exception occurred during the last entry point call.  For COBOL, the variable must be declared at level 01 as a PIC 9(12) COMP-2 item. For COBOL74 and COBOL85, the variable must be declared at level 01 as a PIC 9(12) COMP item.

Table 5-2 gives the exported names of the entry points, as well as the parameters and the data type of each parameter.

Table 5-2. Returning an Exception Word

Exported Name of Entry Point	Parameter	Data Type of Parameter
ALGOLSTATUS	Not applicable.	Not applicable.
DBSTATUS	<exception>	COMP-2 (COBOL) COMP (COBOL74 and COBOL85)
FORTRAN77STATUS	Not applicable.	Not applicable.

### Program Examples

The following program fragments illustrate how to obtain an exception word from the last called entry point.

## Returning the Exception Word (cont.)

---

### ALGOL Program Fragment

%Declaring local variables

```
BOOLEAN  
  STATUSRSLT;
```

%Declaring library entry point

```
LIBRARY DMI;  
BOOLEAN PROCEDURE DBSTATUS;  
  BOOLEAN PROCEDURE DBSTATUS;  
  LIBRARY DMI (ACTUALNAME="ALGOLSTATUS");
```

%Invoking entry point

```
STATUSRSLT:=DBSTATUS;
```

### COBOL, COBOL74, or COBOL85 Program Fragment

\*\*\*Declaring variables for COBOL

```
Ø1 EXCEPTION-WORD          PIC 9(12) COMP-2.
```

\*\*\*Declaring variables for COBOL74 and COBOL85

```
Ø1 EXCEPTION-WORD          PIC 9(12) COMP.
```

\*\*\*Invoking entry point

```
CALL "DBSTATUS OF DMINTERPRETER"  
  USING EXCEPTION-WORD.
```

### FORTRAN77 Program Fragment

\*\*\*Declaring library entry point

```
LOGICAL FUNCTION DBSTAT ()  
  IN LIBRARY DMI (ACTUALNAME = "FORTRAN77STATUS")  
END
```

\*\*\*Setting variables to initial values

```
LOGICAL      RESULT
```

\*\*\*Invoking entry point

```
RESULT = DBSTAT()
```

## Returning the Text of an Exception Message

After the status operation returns the exception word to the program, you can use the exception text operation to return and display the text of an exception message.

The text of the message is returned from the DMSUPPORT library. The message is identified by the category, subcategory, and structure number in the exception word. (Refer to Appendix B, "DMSII Exceptions and Errors," for more information on categories and subcategories.)

### Notes:

- *A call on a status entry point must precede any call on an exception text entry point.*
- *If you receive an error message during an exception text operation, check whether the DMSUPPORT library is currently unavailable.*

### Passing Parameters

There are two required parameters: a variable denoting whether an exception has occurred and a variable for the text of the message.

Parameter	Explanation
<exception>	<p>A variable containing a value that signifies whether an exception occurred during the last entry point call.</p> <p>For COBOL, the variable must be declared at level 01 as a PIC 9(12) COMP-2 item. For COBOL74 and COBOL85, the variable must be declared at level 01 as a PIC 9(12) COMP item.</p>
<text of exception message >	<p>A variable containing the category name, the structure name and number (if a structure is involved), and the text of the exception or error message.</p> <p>If the text cannot fit into the variable, the message is truncated. If the text does not fill the variable, it is padded with trailing blanks.</p> <p>If no exception has occurred, the variable contains the text <i>NO EXCEPTION</i>.</p>

Table 5-3 gives the data types for the parameters and the exported names of the entry points.



## Returning the Text of an Exception Message (cont.)

---

Table 5-3. Returning the Text of an Exception Message

Exported Name of Entry Point	Parameter	Data Type of Parameter
ALGOEXCEPTIONTEXT	<exception>	BOOLEAN
	<text>	STRING
DBEXCEPTIONTEXT	<exception>	COMP-2 (COBOL) COMP (COBOL74 and COBOL85)
	<text>	DISPLAY
FORTRAN77EXCEPTIONTEXT	<exception>	LOGICAL
	<text>	CHARACTER

### Program Examples

The following program fragments use an exception text operation to obtain an exception message.

#### ALGOL Program Fragment

```
%Declaring library entry point

LIBRARY DMI;
BOOLEAN PROCEDURE DBEXCEPTIONTEXT(EXCEPTION,MSG);
  VALUE EXCEPTION;
  BOOLEAN EXCEPTION;
  STRING MSG;
  LIBRARY DMI (ACTUALNAME="ALGOEXCEPTIONTEXT");

%Exception handling procedure

PROCEDURE EXCEPTIONHANDLER;
BEGIN
  STRING EXCEPTIONMSG;
  BOOLEAN STATUSRSLT;

  STATUSRSLT:=DBSTATUS;
  DBEXCEPTIONTEXT(STATUSRSLT,EXCEPTIONMSG);
  WRITE(REMOTEFILE,<EXCEPTION_TEXT=">);
  WRITE(REMOTEFILE,<A80>,EXCEPTIONMSG);
END OF EXCEPTIONHANDLER;
```

**COBOL, COBOL74, or COBOL85 Program Fragment**

```

***Declaring variables for COBOL

Ø1 EXCEPTION-WORD          PIC 9(12) COMP-2.
Ø1 EXCEPTION-MSG           PIC X(8Ø).

***Declaring variables for COBOL74 and COBOL85

Ø1 EXCEPTION-WORD          PIC 9(12) COMP.
Ø1 EXCEPTION-MSG           PIC X(8Ø).

***Invoking entry point

CALL "DBSTATUS OF DMINTERPRETER"
    USING EXCEPTION-WORD.
CALL "DBEXCEPTIONTEXT OF DMINTERPRETER"
    USING EXCEPTION-WORD, EXCEPTION-MSG.
MOVE EXCEPTION-MSG TO REMOTE-REC.
WRITE REMOTE-REC.
    
```

**FORTRAN77 Program Fragment**

```

***Declaring library entry point

SUBROUTINE DBEXCT (EXCEPT, TEXT)
  LOGICAL EXCEPT
  CHARACTER TEXT
  IN LIBRARY DMI (ACTUALNAME = "FORTRAN77EXCEPTIONTEXT")
END

***Exception handling subroutine

SUBROUTINE EXCEPT(RESULT)

$INCLUDE "DATABASE/DMINTERPRETER" 2Ø29ØØØØ-2Ø299999

CHARACTER  MSG    *7Ø
LOGICAL    RESULT

RESULT = DBSTAT()

CALL DBEXCN(RESULT,MSG)
WRITE(6,2ØØ)MSG

CALL DBEXCT(RESULT,MSG)
WRITE(6,2ØØ)MSG
2ØØ FORMAT(1X,A)
RETURN
END
    
```

# Identifying the Type of Exception

After the status operation returns the exception word to the program, can use an exception name operation to return the type of exception. Your program can then display the exception category.

The Accessroutines can return 21 types of exceptions. Each type is considered a distinct major category. The category name is returned from the DMSUPPORT library. Refer to Appendix B, "DMSII Exceptions and Errors," for more information on categories and subcategories.

### Notes:

- *A call on a status entry point must precede any call on an exception name entry point.*
- *If you receive an error message during an exception name operation, check whether the DMSUPPORT library is currently unavailable.*

### Passing Parameters

The exception name entry points have two required parameters: a variable denoting whether an exception has occurred and a variable containing the name of the type of exception.

Parameter	Explanation
<exception>	<p>A variable containing a value that signifies whether an exception occurred during the last entry point call.</p> <p>For COBOL, the variable must be declared at level 01 as a PIC 9(12) COMP-2 item. For COBOL74 and COBOL85, the variable must be declared at level 01 as a PIC 9(12) COMP item.</p>
<name of exception category>	<p>A variable containing the name of the exception. If the name cannot fit into the variable, the name is truncated. For example, if the variable holds only five characters, the category OPENEROR is stored as OPENE. If the name does not fill the variable, trailing blanks are added.</p> <p>If no exception has occurred, the variable contains the text <i>NO EXCEPTION</i>.</p>

Table 5-4 gives the data types for the parameters and the exported names of the entry points.

Table 5-4. Identifying the Type of Exception

Exported Name of Entry Point	Parameter	Data Type of Parameter
ALGOEXCEPTIONNAME	<exception>	BOOLEAN
	<name>	STRING
DBEXCEPTIONNAME	<exception>	COMP-2 (COBOL) COMP (COBOL74 and COBOL85)
	<name>	DISPLAY
FORTRAN77EXCEPTIONNAME	<exception>	LOGICAL
	<name>	CHARACTER

**Program Examples**

The following program fragments use an exception name operation to obtain the type of exception.

**ALGOL Program Fragment**

```

%Declaring library entry point

LIBRARY DMI;
BOOLEAN PROCEDURE DBEXCEPTIONNAME(EXCEPTION,CATEGORY);
  VALUE EXCEPTION;
  BOOLEAN EXCEPTION;
  STRING CATEGORY;
  LIBRARY DMI (ACTUALNAME="ALGOEXCEPTIONNAME");

%Exception handling procedure

PROCEDURE EXCEPTIONHANDLER;
BEGIN
  STRING EXCEPTIONMSG;
  BOOLEAN STATUSRSLT;

  STATUSRSLT:=DBSTATUS;
  DBEXCEPTIONNAME(STATUSRSLT,EXCEPTIONMSG);
  WRITE(REMOTEFILE,<EXCEPTION_NAME="",A14>,EXCEPTIONMSG>);
END OF EXCEPTIONHANDLER;

```

## Identifying the Type of Exception (cont.)

---

### COBOL, COBOL74, or COBOL85 Program Fragment

\*\*\*Declaring variables for COBOL

```
Ø1 EXCEPTION-WORD      PIC 9(12) COMP-2.  
Ø1 EXCEPTION-NAME     PIC X(14).
```

\*\*\*Declaring variables for COBOL74 and COBOL85

```
Ø1 EXCEPTION-WORD      PIC 9(12) COMP.  
Ø1 EXCEPTION-NAME     PIC X(14).
```

\*\*\*Invoking entry point

```
CALL "DBSTATUS OF DMINTERPRETER"  
    USING EXCEPTION-WORD.  
CALL "DBEXCEPTIONNAME OF DMINTERPRETER"  
    USING EXCEPTION-WORD, EXCEPTION-NAME.  
MOVE EXCEPTION-NAME TO REMOTE-REC.  
WRITE REMOTE-REC.
```

### FORTRAN77 Program Fragment

\*\*\*Declaring library entry point

```
SUBROUTINE DBEXCN (EXCEPT, TEXT)  
  LOGICAL EXCEPT  
  CHARACTER NAME  
  IN LIBRARY DMI (ACTUALNAME = "FORTRAN77EXCEPTIONNAME")  
END
```

\*\*\*Exception handling subroutine

```
  SUBROUTINE EXCEPT(RESULT)  
  
  $INCLUDE "DATABASE/DMINTERPRETER" 2Ø29ØØØØ-2Ø299999  
  
  CHARACTER    MSG    *7Ø  
  LOGICAL      RESULT  
  
  RESULT = DBSTAT()  
  
  CALL DBEXCN(RESULT,MSG)  
  WRITE(6,2ØØ)MSG  
  
  CALL DBEXCT(RESULT,MSG)  
  WRITE(6,2ØØ)MSG  
2ØØ FORMAT(1X,A)  
  RETURN  
  END
```

# Section 6

## Restricting Calls to the Accessroutines

This section details the attribute setting entry point you use to restrict the number of times the DMINTERPRETER library can call the Accessroutines to complete a find, lock, or secure operation.

The discussion includes the following:

- A brief explanation of the function of the entry point
- An explanation of the required parameters

Read Section 2, “Accessing the Interpretive Interface,” for an explanation of how each language accesses the interpretive interface and invokes an entry point.

### Passing Parameters

Each language has only one attribute setting entry point. (COBOL, COBOL74, and COBOL85 have the same requirements.) The entry point has two parameters: **limit type** and **limit value**.

Parameter	Explanation
<limit type>	<p>A string parameter that specifies the type of limit to be set (what operation to restrict).</p> <p><b>Note:</b> <i>The only valid limit type is FINDLIMIT. It restricts find, lock, and secure operations and prevents long linear searches. Both record and structure find, lock, and secure operations are restricted.</i></p>
<limit value>	<p>An integer string parameter that specifies the maximum number of times the DMINTERPRETER can call the Accessroutines to perform a find, lock, or secure operation. The limit value has two restrictions:</p> <ul style="list-style-type: none"><li>• When the DMINTERPRETER is first initialized, FINDLIMIT is set to the default value of 0 (zero). A zero value disables the FINDLIMIT mechanism, permitting a linear search of the entire data set.</li><li>• Negative values are not allowed.</li></ul>

Table 6-1 gives the exported names of the entry points, as well as the parameters and the data type of each parameter.

## Restricting Calls to the Accessroutines

---

Table 6-1. Attribute Setting Entry Points—Restricting Calls to Accessroutines

Exported Name of Entry Point	Parameter	Data Type of Parameter
ALGOLSETLIMIT	<limit type>	STRING
	<limit value>	STRING
DBSETLIMIT	<limit type>	DISPLAY
	<limit value>	DISPLAY
FORTRAN77SETLIMIT	<limit type>	CHARACTER
	<limit value>	CHARACTER

**Note:**

- *If you specify an invalid limit type, the Accessroutines returns INTLIBERROR 33.*
- *If the DMINTERPRETER library exceeds the limit value, the Accessroutines returns INTLIBERROR 35.*

Refer to Appendix B, "DMSII Exceptions and Errors," for more information on INTLIBERROR errors.

### Program Examples

The following ALGOL, COBOL, and FORTRAN77 program fragments limit find, lock, and secure operations (on both records and structures), overriding the system default.

In each program fragment the limit type is FINDLIMIT and the limit value is 9.

### ALGOL Program Fragment

```
%Declaring local variables

STRING
  LIMIT_TYPE, LIMIT_VALUE;
BOOLEAN
  RSLT;

%Assigning values to parameters

LIMIT_TYPE:="FINDLIMIT";
LIMIT_VALUE:="9";

%Declaring library entry point

LIBRARY DMI;
BOOLEAN PROCEDURE DBSETLIMIT(LIMITTYPE, LIMITVALUE)
  STRING LIMITTYPE, LIMITVALUE;
  LIBRARY DMI (ACTUALNAME="ALGOLSETLIMIT");

%Invoking entry point

RSLT:=DBSETLIMIT(LIMIT_TYPE, LIMIT_VALUE);
IF RSLT
  THEN EXCEPTIONHANDLER;
```

### COBOL, COBOL74, or COBOL85 Program Fragment

```
***Declaring variables

Ø1 LIMIT-TYPE          PIC X(17) VALUE IS "FINDLIMIT".
Ø1 LIMIT-VALUE        PIC X(17) VALUER IS SPACES.
Ø1 RESULT              PIC 9(1) COMP-1 VALUE IS Ø.

***Invoking entry point

MOVE "9" TO LIMIT-VALUE.
CALL "DBSETLIMIT OF DMINTERPRETER"
  USING LIMIT-TYPE, LIMIT-VALUE
  GIVING RESULT.
```



## Restricting Calls to the Accessroutines

---

### FORTRAN77 Program Fragment

\*\*\*Declaring library entry point

```
LOGICAL FUNCTION DBSETLIMIT (LTT, LTV)
CHARACTER LTT, LTV
IN LIBRARY DMI (ACTUALNAME="FORTRAN77SETTLIMIT")
```

\*\*\*Setting variables to initial values

```
CHARACTER LLT      *9      /"FINDLIMIT"/
                LTV      *2      /"9"/
```

\*\*\*Declaring local variables

```
LOGICAL  RSLT
```

\*\*\*Invoking entry point

```
RSLT = DBSETL(LTT,LTV)
IF (RSLT) CALL EXCEPT
```

# Section 7

## Determining the Database Structure

The DMINTERPRETER library contains an intrinsic, read-only data set named DBSTRUCTURE. This data set holds structural information about the database that allows your application program to do the following:

- Determine the database structure at run time.
- Use general-purpose routines that can access any database and yet be data-independent.

### Generating the DBSTRUCTURE Data Set

As one of its options, the program BUILDINQ can automatically generate the DBSTRUCTURE data set when the DMINTERPRETER library is compiled. If the data set is generated, it need not be declared in the DASDL description of the database.

**Notes:**

- *If the DASDL description of the database contains a user-declared data set named DBSTRUCTURE, BUILDINQ cannot generate the intrinsic data set DBSTRUCTURE.*
- *If you are using A Series Extended Retrieval with Graphic Output (ERGO), you must generate the intrinsic data set DBSTRUCTURE.*

For more information on how to use BUILDINQ to generate the data set, refer to Appendix A, "Generating the DMINTERPRETER Library."

### Accessing the DBSTRUCTURE Data Set

Unlike other data sets, your application program can access the intrinsic DBSTRUCTURE data set before opening the database. However, at all times, only three operations can be performed on the DBSTRUCTURE data set: find, get, and set.

To access items in the DBSTRUCTURE data set, use a find operation entry point. For example, the following call to DBFIND locates the next required item in DBSTRUCTURE:

```
DBFIND ("NEXT", "DBSTRUCTURE", "ITEM-REQUIRED")
```

# Describing the Structure and Contents of DBSTRUCTURE

DBSTRUCTURE contains an entry for each data set, set, subset, access, item, and link in the database. No sets or subsets span DBSTRUCTURE.

DBSTRUCTURE behaves as if it had the following DASDL description:

```
DBSTRUCTURE DATA SET

(DB-ID           FIELD(16)      REQUIRED;
DB-OWNER        FIELD(16)      REQUIRED;
DB-NAME         ALPHA(17)      REQUIRED;
DB-TYPE         ALPHA(17)      REQUIRED;

% ITEMS PRESENT WHEN DB-TYPE = "DATASET"

DATASET-SUBTYPE ALPHA(17);

% ITEMS PRESENT WHEN DB-TYPE = "SET"

SET-SUBTYPE     ALPHA(17);
SET-ACCESS-METHOD ALPHA(17);
SET-SPANS-DATASET NUMBER(4);

% ITEMS PRESENT WHEN DB-TYPE = "ITEM"

ITEM-SUBTYPE    ALPHA(17);
ITEM-SUBSCRIPTS NUMBER(2);
ITEM-OCCURS     NUMBER(4);
ITEM-OWNER-GROUP NUMBER(4);
ITEM-REQUIRED   BOOLEAN;
ITEM-SIZE       NUMBER(4);
ITEM-SCALE-FACTOR NUMBER(2);
ITEM-SIGNED     BOOLEAN;
ITEM-KEY-CLASS  ALPHA(17);
ITEM-RECORD-TYPE NUMBER(3);

% ITEMS PRESENT WHEN DB-TYPE = "LINK"

LINK-TO-DATASET NUMBER(4);
LINK-OCCURS     NUMBER(4);
);
```

As described in Table 7-1, the first four items in the record (DB-ID, DB-OWNER, DB-NAME, and DB-TYPE) uniquely identify the database. These items always have nonnull values; they are required items. When a record is stored in the data set, the system tests all required items. If a required item contains a null value, the storing program receives an exception and does not store the record. You can use the required option to ensure that all data items, with the exception of Boolean items, are assigned a nonnull value.

**Table 7-1. DBSTRUCTURE Items**

Item	Description
<b>DB-ID</b>	Contains a positive number that uniquely identifies each component in the database. DB-ID allows records in DBSTRUCTURE to be associated with each other. Items in more than one variable record type are considered to be different entities, each with a different DB-ID.
<b>DB-OWNER</b>	Contains the DB-ID of the data set that owns this entry. DB-OWNER is 0 (zero) for the global data entry. All disjoint structures and global data items have the DB-ID for global data.
<b>DB-NAME</b>	Contains the name of the entity. Trailing blanks are added, if needed.
<b>DB-TYPE</b>	Contains a string describing the type of the entity. Valid values and their explanations are as follows: <ul style="list-style-type: none"> <li>• DATASET. Specifies a data set.</li> <li>• SET. Specifies a set, subset, or access.</li> <li>• ITEM. Specifies a data item.</li> <li>• LINK. Specifies a link.</li> </ul>

Entities are assigned nonnull and null values as follows:

- If the DB-TYPE is DATASET, the item DATASET-SUBTYPE has a nonnull value. All SET, ITEM, and LINK entities are null.
- If the DB-TYPE is SET, the items SET-SUBTYPE, SET-ACCESS-METHOD, and SET-SPANS-DATASET have nonnull values. All DATASET, ITEM, and LINK entities are null.
- If the DB-TYPE is ITEM, all items beginning with "ITEM-" have nonnull values. All DATASET, SET, and LINK entities are null.
- If the DB-TYPE is LINK, the items LINK-TO-DATASET and LINK-OCCURS have nonnull values. All DATASET, SET, and ITEM entities are null.

**Notes:**

- *All sets with group keys have the following two entries in the DBSTRUCTURE intrinsic data set:*
  - *An entry that lists the group as its key*
  - *An entry that lists the elementary items as its keys*
- *Each database also has an entry for global data. This entry can be used to find all the disjoint structures and global data items in the database.*

## Determining the Database Structure

---

Tables 7-2 through 7-5 describe the items associated with each value of DB-TYPE.

- The items for DATASET are described in Table 7-2.
- The items for SET are explained in Table 7-3.
- The items for ITEM are shown in Table 7-4.
- The items for LINK are given in Table 7-5.

**Table 7-2. DBSTRUCTURE Items for a DB-TYPE of DATASET**

Item	Description
<b>DATASET-SUBTYPE</b>	<p data-bbox="639 747 1328 804">Contains a string describing the type of a data set. Valid values and their explanation are as follows:</p> <ul data-bbox="639 825 1328 1434" style="list-style-type: none"><li>• <b>STANDARD.</b> Identifies a data set that stores new records in areas vacated by old records.</li><li>• <b>RANDOM.</b> Identifies a data set that uses a function on an Access to retrieve records.</li><li>• <b>ORDERED.</b> Identifies a data set that orders records in the sequence defined by the alphanumeric key.</li><li>• <b>UNORDERED.</b> Identifies a data set that stores records in an unpredictable order.</li><li>• <b>GLOBAL.</b> Identifies a data set that locates all disjoint structures and global data items in the database.</li><li>• <b>DIRECT.</b> Identifies a data set that uses an index known as an access.</li><li>• <b>RESTART.</b> Identifies a data set that the programs access during recovery.</li><li>• <b>COMPACT.</b> Identifies a data set that removes null items and trailing blanks from alpha items before storing records on disk.</li><li>• <b>STRUCTURE.</b> Identifies the DBSTRUCTURE itself.</li></ul>

The different types of data sets are detailed in the *DMSII Technical Overview*.

Table 7-3. DBSTRUCTURE Items for a DB-TYPE of SET

Item	Description
<b>SET-SUBTYPE</b>	<p>Contains a string describing the type of a set. Valid values and their explanations are as follows:</p> <ul style="list-style-type: none"> <li>● SET. Identifies a structure that permits logical sequence access. It contains one entry for each record in the data set. The set can be disjoint or embedded. Records are indexed through a key in the data set.</li> <li>● SUBSET. Identifies a structure similar to a set, except it only indexes certain members of data set. A subset can be automatic or manual.</li> <li>● ACCESS. Identifies a method of retrieving records from direct, random, and ordered data sets in the order defined by a key. Accesses do not physically exist on disk.</li> </ul>
<b>SET-SEARCH-METHOD</b>	<p>Contains a string describing the accessing method used by a set. Valid values and their explanations are as follows:</p> <ul style="list-style-type: none"> <li>● DIRECT. Specifies direct data set accesses.</li> <li>● BINARY. Specifies ordered data set accesses, and index sequential and ordered list sets.</li> <li>● HASH. Specifies random data set accesses and index random sets.</li> <li>● LINEAR. Specifies unordered data set accesses and bit vector sets.</li> </ul>
<b>SET-SPANS-DATASET</b>	<p>Contains the DB-ID of the data set spanned by a set.</p>

Consult the *DMSII Technical Overview* for an explanation of types of sets. Read the *DMSII Application Programming Guide* for an explanation of search methods and techniques.

**Table 7-4. DBSTRUCTURE Items for a DB-TYPE of ITEM**

Item	Description
<b>ITEM-SUBTYPE</b>	<p>Contains a string describing the type of an item. Valid values and their explanations are as follows:</p> <ul style="list-style-type: none"> <li>• ALPHA. Identifies words and characters stored as EBCDIC characters.</li> <li>• GROUP. Identifies a collection of related items. The items can be alphabetic or numeric items.</li> <li>• FIELD. Identifies from 1 to 48 Boolean values or an unsigned, nonnegative integer up to 48 bits long.</li> <li>• NUMBER. Identifies integers and fractions with or without signs.</li> <li>• REAL. Identifies single precision floating-point numbers.</li> <li>• BOOLEAN. Identifies true or false values.</li> <li>• COUNT. Identifies a value that tracks the number of links pointing to the record.</li> <li>• TYPE. Identifies the record type of the current record.</li> <li>• POPULATION. Identifies a value that estimates the expected size of the data set.</li> </ul>
<b>ITEM-SUBSCRIPTS</b>	<p>Contains the number of subscripts required when the item is used. If no subscripts are required, ITEM-SUBSCRIPTS is 0 (zero).</p>
<b>ITEM-OCCURS</b>	<p>Stores the subscript limit for an item declared with an OCCURS clause in the DASDL description.</p> <p><b>Notes:</b></p> <ul style="list-style-type: none"> <li>• <i>If ITEM-SUBSCRIPTS is 0 (zero), ITEM-OCCURS contains a null value.</i></li> <li>• <i>Items that have a nonzero value in ITEM-SUBSCRIPTS and a null value in ITEM-OCCURS are nonoccurring items.</i></li> <li>• <i>To find the other subscript limits for multiple-subscripted items, inspect the ITEM-OCCURS entries in which DB-ID equals ITEM-OWNER-GROUP.</i></li> </ul>

continued

Table 7-4. DBSTRUCTURE Items for a DB-TYPE of ITEM (cont.)

Item	Description
<b>ITEM-OWNER-GROUP</b>	<p>Contains the DB-ID of the next higher group or field that contains an item. The DB-ID can be used to determine</p> <ul style="list-style-type: none"> <li>• The index limit for items in occurring groups and fields.</li> <li>• The items that belong to a given group or field.</li> </ul> <p>If an item does not belong to a group or field, ITEM-OWNER-GROUP is 0.</p>
<b>ITEM-REQUIRED</b>	<p>Contains the value TRUE if the entity is a required item. Otherwise, the value is FALSE.</p>
<b>ITEM-SIZE</b>	<p>Contains the declared length of a number, field, alpha, or group item in hexadecimal digits, bits, EBCDIC bytes, and bytes, respectively.</p> <p><i>Note: Ordinarily, ITEM-SIZE does not include any spaces for signed values. However, it does include spaces for group items containing signed number items.</i></p>
<b>ITEM-SCALE-FACTOR</b>	<p>Contains the declared number of digits to the right of the decimal point for number items and real items that have a scale factor declared in the DASDL description.</p> <p>The value of a number cannot exceed the following: (The symbol ** is used to represent the exponent.)</p> $10^{**}[(\text{ITEM-SIZE}) - (\text{ITEM-SCALE-FACTOR})]$ <p>Unless the real value is declared without precision, the real value must be less than the following:</p> $549755813887.5/10^{**}(\text{ITEM-SCALE-FACTOR})$
<b>ITEM-SIGNED</b>	<p>Contains the value TRUE for a number item or real item if the item can be negative (in other words, declared as signed in the DASDL description). It contains the value FALSE for a number or real item that cannot be signed.</p>

continued



## Determining the Database Structure

---

Table 7-4. DBSTRUCTURE Items for a DB-TYPE of ITEM (cont.)

Item	Description
ITEM-KEY-CLASS	Contains a string indicating the key status of an item in a spanning set. Valid entries and their explanations are as follows: <ul style="list-style-type: none"><li>• MAJOR. Identifies the item as the major key of a spanning set or access. (The major key is defined as the first key listed in the declaration of the set.)</li><li>• MINOR. Identifies the items as a key of the set. The item is not the major key of a spanning set or access.</li><li>• DATA. Specifies that the item appears only as key data in spanning sets or accesses.</li><li>• NONE. Specifies that the item does not appear in the key area of any access or spanning set.</li></ul>
ITEM-RECORD-TYPE	Contains the record type of an item. The ITEM-RECORD-TYPE is 0 (zero) for items in fixed-format records or in the fixed part of variable-format records.  <i>Note: Because an item with a particular DB-ID can only be in records of one type, any other item of the same name in a different record type has a different DB-ID.</i>
ITEM-USAGE	Contains a string describing the usage of an item. Valid values are EBCDIC and KANJI.

Consult the *DMSII Technical Overview* for information on types of items, the OCCURS clause, key status, key items, and record types.

Table 7-5. DBSTRUCTURE Items for a DB-TYPE of LINK

Item	Description
LINK-TO-DATASET	Contains the DB-ID of the data set to which a link points.
LINK-OCCURS	Contains the limit of the OCCURS clause for a link. If LINK-OCCURS is 0 (zero), no OCCURS clause was present in the declaration of the link.

The *DMSII Technical Overview* contains detailed information about links.

# Appendix A

## Generating the DMINTERPRETER Library

To use the interpretive interface, each database must have its own custom-tailored DMINTERPRETER library. You can use the program BUILDINQ to generate the library interactively or from a Work Flow Language (WFL) job deck.

Usually, a DMINTERPRETER library is generated only once – after the DASDL source code for the database is compiled. However, if there is a directory-level mismatch or if you use the REORGANIZATION program to make physical changes to the database description, you need to generate the DMINTERPRETER library again. (Consult the *DMSII Utilities Operations Guide* for specifics of the REORGANIZATION program.)

### Capabilities of the BUILDINQ Program

The BUILDINQ program does the following:

- Associates a cycle, version, and timestamp with a copy of each generated file
- Sets either the DASDL option ZIP or the RECOVER option NOZIP
- Allows your application program to interface with
  - An entire physical database
  - Selected parts of a database
  - A logical database
  - A maximum of 255 structures plus global data
- Specifies if the database can be compiled for inquiry only or for inquiry and update

### Files Associated with a DMINTERPRETER Library

Table A-1 lists the files associated with a DMINTERPRETER library.

## Generating the DMINTERPRETER Library

---

Table A-1. Files Associated with a DMINTERPRETER Library

File Name	Use
<b>SYMBOL/BUILDING</b>	Symbolic file of the program compiled to produce OBJECT/BUILDING.  SYMBOL/BUILDING can be compiled using DMALGOL by equating the DMALGOL file TAPE to the file SYMBOL/BUILDING.
<b>SYSTEM/BUILDING</b>	Object code file for SYMBOL/BUILDING.  SYSTEM/BUILDING builds a directory for the portion of the database that DMINTERPRETER can access. The directory, named DMIDIRECTORY/<database name>, is compiled and included in the DMINTERPRETER code file for run-time interpretation of the database structure.
<b>DATABASE/DMINTERPRETER</b>	Symbolic file for DMINTERPRETER.  DATABASE/PROPERTIES and DMIDIRECTORY/<database name> are included in this symbolic file.
<b>SYSTEM/DMINTERFACE</b>	An external coroutine, called by the compilers, to extract and check information from DESCRIPTION/<database name>.
<b>DATABASE/PROPERTIES</b>	Each statement in this file defines some aspect of the database. The file is included in both the DASDL compilation and DATABASE/DMINTERPRETER.
<b>DESCRIPTION/&lt;database name&gt;</b>	File containing a complete description of the database in encoded form. This file is the output from a DASDL compilation and is used by SYSTEM/DMINTERFACE.
<b>DMINTERPRETER/&lt;database name&gt;</b>	The generated DMINTERPRETER code file.
<b>DMIDIRECTORY/&lt;database name&gt;</b>	The intermediate compiler input file. It is produced by SYSTEM/BUILDING and included by DATABASE/DMINTERPRETER.

Figure A-1 illustrates the relationships among the files.

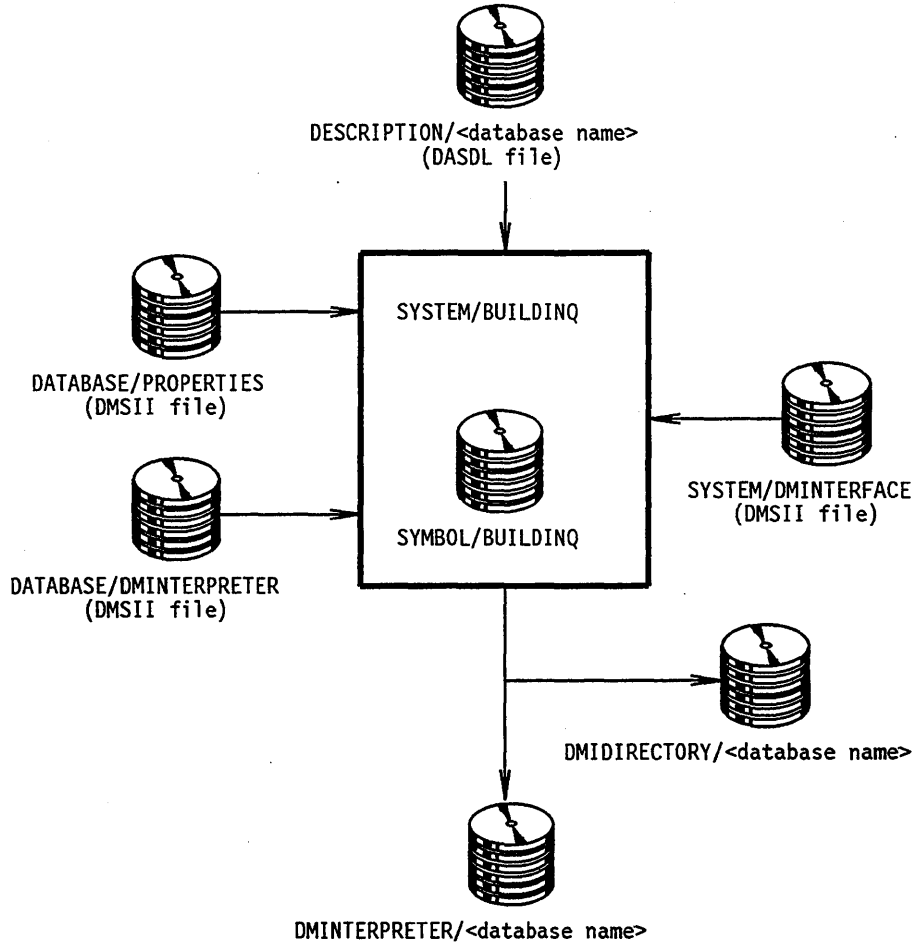


Figure A-1. Relationship of BUILDINGQ Files

### Ensuring Directory-Level Consistency between Files

In order for you to use the interpretive interface, the directory level of the intermediate compiler input file DMIDIRECTORY/< database name > and the directory level of the symbolic program file DATABASE/DMINTERPRETER must be the same.

During compilation of the interpretive interface, the directory levels are checked to ensure consistency. If a level mismatch is found, the error count for the compiler increases by 1, and the following error message is displayed:

```
DMIDIRECTORY LEVEL MISMATCH
```

You cannot successfully generate the DMINTERPRETER library with a directory-level mismatch. To correct the problem, rerun the BUILDINQ program to create a new DMIDIRECTORY file with the correct directory level.

### Generating the DMINTERPRETER Library Interactively

You can use the BUILDINQ program to interactively generate a DMINTERPRETER library from a remote terminal and to set the RECOVER option NOZIP.

By default, the DASDL option ZIP is set. The ZIP option causes automatic compilation of the DMINTERPRETER program; the NOZIP option inhibits automatic compilation. When the NOZIP option is set, BUILDINQ builds a Work Flow Language (WFL) deck for compiling DMINTERPRETER. Use the NOZIP option if you want to set any of the following:

- Class specification
- Family specification
- Usercode specification
- Fetch specification
- Job attribute assignment

The procedure to use the default of ZIP and the procedure to set NOZIP are basically the same. The procedure for using the default is explained in "Remote Generation Using the ZIP Option." The differing steps needed to set the NOZIP option are explained in "Remote Generation Using the NOZIP Option."

**Note:** *All input to the BUILDINQ program must be entered in uppercase letters. If you use any lowercase letters, you are reprompted.*

### Remote Generation Using the ZIP Option

Use the following basic procedure to execute BUILDINQ and to use the default ZIP option:

1. Initiate the BUILDINQ program.
2. Select the database.
3. Select a view of the database.
4. If applicable, select data sets.
5. If applicable, enter the name of a logical database.
6. Determine the compiled access mode for the DMINTERPRETER library.
7. If applicable, rename hyphenated logical database name.
8. Enter a database name for the DMINTERPRETER code file.
9. Enter compilation queue.
10. Check for verification message.

These 10 steps are detailed in the following text, including an explanation of when they apply.

Consult the *DMSII DASDL Reference Manual* and the *DMSII Utilities Operations Guide* for more information about the ZIP option.

#### Step 1. Initiating the BUILDINQ Program

Begin executing the BUILDINQ program using the RUN command. (RUN can be abbreviated as R or RU.)

```
R BUILDINQ
```

#### Step 2. Selecting the Database

Each database must have its own, custom DMINTERPRETER library. At the following prompt, enter the name of the database:

```
WHAT DATABASE?
```

**Note:** To use the NOZIP option, follow the steps shown in "Remote Generation Using the NOZIP Option" in this section before proceeding to Step 3.

## Generating the DMINTERPRETER Library

---

### Step 3. Selecting a View of the Database

You can generate the DMINTERPRETER library for the physical database, disjoint data sets, or a logical database. At the following prompt, enter the number for the database option you want:

```
WHICH OPTION?  
1 TOTAL DATABASE  
2 SELECTED DATASETS  
3 LOGICAL DATABASE
```

*Note: If no logical databases are declared for the specified database, option 3 is not displayed.*

As shown in the following table, your entry determines the subsequent sequence of prompts:

Option	Result
1	Accesses the physical database—all database structures, including the intrinsic data set DBSTRUCTURE.  The program skips steps 4 and 5; it resumes at “Step 6. Determining the Compiled Access Mode for the DMINTERPRETER.” See Figure A-2.
2	Accesses only those disjoint data sets that are interactively selected.  The program continues at “Step 4. Selecting Data Sets.” See Figure A-3.
3	Accesses only a logical database. The logical database represents only parts of the database to be used for limited purposes.  The program skips step 4 and continues at “Step 5. Entering the Name of a Logical Database.” See Figure A-4.

The following three flowcharts, A-2, A-3, and A-4, illustrate how the sequence of steps differs, depending on your choice.

Figure A-2 is a flowchart for compiling a DMINTERPRETER library for the total database.

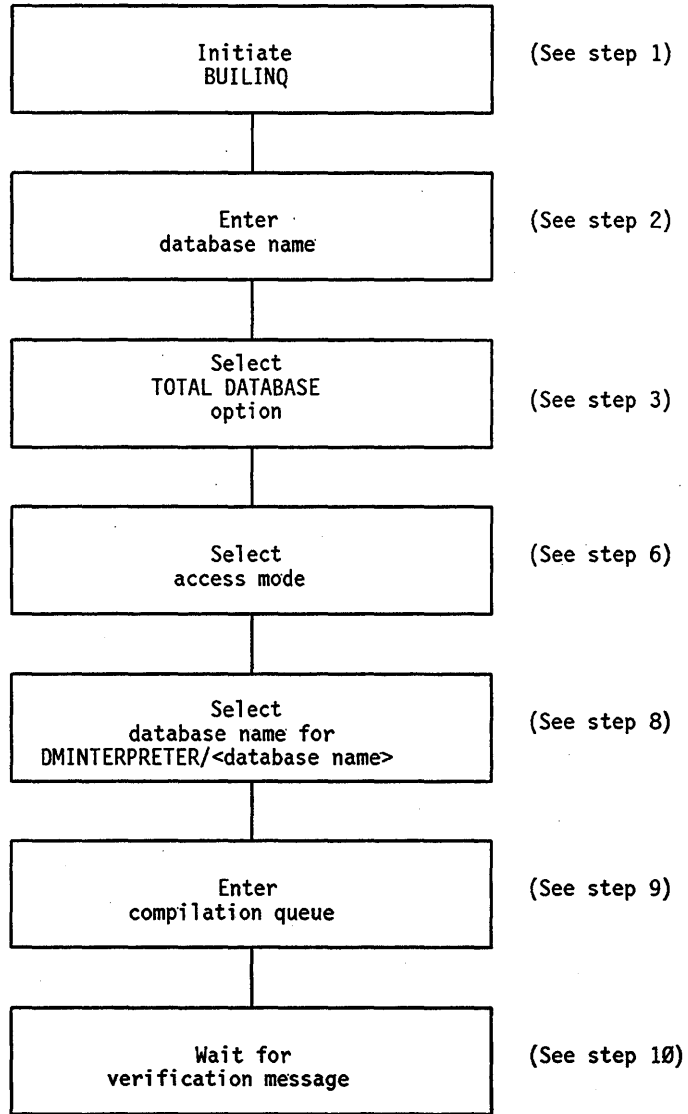


Figure A-2. Compiling a DMINTERPRETER Library for the Total Database



## Generating the DMINTERPRETER Library

---

Figure A-3 is a flowchart for compiling a DMINTERPRETER library for selected data sets.

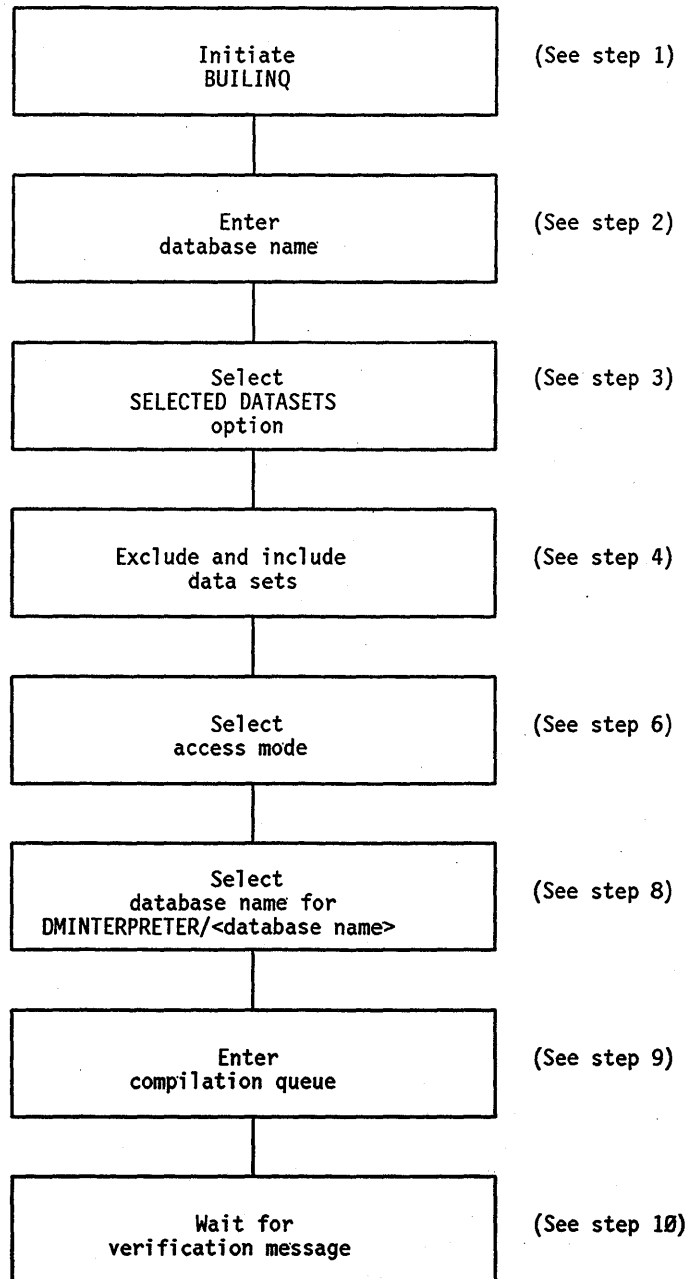


Figure A-3. Compiling a DMINTERPRETER Library for Selected Data Sets

## Generating the DMINTERPRETER Library

Figure A-4 is a flowchart for compiling a DMINTERPRETER library for a logical database.

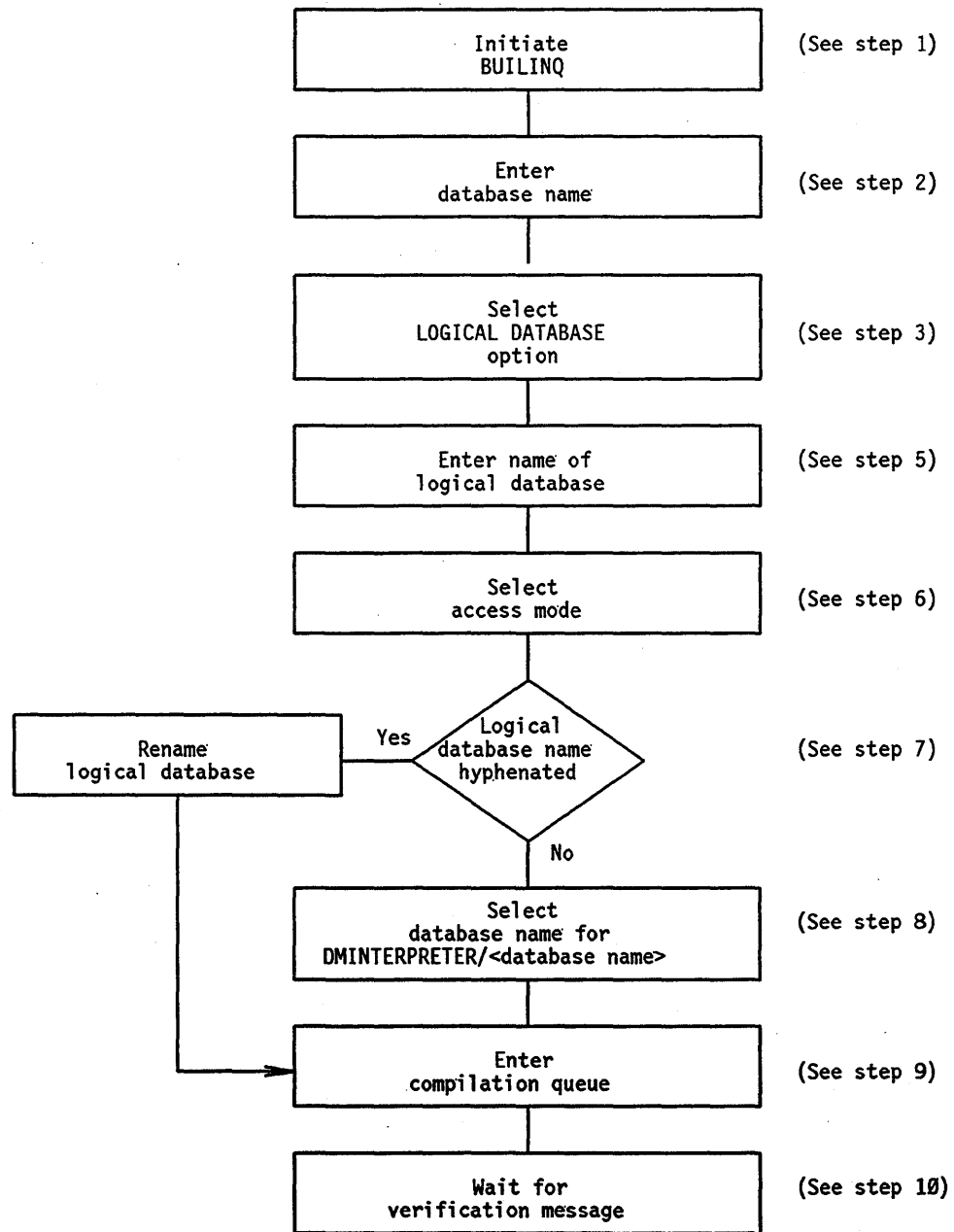


Figure A-4. Compiling a DMINTERPRETER Library for a Logical Database

## Generating the DMINTERPRETER Library

---

### Step 4. Selecting Data Sets

If you choose to generate a DMINTERPRETER library for selected data sets, BUILDINQ displays the name of each disjoint data set, one at a time. After each data set name is shown, BUILDINQ displays the following prompt:

```
FOR FOLLOWING DATASETS ENTER:  
  
E (OR EMPTY INPUT) TO EXCLUDE FROM SOFTWARE  
  
I TO INCLUDE INTO SOFTWARE  
  
C TO INDICATE DONE  
  
* TO TERMINATE
```

By entering E or I, you decide whether to exclude or include each disjoint data set.

By entering C or an asterisk (\*), you signal the BUILDINQ that you want to terminate the run.

- To terminate the run and generate a DMINTERPRETER library for the included data sets, as well as deciding if you want to include the intrinsic data set DBSTRUCTURE, enter a C.
- To terminate the run without generating any DMINTERPRETER library, enter the asterisk.

Option	Result
E	Excludes the data set.
null entry	Excludes the data set.
I	Includes the data set.
C	Indicates that all desired disjoint data sets have been included. BUILDINQ then asks if you want the intrinsic data set DBSTRUCTURE included with the selected data sets. Enter <i>YES</i> or <i>NO</i> .
*	Terminates the run without generating a DMINTERPRETER library.

**Note:** *DBSTRUCTURE is an optional intrinsic data set that hold structural information about the database. DBSTRUCTURE can be accessed before the database is opened.*

*If you use Extended Retrieval with Graphic Output (ERGO) on the database, you must explicitly include the DBSTRUCTURE data set using the I option.*

*For more information about the DBSTRUCTURE data set, read Section 7, "Determining the Database Structure."*

Once you have selected the data sets and decided whether to include the DBSTRUCTURE data set, BUILDINQ skips step 5 and continues at step 6.

### Step 5. Entering the Name of a Logical Database

At step 3 you can narrow the access to a logical database only. The logical database must be identified by name. At the following prompt, enter the name of the logical database:

LOGICAL DATABASE NAME?

*Note: If the logical database contains an embedded set that is invoked more than once, a remap error can occur.*

### Step 6. Determining the Compiled Access Mode for the DMINTERPRETER Library

A DMINTERPRETER library can be compiled with inquiry-only access (read-only) or with both inquiry and update access. At the following prompt, enter the access mode you want for the DMINTERPRETER library:

INQUIRY ONLY (YES OR NO)?

The following table gives the result of each entry:

Option	Result
YES	Programs that access the database cannot create, delete, or update records; the programs can report data.
NO	Programs that access the database can create, delete, update, or report data.

*Note: If you compile a DMINTERPRETER library with an inquiry-only access, any attempt by a program to open a database with an OPEN UPDATE command results in an UNKNOWN OPEN TYPE error. Attempts to create, delete, or update records result in READONLY errors.*

If your logical database name (as entered in step 5) has no hyphen (-), BUILDINQ goes to step 8 and then step 9. If your database name has a hyphen, BUILDINQ proceeds to step 8, where you identify the interpreter library, and then proceeds to step 9.

### Step 7. Renaming Hyphenated Logical Database Names

The BUILDINQ program uses the name of the database as part of the name for the DMINTERPRETER code file DMINTERPRETER/< database name >. The database name cannot contain any hyphens. Therefore, if the logical database name entered at step 4 contains a hyphen, BUILDINQ displays the following prompt:

INTERPRETER NAME?

Enter a name consisting of from 1 to 11 identifiers separated by slashes. Each identifier can contain from 1 to 17 alphanumeric characters only. For example, the database ACCOUNTING/PAYROLL 5 has two identifiers: ACCOUNTING and PAYROLL5. Each identifier is separated by a slash. Each identifier is less than 17 alphanumeric characters.

## Generating the DMINTERPRETER Library

---

*Note: If a file with the same name already exists, it is replaced by the newly compiled file.*

BUILDINQ then proceeds to step 9.

### Step 8. Entering a Database Name for the DMINTERPRETER Code File

At the following prompt, enter the database name to be used in creating the generated DMINTERPRETER code file DMINTERPRETER/< database name > :

INTERPRETER NAME (NULL FOR DEFAULT)?

You can either enter an unhyphenated name or use the default – the name you entered in step 2. If you enter a name, the name can consist of from 1 to 11 identifiers separated by slashes. Each identifier can contain 1 to 17 alphanumeric characters only. For example, the database BRANCHES/OVERSEAS/EUROPE has three identifiers: BRANCHES, OVERSEAS, and EUROPE. Each identifier is separated by a slash. Each identifier is less than 17 alphanumeric characters.

Entry	Result
<database name>	BUILDINQ uses this database name when it generates the code file DMINTERPRETER/<database name>.
null	BUILDINQ uses the <database name> you entered in step 2 when it generates the code file.

*Note: If a file with the same name already exists, it is replaced by the newly compiled file.*

### Step 9. Entering the Compilation Queue

At the following prompt, enter the number of the queue where the DMINTERPRETER library should be compiled:

WHAT QUEUE (NULL FOR DEFAULT)?

You can use the system default queue or select a specific queue:

Option	Result
null	BUILDINQ compiles DMINTERPRETER in the system default queue.
<queue number>	BUILDINQ compiles DMINTERPRETER in the specified queue.

### Step 10. Displaying the Verification Message

The BUILDINQ program displays the following message as it initiates the compilation of the DMINTERPRETER library:

```
# GENERATING DMINTERPRETER
```

## Remote Generation Using the NOZIP Option

You can elect to run BUILDINQ but inhibit the ZIP option by entering *\$NOZIP* at the prompt for a database name. Use the NOZIP option when you want to set any of the following:

- Class specification
- Family specification
- Usercode specification
- Fetch specification
- Job attribute assignment

### Selecting the NOZIP Option

The procedure for setting NOZIP is identical to the procedure for generating a DMINTERPRETER library with the ZIP option, with the following exceptions:

- You must respond twice to the *WHAT DATABASE* prompt: once to inhibit ZIP and once to name the database.
- When NOZIP is used, BUILDINQ does not prompt you for a queue; the default system queue is always used.
- BUILDINQ does not compile the DMINTERPRETER library; you must use a WFL job deck to compile the library.

The first prompt after you initiate the BUILDINQ program requests the name of the database. To inhibit ZIP, enter *\$NOZIP*. When the BUILDINQ program reprompts for the database name, enter the name of the database.

You begin the sequence by entering

```
R BUILDINQ
```

The rest of the sequence is as follows:

BUILDINQ Prompt	Your Entry
WHAT DATABASE?	<i>\$NOZIP</i>
WHAT DATABASE?	<database name>

BUILDINQ continues by requesting the view of the database. (See “Step 3. Selecting a View of the Database,” in this appendix.)

Figure A-5 shows the flow of steps when you use the NOZIP option.

## Generating the DMINTERPRETER Library

---

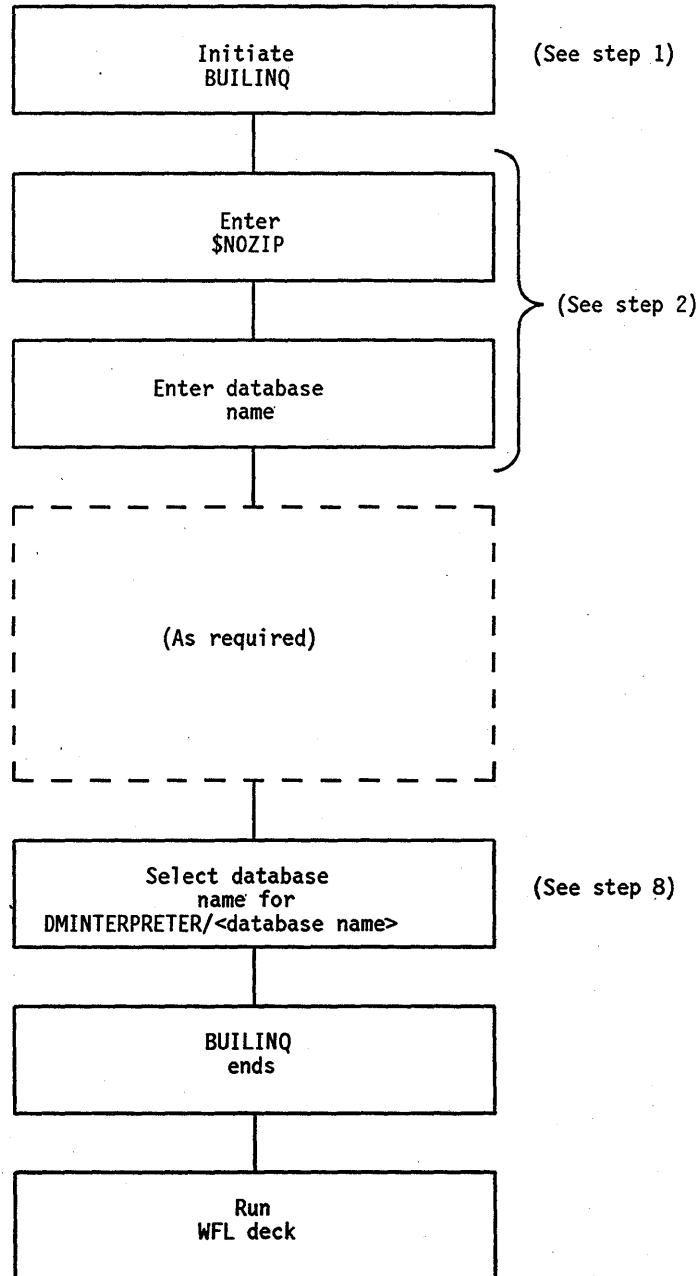


Figure A-5. Using the BUILDINQ Program with the NOZIP Option

### Compiling a DMINTERPRETER Library Where NOZIP Is Set

The BUILDINQ program cannot compile a DMINTERPRETER library where the NOZIP option is set; therefore BUILDINQ does not display a request for the compilation

queue. Instead, after you enter the database name for the DMINTERPRETER code file, BUILDINQ successfully terminates.

To compile a DMINTERPRETER library with the NOZIP option, you must use a WFL job deck that includes the following cards. Do not include a QUEUE statement; any queue information is ignored.

```

COMPILE DMINTERPRETER/<database name>
      WITH DMALGOL LIBRARY;
COMPILE FILE TAPE = DATABASE/DMINTERPRETER;
COMPILE FILE DASDL = DMIDIRECTORY/<database name>;
COMPILE FILE PROPERTIES = DATABASE/PROPERTIES;
      ? DATA
      $ MERGE
      ? END
    
```

*Note: The file-equation cards might require modification, depending on the location of the files being used.*

## Generating the DMINTERPRETER Library from WFL Job Decks

BUILDINQ can also be run entirely using a WFL job deck. The cards listed in Table A-2 can appear, in any order, in the WFL job.

**Table A-2. WFL Cards Used to Generate a DMINTERPRETER Library**

Card	Action
<b>DATABASE : &lt;database name&gt;</b>	<p>If this card is present, the database description file is found in a file titled DESCRIPTION/&lt;database name&gt;. BUILDINQ produces the DMIDIRECTORY but does not begin compilation of DMINTERPRETER.</p> <p>If this card is not present, the DASDL source file must be label-equated to the title of the database description file.</p>
<b>DMINTERPRETER</b>	This card must be present to specify the DMINTERPRETER.
<b>LDBNAME : &lt;database name&gt;</b>	If this card is present, access to the specified logical database is assumed; otherwise, access to the total database is assumed. The <database name> must refer to a logical database declared in DASDL.
<b>QUEUE : &lt;integer&gt;</b>	If this card is present, DMINTERPRETER is compiled in the specified queue; otherwise, it is compiled in the system default queue.

continued



## Generating the DMINTERPRETER Library

---

Table A-2. WFL Cards Used to Generate a DMINTERPRETER Library (cont.)

Card	Action
NOUPDATE	<p>If this card is present, records cannot be updated. There is no ability to create, delete, or update records; you only can inquire. If this card is not present, records can be created, deleted, and updated, and you also can inquire.</p> <p><i>Note: If you compile a DMINTERPRETER library with an INQUIRY only access, any attempt by a program to open a database with an OPEN UPDATE command results in an UNKNOWN OPEN TYPE error. Attempts to create, delete, or update records result in READONLY errors.</i></p>

The CARD input file for BUILDINQ can be made into a file of type DISK, REMOTE, or READER.

If the WFL compiler encounters an error when running the BUILDINQ job deck, WFL displays the appropriate error message and terminates BUILDINQ without completing the job.

Consult the *Task Attributes Reference Manual*, the *Task Management Guide*, and the *WFL Reference Manual* for more information about WFL and WFL job decks.

The following three examples show how to use these cards in a WFL job deck.

### Example 1: Basic Implementation

In the first WFL job, the DATABASE card specifies the database (MYDB) whose description file is used to build DMINTERPRETER. DMINTERPRETER is compiled from queue 40.

```
?BEGIN JOB BUILDINQFROMCARDS;  
RUN OBJECT/BUILDINQ;  
DATA  
DATABASE : MYDB  
DMINTERPRETER  
QUEUE : 40  
?END JOB.
```

### Example 2: Using a File Equation

The second WFL job uses a file equation to give the external name of the DASDL file (DESCRIPTION/MYDB) for the database whose description file is used to build DMINTERPRETER. Again, DMINTERPRETER is compiled from queue 40.

```
?BEGIN JOB BUILDINQFROMCARDS;
RUN OBJECT/BUILDINQ;
FILE DASDL (TITLE=DESCRIPTION/MYDB);
DATA
DMINTERPRETER
QUEUE : 40
?END JOB
```

### Example 3: Generating in a Batch Environment

The third WLF job shows how the interpretive interface can be generated in a batch environment for a database named DMIDB.

```
?BEGIN JOB CREATE/DMINTERPRETER;
TASK T;
REMOVE DMIDIRECTORY/DMIDB;
RUN OBJECT/BUILDINQ [T];
DATA
$NOZIP
DATABASE : DMIDB
DMINTERPRETER
? % --- END BUILDINQ INPUT ---
IF T ISNT COMPLETEDOK THEN
ABORT "BUILDINQ RUN FAILED";
COMPILE DMINTERPRETER/DMIDB WITH DMALGOL [T] LIBRARY;
COMPILER FILE DASDL (TITLE=DMIDIRECTORY/DMIDB);
COMPILER FILE CARD (KIND=PACK,
                    TITLE=DATABASE/DMINTERPRETER);
IF T ISNT COMPILEDOK THEN
ABORT "DMINTERPRETER COMPILE FAILED";
?END JOB.
```



# Appendix B

## DMSII Exceptions and Errors

The Access routines return exception and error messages to an application program when data management operations cannot be successfully or correctly completed.

- Exceptions are caused by problems within the application program. For example, an exception is returned if your application program tries to nest logical transactions.
- Errors are caused by problems within the database or DMSII software. For instance, an error is returned if there is an invalid index within the DMSII code.

Many responses to error messages require running DMSII utilities such as REORGANIZATION, DBCERTIFICATION, DMUTILITY, and REBUILD. For details about the DMSII utilities, consult the *DMSII Utilities Operations Guide*. Some responses require changes to the DASDL source file for your database. For in-depth information on DASDL, read the *DMSII DASDL Reference Manual*.

## Categorizing Exceptions and Errors

The exceptions and errors are divided into 21 major categories. Each major category has an associated category number and mnemonic. For example, the major category known by the mnemonic SYSTEMERROR is also known as major category 7. Table B-1 lists the numbers and mnemonics for the major categories by category number, in ascending order.

Table B-1. Major Categories for Exceptions and Errors by Category Number

Number	Mnemonic	Number	Mnemonic
1	NOTFOUND	11	OPENEROR
2	DUPLICATES	12	CLOSEERROR
3	DEADLOCK	13	NORECORD
4	DATAERROR	14	INUSE
5	NOTLOCKED	15	AUDITEROR
6	KEYCHANGED	16	ABORT
7	SYSTEMERROR	17	SECURITYERROR
8	READONLY	18	VERSIONERROR
9	IOERROR	19	FATALERROR
10	LIMITERROR	20	INTEGRITYERROR
		21	INTLIBERROR

*Note: Any message that can be returned through the language extensions can be returned through the interpretive interface. However, INTLIBERROR (category 21) messages can be returned only to application programs using the interpretive interface.*

Within each major category, each cause for the exception or error is a subcategory. Every subcategory has an associated subcategory number. Tables B-2 through B-21 list the major categories alphabetically by mnemonic. Under each category, the table lists the valid subcategories and their meaning.

## ABORT Category

The ABORT category is also known as major category 16. The most probable cause for an ABORT message is a database abort or a rollback.

Table B-2 lists the exceptions and errors subcategories associated with an ABORT message.

Table B-2. ABORT Subcategories

Subcategory Number	Meaning
1	The database was aborted and rolled back.
2	An abort occurred, and the Transaction Processing System (TPS) rerun was not finished.

## AUDITERROR Category

The AUDITERROR category is also known as major category 15. The most probable cause for an AUDITERROR message is a conflict in transaction state.

Table B-3 lists the exceptions and errors subcategories associated with an AUDITERROR message.

Table B-3. AUDITERROR Subcategories

Subcategory Number	Meaning
1	The program attempted to execute a begin transaction operation on a database that is already in transaction state.
2	The program attempted to execute an end transaction operation on a database that is not in transaction state.
3	The program attempted to execute a MIDTRANSACTION statement on a database that is not in transaction state.
4	The program attempted to execute a normal begin or end transaction operation on a database that was opened by TPS.
5	The program attempted to execute a TPS begin transaction or end transaction operation on a database that was not opened by TPS.
6	The transaction record has an improper TRSTATE value.
7	The program attempted to execute a MIDTRANSACTION statement on a database that was not opened by TPS.
8	The program attempted to update an audited database when the program was not in transaction state.
9	The program attempted to abort a single transaction when the program was not in transaction state.
10	The program attempted to write a savepoint record when the program was not in transaction state.

continued

Table B-3. AUDITERROR Subcategories (cont.)

Subcategory Number	Meaning
11	The program attempted either to abort a single transaction or to write a savepoint record, but the DASDL option INDEPENDENTTRANS was not set for the database.

## CLOSEERROR Category

The CLOSEERROR category is also known as major category 12. The most probable cause for a CLOSEERROR message is an error in closing the database.

Table B-4 lists the exceptions and errors subcategories associated with a CLOSEERROR message.

Table B-4. CLOSEERROR Subcategories

Subcategory Number	Meaning
1	The database is not open.
2	An Accessroutines error occurred on one or more structures.

## DATAERROR Category

The DATAERROR category is also known as major category 4. The most probable cause for a DATAERROR message is a null value in a required item.

Table B-5 lists the exceptions and errors subcategories associated with a DATAERROR message.

Table B-5. DATAERROR Subcategories

Subcategory Number	Meaning
1	Either some required items are null, or the VERIFY condition failed. (The divisor of an aggregate item must not be zero.)
2	The program attempted to create variable-format records with an invalid value for the record type.

continued

Table B-5. DATAERROR Subcategories (cont.)

Subcategory Number	Meaning
3	The partition variable yielded an illegal file identifier.
4	READONLY items were changed and now cannot be stored.
5	The divisor of a virtual item was 0 (zero). (This condition is data dependent.)

## DEADLOCK Category

The DEADLOCK category is also known as major category 3. The most probable cause for a DEADLOCK message is either a deadly embrace or deadlock condition.

Table B-6 lists the exceptions and errors subcategories associated with a DEADLOCK message.

Table B-6. DEADLOCK Subcategories

Subcategory Number	Meaning
1	A deadly embrace occurred when the program attempted to lock a record or while the program was waiting for a begin transaction operation.  The system automatically frees all records for this program.
2	A timeout occurred when the program attempted to lock a record or while the program was waiting for a begin transaction operation.
3	The LOCKLIMIT value was exceeded. Too many records in this database are currently locked.

## DUPLICATES Category

The DUPLICATES category is also known as major category 2. This category usually is reported when duplicates are not allowed.

Table B-7 lists the exceptions and errors subcategories associated with a DUPLICATES message.



Table B-7. DUPLICATES Subcategories

Subcategory Number	Meaning
1	The program attempted to store a duplicate key item in an index set in which duplicates are not allowed.
2	The program attempted to insert a duplicate key item in a subset or data set in which duplicates are not allowed.

### FATALERROR Category

The FATALERROR category is also known as major category 19. The most probable cause for a FATALERROR message is a fatal error in the Accessroutines. No exception subcategories are returned for large systems.

### INTEGRITYERROR Category

The INTEGRITYERROR category is also known as major category 20. The most probable cause for an INTEGRITYERROR message is an internal structure problem in the database.

Table B-8 lists the exceptions and errors subcategories associated with an INTEGRITYERROR message.

Table B-8. INTEGRITYERROR Subcategories

Subcategory Number	Meaning
1	The key and the data do not match.
2	The program attempted to store a record in a missing automatic set or subset.
3	The program attempted to delete a record in a missing automatic set or subset.
4	An automatic set or subset is pointing to a deleted record.
5	The program attempted to allocate more than 1000 rows in a file.

### INTLIBERROR Category

The INTLIBERROR category is also known as major category 21. This category relates only to exceptions and errors generated when the interpretive interface is used.

Table B-9 lists the exceptions and errors subcategories associated with an INTLIBERROR message.

**Table B-9. INTLIBERROR Subcategories**

Subcategory Number	Meaning
1	The first parameter of the entry point DEVERB or DBDATA is not a valid language extension verb.
2	The specified string parameter is not a valid keyword.
3	The specified structure is not the correct structure type.
4	The structure or link name is not a known database identifier.
5	The program gave conflicting or insufficient qualification for the structure or link name.
6	The item name is not a database identifier.
7	The item type does not correspond to the requested data type.
8	The specified item is not in the specified structure.
9	The program gave an invalid index for an OCCURS item.
10	The program gave an incorrect number of subscripts for an OCCURS item.
11	Either the specified item is not in the current variable record type, or the program specified an incorrect record-type syntax.
12	The program placed a negative value in an unsigned item.
13	The program attempted a get operation for this item and found that the value was null.
14	The string parameter in a putstring operation is not the same length as the item.
15	The structure or link name is invalid.
16	The item name is invalid.
17	A selection expression contains an invalid literal.
18	A right parenthesis is missing in the selection expression.
19	A relational operator is expected in the selection expression.
20	In a selection expression, either a relational operator specified a null value for a required item, or the relational operator was neither EQU (equal to) nor GEQ (greater than or equal to).
21	Either the selection expression did not end correctly, or some unknown error occurred.
22	A numeric argument to a put operation is invalid for the item.
23	This operation is not valid for global data.

continued

**Table B-9. INTLIBERROR Subcategories (cont.)**

<b>Subcategory Number</b>	<b>Meaning</b>
24	In a selection expression, an item and a literal have either a mismatched length or type.
25	More than 47 relations appear in a selection expression.
26	No valid set spans the embedded data set.
27	A software failure occurred in a selection expression tree execution.
28	The program attempted to execute an invalid request while the database was closed.
29	The program attempted to execute a store or delete operation when the program was not in transaction state.
30	An index set was left undefined by a data set condition search.
31	The parameter is too short to hold the response.
32	The value of a number item does not fit into a real parameter.
33	The LIMITTYPE parameter is invalid.
34	The LIMITVALUE parameter is invalid.
35	The FINDLIMIT value was exceeded.

## **INUSE Category**

The INUSE category is also known as major category 14. The most probable cause for an INUSE messages is a problem with a master record that has embedded records.

Table B-10 lists the exceptions and errors subcategories associated with an INUSE message.

**Table B-10. INUSE Subcategories**

<b>Subcategory Number</b>	<b>Meaning</b>
1	The program attempted to delete a record without setting its reference and embedded set to null.
2	The program attempted to delete a record when the count item was not zero.
3	The program attempted to delete a record that has a nonnull embedded structure.

## IOERROR Category

The IOERROR category is also known as major category 9. The most probable cause for an IOERROR message is an input or output error.

Table B-11 lists the exceptions and errors subcategories associated with an IOERROR message.

**Table B-11. IOERROR Subcategories**

Subcategory Number	Meaning
1	The unit is not ready.
2	A read parity error occurred.
4	A descriptor error occurred.
5	The Accessroutines encountered a NO SPACE condition. This type of exception can occur under the following conditions: <ul style="list-style-type: none"> <li>• A DMSII data file is contained on a multipack family.</li> <li>• One of the packs of the multipack family is re-created.</li> <li>• A program attempted a DMSII operation involving a record on the newly re-created pack.</li> </ul>
6	The program attempted to read past the end of the file.
7	The program attempted to read or write past the end of the last row.
8	The I/O was canceled.
11	A pack write-lockout occurred.
12	A multiplexor or controller error occurred.
13	The pack drive failed.
14	A pack sector format error occurred.
15	A storage unit timeout occurred.
16	A row was locked out because of a previous write error.
17	The checksum operation failed.
99	An error was generated by an unknown cause.
256	The ADDRESSCHECK operation failed.

## KEYCHANGED Category

The KEYCHANGED category is also known as major category 6. The most probable cause for a KEYCHANGED message is a change to a key item when no duplicates are allowed.

## DMSII Exceptions and Errors

---

Table B-12 lists the exceptions and errors subcategory associated with a **KEYCHANGED** message.

**Table B-12. KEYCHANGED Subcategories**

<b>Subcategory Number</b>	<b>Meaning</b>
1	The program attempted to change the key item of a set or automatic subset that does not allow duplicate keys.

## LIMITERROR Category

The **LIMITERROR** category is also known as major category 10. The most probable cause for a **LIMITERROR** message is a file reaching its maximum physical limit.

Table B-13 lists the exceptions and errors subcategories associated with a **LIMITERROR** message.

**Table B-13. LIMITERROR Subcategories**

<b>Subcategory Number</b>	<b>Meaning</b>
1	The program attempted to store too many records in a structure.
2	The program attempted to make too many counted links refer to one record. The count list in the DASDL source file was exceeded.
3	The program attempted to put too many entries into a set or subset.
5	The program attempted to open too many partitions.
6	The program referenced a partitioned structure in a DMUTILITY INITIALIZE statement.
7	There are too many pointers for this structure.

## NORECORD Category

The **NORECORD** category is also known as major category 13. The most probable cause for a **NORECORD** message is an invalid current path.

Table B-14 lists the exceptions and errors subcategories associated with a **NORECORD** message.

Table B-14. NORECORD Subcategories

Subcategory Number	Meaning
1	The current path of the record to be inserted is not valid.
2	The current path of the record to be deleted (removed) is not valid.
3	The current path containing the link is not valid.
4	The current path of the record to be linked to is not valid.
5	The current path of the master record is not valid.
6	The current path of the record to be found is not valid.

## NOTFOUND Category

The NOTFOUND category is also known as major category 1. A NOTFOUND message is most likely to be issued when no record that meets the given criteria can be found.

Table B-15 lists the exceptions and errors subcategories associated with a NOTFOUND message.

Table B-15. NOTFOUND Subcategories

Subcategory Number	Meaning
1	No record satisfies the condition.
2	The program attempted to execute a next operation when the pointer was at the last entry in the structure.
3	The program attempted to execute a prior operation when the pointer was at the first entry in the structure.
4	The manual subset key does not correspond with the key in the data record.
5	No current record could be found.
6	The current path is undefined.
7	Either the master record pointer is undefined, or the embedded structure has not been created.
8	The link is undefined.
9	The link does not agree with the data record.

continued

**Table B-15. NOTFOUND Subcategories (cont.)**

Subcategory Number	Meaning
10	The link does not exist in the current record. (The program specified the wrong type of variable-format record.)
11	A select-text error occurred.
12	An invalid index resulted from an OCCURS clause.

## NOTLOCKED Category

The NOTLOCKED category is also known as major category 5. A NOTLOCKED message is most likely to be issued when a store operation is not preceded by a create or modify operation.

Table B-16 lists the exceptions and errors subcategories associated with a NOTLOCKED message.

**Table B-16. NOTLOCKED Subcategories**

Subcategory Number	Meaning
1	The program attempted to execute a store operation without first performing either a create operation, a lock and modify operation, or a store operation.
2	The program attempted to modify an unlocked record.

## OPENERERROR Category

The OPENERERROR category is also known as major category 11. The most probable cause for an OPENERERROR message is an error in opening the database.

Table B-17 lists the exceptions and errors subcategories associated with an OPENERERROR message.

Table B-17. OPENEROR Subcategories

Subcategory Number	Meaning
1	The Accessroutines is not present on disk.
2	An I/O error occurred in the Accessroutines code file.
4	The DMUTILITY program tried to initialize when the database was in use.
5	The run-time description of the database does not match the compile-time description.
6	An Accessroutines error occurred on one or more structures.
7	The description files for the application program and the DMSUPPORT library were compiled with different timestamps.
9	The program attempted to open RECOVERY, but RECOVERY was not necessary.
10	The program discontinued when the restart data set was opened.
11	A read error occurred on segment 0 of the restart data set.
12	The program discontinued when initializing the restart data set to empty.
13	The program discontinued when changing the names of the restart data set.
14	The restart data set is not present.
15	The program attempted to open RECOVERY, but RECOVERY was already in progress.
16	A required structure was not invoked.
17	An invoked structure was not found.
18	The program attempted to open RECONSTRUCTION when RECONSTRUCTION was already in progress.
19	The reconstruction was unsuccessful.
20	The DBS stack overflowed.
23	The program attempted to initialize one or more partitions when the partitions already existed.
24	The program discontinued because a file attribute error occurred when the program attempted to open a file.
25	The program attempted to open a logical database that did not exist.
26	The program received a security error when attempting to execute an open operation.

continued



## DMSII Exceptions and Errors

---

Table B-17. OPENERROR Subcategories (cont.)

Subcategory Number	Meaning
27	<p>The Accessroutines code file is not the same as the code file used by the database.</p> <p>If the Accessroutines is copied while the database is open, any subsequent program that opens the database also receives this error until the database is closed.</p> <p>Either specify the correct Accessroutines title in the DASDL source file, or remove the incorrect copy from the usercode and family of the program that opened the database.</p>
28	<p>The system was unable to read row 0 of the restart data set. Run the RECONSTRUCT program.</p>
29	<p>The program was compiled with a description file that has a higher update level than the description file used to compile the DMSUPPORT library.</p>
31	<p>An I/O error occurred in the control file.</p>
32	<p>The program attempted to execute an unknown type of open operation.</p>
33	<p>The database was open when a halt/load occurred.</p>
34	<p>The audit trail could not be initialized.</p>
35	<p>There is insufficient memory for the DBS stack.</p>
36	<p>The Accessroutines code file is unacceptable.</p>
37	<p>An open error occurred in the control file.</p>
38	<p>The database software does not match the format for the structure.</p>
39	<p>The control file is locked by an exclusive function.</p>
40	<p>The application program must be recompiled on a more recent release.</p>
41	<p>The program and the DMSUPPORT library are compiled with different description files.</p>
44	<p>The DBS is not visible from the subsystem for the program.</p>
45	<p>This option is not permitted while TPS is active.</p>
46	<p>This option is not permitted until the TPS RECOVERY program is complete.</p>
47	<p>The initiation of the Accessroutines failed.</p>
48	<p>The application program discontinued before the Accessroutines was frozen.</p>
49	<p>The control file is not present on disk.</p>
50	<p>The program received an I/O error while reading block 0 of the control file.</p>
56	<p>The Accessroutines code file does not support audit.</p>

continued

Table B-17. OPENERROR Subcategories (cont.)

Subcategory Number	Meaning
57	The Accessroutines code file does not support partitions.
58	The Accessroutines code file does not support local buffering.
59	The program attempted to open a database that is already in use by an exclusive or single update user.
60	The program attempted to either open a database exclusively or make a database a single update database when the database is already in use by another program.
61	The program attempted to open a Semantic Information Manager (SIM) database, but SIM has not been implemented on the system.
62	The program attempted to update a SIM database that can be accessed only for inquiry purposes.
63	The program attempted to update a DMSII database that can be accessed only for inquiry purposes.
64	The program attempted to library-equate a new DMSUPPORT library, but either there is an existing DMSUPPORT library or the old DMSUPPORT library is still being accessed.
65	The usercode under which the database control file is stored is not a valid usercode for the system.  Either make the usercode a valid system usercode, or move the control file to an existing valid system usercode.
66	The DMSUPPORT library is not resident.
67	The DMSUPPORT library needs to be recompiled.
68	The Absolute Block Serial Number (ABSN) is approaching its maximum allowable value and must be reset. (For information on resetting the ABSN, refer to the <i>DMSII Utilities Operations Guide</i> .)
69	A rebuild or rollback procedure is in progress.
70	The charge code for the database owner is no longer valid.

## READONLY Category

The READONLY category is also known as major category 8. The most probable cause for a READONLY message is an attempt to alter a record when the database is open for inquiry only.

Table B-18 lists the exceptions and errors subcategories associated with a READONLY message.

**Table B-18. READONLY Subcategories**

Subcategory Number	Meaning
1	The program attempted to change the database while in the inquiry access mode.
2	The program attempted an insert or remove operation on a READONLY record.
3	The program attempted a create operation for a new partition while in inquiry mode.

## SECURITYERROR Category

The SECURITYERROR category is also known as major category 17. The most probable cause for a SECURITYERROR message is a violation of system security.

Table B-19 lists the exceptions and errors subcategory associated with a SECURITYERROR message.

**Table B-19. SECURITYERROR Subcategories**

Subcategory Number	Meaning
1	Verb security: either the program or the user is not permitted to use this function.

## SYSTEMERROR Category

The SYSTEMERROR category is also known as major category 7. The most probable cause for a SYSTEMERROR message is an invalid structure number.

Table B-20 lists the exceptions and errors subcategories associated with a SYSTEMERROR message.

**Table B-20. SYSTEMERROR Subcategories**

Subcategory Number	Meaning
1	The program attempted a find first, next, prior, or last operation on an embedded standard or compact data set.
2	A generate operation is not allowed on this type of set.

continued

Table B-20. SYSTEMERROR Subcategories (cont.)

Subcategory Number	Meaning
3	The system attempted to read beyond the end of the file.
4	An Accessroutines fault occurred during a DMSFREE operation.
5	An invalid calling sequence or other sequence of events occurred.
6	<p>A fatal software error occurred. A fatal software error is produced in the following situations:</p> <ul style="list-style-type: none"> <li>● Internal system locks are used inconsistently.</li> <li>● Linkage between blocks of data is inconsistent.</li> <li>● Information read from a mass storage device is inconsistent with information requested.</li> </ul>
7	<p>A fatal integrity error occurred. A fatal integrity error is produced in the following situations:</p> <ul style="list-style-type: none"> <li>● An address of 0 (zero) is encountered.</li> <li>● A block address is not on a block boundary.</li> <li>● A record address is not on a record boundary.</li> <li>● The value of a key item in a set or subset does not match the value in a data set.</li> <li>● A set or an automatic subset points to a deleted record.</li> <li>● A record does not have an entry in a set or an automatic subset when it should have such an entry.</li> </ul>
8	<p>A fatal I/O error occurred. A fatal I/O error occurs in the following situations:</p> <ul style="list-style-type: none"> <li>● The audit file cannot be opened.</li> <li>● Recovery information cannot be read or written.</li> <li>● The Accessroutines encountered I/O errors when attempting to allocate or deallocate file available space.</li> </ul>
9	<p>A fatal resource error occurred. A fatal resource error is encountered in the following situations:</p> <ul style="list-style-type: none"> <li>● The main memory is exhausted.</li> <li>● There is not enough space in the restart data set.</li> </ul>
10	A fatal unknown error occurred. This error occurs when abort recovery terminates on an error.

continued

**Table B-20. SYSTEMERROR Subcategories (cont.)**

<b>Subcategory Number</b>	<b>Meaning</b>
11	A fatal operator discontinue database error occurred. This error occurs in the following situations: <ul style="list-style-type: none"><li>• A program attempted to open a database just after the operator discontinued the database.</li><li>• The database is being terminated due to some fatal error.</li></ul>
12	A software error occurred in the control file handler.
13	The partitions control in the control file are not initialized.
14	The Accessroutines code file is compiled without this structure type.
15	An error occurred while the system was executing an internal SIM remap and test procedure.

## VERSIONERROR Category

The VERSIONERROR category is also known as major category 18. The most probable cause for a VERSIONERROR message is a program-to-database mismatch.

Table B-21 lists the exceptions and errors subcategories associated with a VERSIONERROR message.

**Table B-21. VERSIONERROR Subcategories**

<b>Subcategory Number</b>	<b>Meaning</b>
1	The link is unknown or was deleted.
2	The program attempted to reference a structure that has been deleted or changed since the program was compiled.

## Summarizing the Relationship of Data Management Operations to Exceptions and Errors

Figure B-1 summarizes the data management operations that can cause each exception and error subcategory.

### *Notes:*

- *Exception categories and subcategories resulting from a close operation are valid, whether or not the exception branch is taken.*
- *No DMSII data management operation produces a FATALERROR exception.*
- *Any entry point in the DMINTERPRETER library can cause an INTLIBERROR exception.*
- *The specified LIMITERROR exceptions apply only when the file size is exceeded. Any operation which causes a new partition to be opened implicitly can cause a limiterror. The close, begin transaction, and end transaction operations are the only operations that cannot produce a LIMITERROR exception.*
- *A test link for null can result in a VERSIONERROR if the link was deleted using the REORGANIZATION program.*

# DMSII Exceptions and Errors

OPERATION \ ERRORS AND EXCEPTIONS	ERRORS AND EXCEPTIONS																					
	ABORT	AUDITERROR	CLOSEERROR	DATAERROR	DEADLOCK	DUPLICATES	FATALERROR	INTEGRITY ERROR	INTLIBERROR	INUSE	IOERROR	KEYCHANGE	LIMITERROR	NORECORD	NOTFOUND	NOTLOCKED	OPENEROR	READONLY	SECURITYERROR	SYSTEMERROR	VERSIONERROR	
ABORTTRANSACTION		X																				
ASSIGN		X								X		X	X					X	X			X
BEGINTRANSACTION	X	X		X	X																	
CANCELTRANSACTION		X																				
CLOSE	X		X																			
COMPUTE																						
CREATE				X																		
DELETE		X		X	X			X		X	X		X	X	X			X	X	X	X	X
DMKEY																						
DMTERMINATE																						
ENDTRANSACTION	X	X		X	X																	
FIND								X		X		X	X					X	X	X	X	X
FREE																						
GENERATE		X								X								X	X	X	X	X
GET																						
IF																						
INSERT		X				X				X	X	X	X	X	X			X	X	X	X	X
LOCK/MODIFY				X				X		X		X	X	X	X			X	X	X	X	X
MIDTRANSACTION		X																				
OPEN																	X		X			
PUT																						
RECREATE				X																		
REMOVE		X								X		X	X	X	X			X	X	X	X	X
SAVEPOINT		X																				
SECURE				X				X		X		X	X	X	X			X	X	X	X	X
SET										X		X	X	X	X			X	X	X	X	X
STORE		X		X		X		X		X	X	X	X	X	X			X	X	X	X	X

Figure B-1. Relationship of Data Management Operations to Errors and Exceptions

# Appendix C

## Sample Programs That Use the Interpretive Interface

The ALGOL, COBOL74, and FORTRAN77 application programs shown in this appendix use the DMSII interpretive interface to create, store, retrieve, delete, and re-create various data items of differing types.

All the programs use the same database. The DASDL description of the database is presented first, followed by the programs.



## DASDL Database for Sample Programs

The following DASDL code creates EMPJOB, the database that is accessed by all the sample application programs provided in this appendix.

EMPJOB contains a restart data set, a global record, and three data sets: EMP, JOB, and ADR.

- The EMP data set has an embedded data set (EMP-ED), three sets (EMP-DATE-SET, EMP-NO-SET and EMP-LNAME-SET), and an automatic subset (EMP-OVER10-SUBSET).
- The JOB data set is a variable format data set with two record types. The JOB data set has a set (JOB-CODE-SET), an automatic subset (JOB-MNG-SUBSET), and an embedded, manual subset (JOB-EMP-SUBSET).
- The ADR data set has one set, ADR-ZIP-SET.

For details on DASDL, read the *DMSII DASDL Programming Reference Manual*. For database design techniques, read the *DMSII Technical Overview*.

## DASDL Database for Sample Programs (cont.)

RSTART RESTART DATA SET  
(ALPHAID ALPHA(7));

\*\*\*\*\* GLOBAL RECORD \*\*\*\*\*

FISCAL-YEAR "NEEDS UPDATING EVERY YEAR" NUMBER (6);  
POP-EMP POPULATION (50) OF EMP;  
DIV-NUMBER NUMBER (5);

\*\*\*\*\* EMP \*\*\*\*\*

EMP DATA SET "MASTER EMPLOYEE FILE"  
POPULATION = 50

```
(
EMP-NO                NUMBER      (4); %ACTUAL EMP NUMBER
EMP-NAME              GROUP
  (
  \ EMP-LNAME          ALPHA      (10);
  \ EMP-FNAME          ALPHA      (10);
  \ );
EMP-SSN               NUMBER      (9);
EMP-DATE-HIRED "YYMMDD" NUMBER      (6);
EMP-SEC-CLEAR "T=YES"  BOOLEAN
EMP-SALARY "MONTHLY"  NUMBER      (6,2);
```

\ %\*\*\*\*\* EMP-ED \*\*\*\*\*

EMP-ED DATA SET "EDUCATIONAL RECORDS FOR EACH EMPLOYEE"

```
\ POPULATION = 10
  (
  \ ED-DATE "YYMMDD"   NUMBER      (6);
  \ ED-INST-NAME       ALPHA      (30);
  \ ED-INST-CODE       NUMBER      (4); %UNIQUE TO SCHOOL
  \ ED-COURSE          ALPHA      (10);
  \ ED-GPA             NUMBER      (3,2); %GRADE PT AVR
  \ );
ED-DATE-SET          SET OF EMP-ED KEY ED-DATE I-S;
EMP-JOB-REF          REFERENCE TO JOB COUNTED;
);
EMP-NO-SET           SET OF EMP KEY EMP-NO I-S;
EMP-LNAME-SET        SET OF EMP
                     KEY EMP-LNAME DUPLICATES I-S;
EMP-OVER10-SUBSET "VESTED EMPLOYEES" SUBSET OF EMP
  \ WHERE (EMP-DATE-HIRED < 7000000)
  \ BIT VECTOR;
```

\*\*\*\*\* JOB \*\*\*\*\*

JOB DATA SET "REC TYPE 1 IS CLERICAL, REC TYPE 2 IS NON-CLERICAL"  
POPULATION = 10

```
(
JOB-COUNT             COUNT      (10);
JOB-TYPE              TYPE       (2);
```

## DASDL Database for Sample Programs (cont.)

---

```
JOB-CODE          NUMBER      (4);
JOB-TITLE         ALPHA       (30);
JOB-EMP-SUBSET   SUBSET OF EMP KEY EMP-NO;
),
  1:
  (
  TYPE-SPEED      NUMBER      (2);
  SHORTHAND-SPEED NUMBER      (2);
  CRTUSE-REQD     BOOLEAN      ;
  ),
  2:
  (
  SPANISH-REQD    BOOLEAN      ;
  DRIVING-REQD    BOOLEAN      ;
  JOB-COMMENT     ALPHA       (50);
  );
JOB-CODE-SET      SET OF JOB KEY JOB-CODE I-S;
JOB-MNG-SUBSET "MANAGERS" SUBSET OF JOB WHERE
  \              (JOB-CODE > 4999);
```

\*\*\*\*\* ADR \*\*\*\*\*

```
ADR DATA SET "MASTER ADDRESS FILE"
POPULATION = 50
(
  ADR-EMP-NO      NUMBER      (4);
  ADR-INFO        GROUP
  \ (
  \  ADR-STREET-NO ALPHA      (30);
  \  ADR-ZIP       NUMBER      (5);
  \ );
  );
ADR-ZIP-SET      SET OF ADR KEY ADR-ZIP I-S DUPLICATES;
```

## ALGOL Application Program Using the Interpretive Interface

```

BEGIN
FILE REMOTEFILE
    (KIND=REMOTE
    ,UNITS=CHARACTERS
    ,MAXRECSIZE=1920
    ,MINRECSIZE=1
    ,FILETYPE=3
    );
$$ INCLUDE DMISYM="DATABASE/DMINTERPRETER" 20100000-2019999

%*****%
%* This program uses the ALGOL declaration of library entry points      *%
%* from the DATABASE/DMINTERPRETER symbolic file rather than          *%
%* explicitly declaring each entry point.                               *%
%*****%

%*****%
%*                               LOCAL DECLARATIONS                       *%
%*****%

REAL
    EXPR_1;
DOUBLE
    EXPR_2;
STRING
    DATA_SET_NAME
    ,FIRST_1
    ,PRIOR_1
    ,NEXT_1
    ,P_AUDIT
    ,OPEN_TYPE
    ,BEGIN_1
    ,CONDITION_1
    ,SPACE_1
    ,DATA_BUFFER
    ,DATA_REQUEST
    ,ITEM_NAME
    ,EXPR_3
    ,EXCEPTIONMSG;
BOOLEAN
    RSLT
    ,EXPR_4;

```

# ALGOL Application Program Using the Interpretive Interface (cont.)

---

```
%*****%
%*                INTERNAL PROCEDURE TO HANDLE EXCEPTIONS                *%
%*****%
```

```
PROCEDURE EXCEPTIONHANDLER;
BEGIN
STRING
    EXCEPTIONMSG;
BOOLEAN
    STATUSRSLT;

    STATUSRSLT:=DBSTATUS;
    DBEXCEPTIONNAME(STATUSRSLT,EXCEPTIONMSG);
    WRITE(REMOTEFILE,<"EXCEPTION-NAME=",A14>,EXCEPTIONMSG);
    DBEXCEPTIONTEXT(STATUSRSLT,EXCEPTIONMSG);
    WRITE(REMOTEFILE,<"EXCEPTION-TEXT=">);
    WRITE(REMOTEFILE,<A60>,EXCEPTIONMSG);
END OF EXCEPTIONHANDLER;
```

```
%*****%
%*                BODY OF MAIN PROGRAM                *%
%*****%
```

```
DMI.TITLE:="DMINTERPRETER/EMPJOB";
%*****%
DATA_SET_NAME:="EMP";
FIRST_1:="FIRST";
PRIOR_1:="PRIOR";
NEXT_1:="NEXT";
P_AUDIT:="AUDIT";
OPEN_TYPE:="UPDATE";
BEGIN_1:="BEGINNING";
CONDITION_1:="EMP-NO=11";
SPACE_1:=" ";
```

```
%*****%
%*                OPENING THE DATABASE                *%
%*****%
```

```
RSLT:=DBOPEN(OPEN_TYPE);
IF RSLT
    THEN EXCEPTIONHANDLER;
WRITE(REMOTEFILE,<"SUCCESSFUL OPEN OF EMPJOB">);
```

## ALGOL Application Program Using the Interpretive Interface (cont.)

```
*****%
%*          DELETING A RECORD          *%
*****%
```

```
RSLT:=DBBEGINTRANSACTION(P_AUDIT);
IF RSLT
  THEN EXCEPTIONHANDLER;

RSLT:=DBDELETE(FIRST_1,DATA_SET_NAME,CONDITION_1);
IF RSLT
  THEN EXCEPTIONHANDLER;

CONDITION_1:="EMP-NO=12";
RSLT:=DBDELETE(FIRST_1,DATA_SET_NAME,CONDITION_1);
IF RSLT
  THEN EXCEPTIONHANDLER;

RSLT:=DBENDTRANSACTION(P_AUDIT);
IF RSLT
  THEN EXCEPTIONHANDLER;

WRITE(REMOTEFILE, <"SUCCESSFUL DELETE OF RECORD">);
```

```
*****%
%*          CREATING A RECORD          *%
*****%
```

```
RSLT:=DBCREATE(DATA_SET_NAME,SPACE_1);
IF RSLT
  THEN EXCEPTIONHANDLER;

WRITE(REMOTEFILE, <"SUCCESSFUL CREATE OF RECORD">);
```

```
*****%
%*          FILLING IN A RECORD        *%
*****%
```

```
%
%          USING DBPUTREAL ENTRY POINT
%

ITEM_NAME:="EMP-NO";
EXPR_1:=11;
RSLT:=DBPUTREAL(DATA_SET_NAME,ITEM_NAME,EXPR_1);
IF RSLT
  THEN EXCEPTIONHANDLER;

ITEM_NAME:="EMP-SALARY";
EXPR_1:=4500.95;
RSLT:=DBPUTREAL(DATA_SET_NAME,ITEM_NAME, 2 * EXPR_1);
IF RSLT
  THEN EXCEPTIONHANDLER;
```

## ALGOL Application Program Using the Interpretive Interface (cont.)

---

```
%  
%       USING DBPUTDOUBLE ENTRY POINT  
%  
ITEM_NAME:="EMP-SSN";  
EXPR_2:=573230911;  
RSLT:=DBPUTDOUBLE(DATA_SET_NAME,ITEM_NAME,EXPR_2 - 11 );  
IF RSLT  
    THEN EXCEPTIONHANDLER;  
  
%  
%       USING DBDATA ENTRY POINT  
%  
DATA_REQUEST:="PUT REAL EMP EMP-DATE-HIRED";  
DATA_BUFFER:="811223";  
RSLT:=DBDATA(DATA_REQUEST,DATA_BUFFER);  
IF RSLT  
    THEN EXCEPTIONHANDLER;  
  
DATA_REQUEST:="PUT STRING EMP EMP-LNAME";  
DATA_BUFFER:="STACK";  
RSLT:=DBDATA(DATA_REQUEST,DATA_BUFFER);  
IF RSLT  
    THEN EXCEPTIONHANDLER;  
  
%  
%       USING DBPUTSTRING ENTRY POINT  
%  
ITEM_NAME:="EMP-FNAME";  
EXPR_3:="CATHY";  
RSLT:=DBPUTSTRING(DATA_SET_NAME,ITEM_NAME,EXPR_3 CAT "LINA");  
IF RSLT  
    THEN EXCEPTIONHANDLER;  
  
%  
%       USING DBPUTBOOLEAN  
%  
ITEM_NAME:="EMP-SEC-CLEAR";  
EXPR_4:=TRUE;  
RSLT:=DBPUTBOOLEAN(DATA_SET_NAME,ITEM_NAME,NOT EXPR_4);  
IF RSLT  
    THEN EXCEPTIONHANDLER;
```

## ALGOL Application Program Using the Interpretive Interface (cont.)

---

```

%*****%
%*          RECORD SETUP IN USER WORK AREA          *%
%*****%
%*          *%
%* EMP      EMP      EMP      EMP      EMP      EMP      EMP      *%
%* NO      LNAME    FNAME    SSN     DATE-HIRED  SEC-CLEAR  SALARY  *%
%*-----*%
%*21  STACK  CATHYLINA 573230900  811223      FALSE      9001.9 *%
%*****%

```

```

%*****%
%*          STORING A RECORD                        *%
%*****%

```

```

RSLT:=DBBEGINTRANSACTION(P_AUDIT);
IF RSLT
  THEN EXCEPTIONHANDLER;

```

```

RSLT:=DBSTORE(DATA_SET_NAME);
IF RSLT
  THEN EXCEPTIONHANDLER;

```

```

RSLT:=DBENDTRANSACTION(P_AUDIT);
IF RSLT
  THEN EXCEPTIONHANDLER;

```

```

WRITE(REMOTEFILE, <"SUCCESSFUL STORE OF RECORD">);

```

```

%*****%
%*          RE-CREATING A RECORD                    *%
%*****%

```

```

RSLT:=DBRECREATE(DATA_SET_NAME,SPACE_1);
IF RSLT
  THEN EXCEPTIONHANDLER;

```

```

WRITE(REMOTEFILE, <"SUCCESSFUL RECREATE OF RECORD">);

```

```

ITEM_NAME:="EMP-NO";
EXPR_1:=12;
RSLT:=DBPUTREAL(DATA_SET_NAME,ITEM_NAME,EXPR_1);
IF RSLT
  THEN EXCEPTIONHANDLER;

```

```

ITEM_NAME:="EMP-SSN";
EXPR_2:=549927328;
RSLT:=DBPUTDOUBLE(DATA_SET_NAME,ITEM_NAME,EXPR_2 - 11 );
IF RSLT
  THEN EXCEPTIONHANDLER;

```



## ALGOL Application Program Using the Interpretive Interface (cont.)

```

DATA_REQUEST:="PUT REAL EMP EMP-DATE-HIRED";
DATA_BUFFER:="811228";
RSLT:=DBDATA(DATA_REQUEST,DATA_BUFFER);
IF RSLT
    THEN EXCEPTIONHANDLER;

DATA_REQUEST:="PUT STRING EMP EMP-LNAME";
DATA_BUFFER:="WOLVERTON";
RSLT:=DBDATA(DATA_REQUEST,DATA_BUFFER);
IF RSLT
    THEN EXCEPTIONHANDLER;

ITEM_NAME:="EMP-FNAME";
EXPR_3:="LAURA";
RSLT:=DBPUTSTRING(DATA_SET_NAME,ITEM_NAME,EXPR_3 CAT "JANE" );
IF RSLT
    THEN EXCEPTIONHANDLER;

RSLT:=DBBEGINTRANSACTION(P_AUDIT);
IF RSLT
    THEN EXCEPTIONHANDLER;

RSLT:=DBSTORE(DATA_SET_NAME);
IF RSLT
    THEN EXCEPTIONHANDLER;

RSLT:=DBENDTRANSACTION(P_AUDIT);
IF RSLT
    THEN EXCEPTIONHANDLER;

WRITE(REMOTEFILE, <"SUCCESSFUL STORE OF RE-CREATED RECORD">);

```

```

%*****%
%*          SETUP OF RE-CREATED RECORD IN USER WORK AREA          *%
%*****%
%*          *%
%* EMP      EMP      EMP      EMP      EMP      EMP      EMP      *%
%* NO      LNAME     FNAME     SSN      DATE-HIRED  SEC-CLEAR  SALARY *%
%*-----*%
%* 12  WOLVERTON  LAURAJANE  549927317  811228      FALSE      9001.9 *%
%*****%

```

```

%*****%
%*          SETTING DATA SET TO THE BEGINNING                    *%
%*****%

```

```

RSLT:=DBSET(DATA_SET_NAME,BEGIN_1);
IF RSLT
    THEN EXCEPTIONHANDLER;

WRITE(REMOTEFILE, <"SUCCESSFUL DBSET OF DATA SET">);

```

## ALGOL Application Program Using the Interpretive Interface (cont.)

---

```
%*****%
%*                               FINDING A RECORD                               *%
%*****%
```

```
RSLT:=DBFIND(FIRST_1,DATA_SET_NAME,CONDITION_1);
IF RSLT
  THEN EXCEPTIONHANDLER;

WRITE(REMOTEFILE, <"SUCCESSFUL FIND OF RECORD">);
```

```
%*****%
%*                               LOCKING A RECORD                               *%
%*****%
```

```
RSLT:=DBLOCK(FIRST_1,DATA_SET_NAME,CONDITION_1);
IF RSLT
  THEN EXCEPTIONHANDLER;

WRITE(REMOTEFILE, <"SUCCESSFUL LOCK OF RECORD">);
```

```
%*****%
%*                               VERIFYING ITEMS IN THE RECORD                               *%
%*****%
```

```
%
%   USING DBGETREAL ENTRY POINT
%
ITEM_NAME:="EMP-NO";
RSLT:=DBGETREAL(DATA_SET_NAME,ITEM_NAME,EXPR_1);
IF RSLT
  THEN EXCEPTIONHANDLER;
WRITE(REMOTEFILE, <"EMP-NO=",I4>,EXPR_1);

ITEM_NAME:="EMP_SALARY";
RSLT:=DBGETREAL(DATA_SET_NAME,ITEM_NAME,EXPR_1);
IF RSLT
  THEN
  WRITE(REMOTEFILE, <"EMP-SALARY=",F9.2>,EXPR_1);

%
%   USING DBGETDOUBLE ENTRY POINT
%
ITEM_NAME:="EMP-SSN";
RSLT:=DBGETDOUBLE(DATA_SET_NAME,ITEM_NAME,EXPR_2);
IF RSLT
  THEN EXCEPTIONHANDLER;
WRITE(REMOTEFILE, <"EMP-SSN=",I9>,EXPR_2);
```

## ALGOL Application Program Using the Interpretive Interface (cont.)

---

```
%
%       USING DBDATA ENTRY POINT
%
DATA_REQUEST:="GET REAL EMP EMP-DATE-HIRED";
RSLT:=DBDATA(DATA_REQUEST,DATA_BUFFER);
IF RSLT
    THEN EXCEPTIONHANDLER;
WRITE(REMOTEFILE, <"EMP-DATE-HIRED=",A6>,DATA_BUFFER);

DATA_REQUEST:="GET REAL EMP EMP-SALARY";
RSLT:=DBDATA(DATA_REQUEST,DATA_BUFFER);
IF RSLT
    THEN EXCEPTIONHANDLER;
WRITE(REMOTEFILE, <"EMP-SALARY=",A6>,DATA_BUFFER);

DATA_REQUEST:="GET STRING EMP EMP-LNAME";
RSLT:=DBDATA(DATA_REQUEST,DATA_BUFFER);
IF RSLT
    THEN EXCEPTIONHANDLER;
WRITE(REMOTEFILE, <"EMP-LNAME=",A10>,DATA_BUFFER);

%
%       USING DBGETSTRING ENTRY POINT
%
ITEM_NAME:="EMP-FNAME";
RSLT:=DBGETSTRING(DATA_SET_NAME,ITEM_NAME,EXPR_3);
IF RSLT
    THEN EXCEPTIONHANDLER;
WRITE(REMOTEFILE, <"EMP-FNAME=",A10>,EXPR_3);

%
%       USING DBGETBOOLEAN ENTRY POINT
%
ITEM_NAME:="EMP-SEC-CLEAR";
RSLT:=DBGETBOOLEAN(DATA_SET_NAME,ITEM_NAME,EXPR_4);
IF RSLT
    THEN EXCEPTIONHANDLER;
WRITE(REMOTEFILE, <"EMP-SEC-CLEAR=",L5>,EXPR_4);

%*****%
%*               FREEING A RECORD               *%
%*****%

RSLT:=DBFREE(DATA_SET_NAME);
IF RSLT
    THEN EXCEPTIONHANDLER;

WRITE(REMOTEFILE, <"SUCCESSFUL FREEING OF RECORD">);
```

## ALGOL Application Program Using the Interpretive Interface (cont.)

---

```
%*****%
%*                CAUSING AN EXCEPTION                *%
%*****%

WRITE(REMOTEFILE, <"CAUSING AN EXCEPTION">);

RSLT:=DBSET(DATA_SET_NAME,BEGIN_1);
IF RSLT
  THEN EXCEPTIONHANDLER;

RSLT:=DBFIND(PRIOR_1,DATA_SET_NAME,SPACE_1);
IF RSLT
  THEN EXCEPTIONHANDLER;

%
%*****%
%*                USING DBVERB                *%
%*****%

%
%                USING DBVERB ON A SET STATEMENT
%
RSLT:=DBVERB("SET EMP TO BEGINNING");
IF RSLT
  THEN EXCEPTIONHANDLER
WRITE(REMOTEFILE,<"SUCCESSFUL SET EMP TO BEGINNING">);

%
%                USING DBVERB ON A SEARCH STATEMENT
%
RSLT:=DBVERB("FIND FIRST EMP AT EMP-NO=11");
IF RSLT
  THEN EXCEPTIONHANDLER;
WRITE(REMOTEFILE,<"SUCCESSFUL FIND FIRST EMP AT EMP-NO=11">);

%
%                USING DBVERB ON A STATUS STATEMENT
%
RSLT:=DBVERB("SET EMP TO BEGINNING");
IF RSLT
  THEN EXCEPTIONHANDLER;
DBVERB("FIND PRIOR EMP ");
RSLT:=DBVERB("DMSTATUS"); % TO FIND OUT IF THE FIND WAS OK
DBEXCEPTIONNAME(RSLT,EXCEPTIONMSG);
WRITE (REMOTEFILE,<"EXCEPTION-NAME=",A14>,EXCEPTIONMSG);
DBEXCEPTIONTEXT(RSLT,EXCEPTIONMSG);
WRITE(REMOTEFILE,<"EXCEPTION-TEXT=">);
WRITE(REMOTEFILE,<A60>,EXCEPTIONMSG);
WRITE(REMOTEFILE,<"SUCCESSFUL STATUS STATEMENT">);
```

## ALGOL Application Program Using the Interpretive Interface (cont.)

---

```
%*****%
%*                CLOSING THE DATABASE                *%
%*****%

    RSLT:=DBCLOSE;
    IF RSLT
      THEN EXCEPTIONHANDLER

    WRITE(REMOTEFILE, <"SUCCESSFUL CLOSE OF EMPJOB">);

%*****%
%*****%
END OF DMIEXAMPLE.
```

# COBOL74 Application Program Using the Interpretive Interface

```

*****
IDENTIFICATION DIVISION.
*****
*PROGRAM-ID.
*AUTHOR.
*INSTALLATION.
*DATE-WRITTEN.
*DATE-COMPILED.
*SECURITY.      PUBLIC.
*REMARKS.
*THIS PROGRAM USES ENTRY POINTS OF THE DMINTERPRETER LIBRARY
*COMPILED FOR THE EMPJOB SAMPLE DATABASE.
*****
ENVIRONMENT DIVISION.
*****
CONFIGURATION SECTION.
*=====
SOURCE-COMPUTER. A 15.
OBJECT-COMPUTER. A 15.
*=====
INPUT-OUTPUT SECTION.
*=====
FILE-CONTROL.
    SELECT REMOTEFIL ASSIGN TO REMOTE.
*****
DATA DIVISION.
*****
FILE SECTION.
FD  REMOTEFIL
   RECORD CONTAINS 80 CHARACTERS
   VALUE OF TITLE IS "REMOTEFIL".
   01  REMOTE-REC          PIC X(80).
*=====
WORKING-STORAGE SECTION.
*DMINTERPRETER CODE FILE.
*-----
   01  DMI-NAME            PIC X(24) VALUE "DMINTERPRETER/EMPJOB".
*DMI VOCABULARY.
*-----
   01  DATA-SET-NAME     PIC X(17) VALUE IS "EMP".
   01  ITEM-NAME          PIC X(17) VALUE IS SPACES.
   01  FIRST-1            PIC X(5) VALUE "FIRST".
   01  PRIOR-1            PIC X(5) VALUE "PRIOR".
   01  NEXT-1             PIC X(4) VALUE "NEXT".
   01  P-AUDIT            PIC X(5) VALUE "AUDIT".
   01  N-AUDIT            PIC X(8) VALUE "NOAUDIT".
   01  OPEN-TYPE          PIC X(6) VALUE "UPDATE".
   01  BEGIN-1           PIC X(9) VALUE "BEGINNING".

```

COBOL74 Application Program Using the Interpretive Interface (cont.)

Ø1 CONDITION-1 PIC X(9) VALUE "EMP-NO=11".  
Ø1 SPACE-1 PIC X(1) VALUE " ".  
Ø1 DATA-BUFFER PIC X(1Ø) VALUE IS SPACES.  
Ø1 DATA-REQUEST PIC X(4Ø) VALUE IS SPACES.  
Ø1 VAR-1 PIC X(1Ø) VALUE IS SPACES.

\*  
\* EXCEPTION AREA

Ø1 RESULT PIC 9(1) COMP VALUE IS Ø.  
Ø1 EXCEPTION-NAME PIC X(14).  
Ø1 EXCEPTION-MSG PIC X(8Ø).  
Ø1 EXCEPTION-WORD PIC 9(12) COMP.

\*\*\*\*\* EXCEPTION-WORD FIELDS\*\*\*\*\*

\* Ø2 STRUCTURE PIC 9(4) COMP.  
\* Ø2 CATEGORY PIC 9(3) COMP.  
\* Ø2 SUB-CATEGORY PIC 9(4) COMP.  
\* Ø2 FLAG PIC 9(1) COMP.  
\*

\*\*\*\*\*

PROCEDURE DIVISION.  
MAIN-SECTION.

\*\*\*\*\*  
\* BODY OF PROGRAM \*

\*\*\*\*\* LINKING TO DMI LIBRARY \*\*\*\*\*

CHANGE ATTRIBUTE TITLE OF "DMINTERPRETER"  
TO DMI-NAME.

\*\*\*\*\*

OPEN OUTPUT REMOTEFILE.

\*\*\*\*\*  
\* OPENING THE DATABASE \*

CALL "DBOPEN OF DMINTERPRETER"  
USING OPEN-TYPE  
GIVING RESULT.  
IF RESULT = 1  
PERFORM EXCEPTION-HANDLER.

## COBOL74 Application Program Using the Interpretive Interface (cont.)

---

```
*****  
                                DELETING A RECORD                                *  
*****
```

```
CALL "DBBEGINTRANSACTION OF DMINTERPRETER"  
  USING P-AUDIT  
  GIVING RESULT.  
IF RESULT = 1  
  PERFORM EXCEPTION-HANDLER.  
CALL "DBDELETE OF DMINTERPRETER"  
  USING FIRST-1, DATA-SET-NAME, CONDITION-1  
  GIVING RESULT.  
IF RESULT = 1  
  PERFORM EXCEPTION-HANDLER.  
MOVE "EMP-NO=12" TO CONDITION-1.  
CALL "DBDELETE OF DMINTERPRETER"  
  USING FIRST-1, DATA-SET-NAME, CONDITION-1  
  GIVING RESULT.  
IF RESULT = 1  
  PERFORM EXCEPTION-HANDLER.  
CALL "DBENDTRANSACTION OF DMINTERPRETER"  
  USING P-AUDIT  
  GIVING RESULT.  
IF RESULT = 1  
  PERFORM EXCEPTION-HANDLER.
```

```
*****  
                                CREATING A RECORD                                *  
*****
```

```
CALL "DBCREATE OF DMINTERPRETER"  
  USING DATA-SET-NAME, SPACE-1  
  GIVING RESULT.  
IF RESULT = 1  
  PERFORM EXCEPTION-HANDLER.
```

```
*****  
                                FILLING IN A RECORD                                *  
*****
```

```
*  
*  
*                                USING DBDATA ENTRY POINT                                *  
*  
MOVE "PUT REAL EMP EMP-NO" TO DATA-REQUEST.  
MOVE "11" TO DATA-BUFFER.  
CALL "DBDATA OF DMINTERPRETER"  
  USING DATA-REQUEST, DATA-BUFFER  
  GIVING RESULT.  
IF RESULT = 1  
  PERFORM EXCEPTION-HANDLER.
```



## COBOL74 Application Program Using the Interpretive Interface (cont.)

---

```
MOVE "PUT REAL EMP EMP-SALARY" TO DATA-REQUEST.
MOVE "9001.90" TO DATA-BUFFER.
CALL "DBDATA OF DMINTERPRETER"
  USING DATA-REQUEST,DATA-BUFFER
  GIVING RESULT.
IF RESULT = 1
  PERFORM EXCEPTION-HANDLER.

MOVE "PUT REAL EMP EMP-SSN" TO DATA-REQUEST.
MOVE "573230911" TO DATA-BUFFER.
CALL "DBDATA OF DMINTERPRETER"
  USING DATA-REQUEST,DATA-BUFFER
  GIVING RESULT.
IF RESULT = 1
  PERFORM EXCEPTION-HANDLER.

MOVE "PUT REAL EMP EMP-DATE-HIRED" TO DATA-REQUEST.
MOVE "811223" TO DATA-BUFFER.
CALL "DBDATA OF DMINTERPRETER"
  USING DATA-REQUEST,DATA-BUFFER
  GIVING RESULT.
IF RESULT = 1
  PERFORM EXCEPTION-HANDLER.

MOVE "PUT STRING EMP EMP-LNAME" TO DATA-REQUEST.
MOVE "STACK" TO DATA-BUFFER.
CALL "DBDATA OF DMINTERPRETER"
  USING DATA-REQUEST,DATA-BUFFER
  GIVING RESULT.
IF RESULT = 1
  PERFORM EXCEPTION-HANDLER.
*
*           USING DBPUTDISPLAY ENTRY POINT
*
MOVE "EMP-FNAME" TO ITEM-NAME.
MOVE "CATHY" TO VAR-1.
CALL "DBPUTDISPLAY OF DMINTERPRETER"
  USING DATA-SET-NAME,ITEM-NAME,VAR-1
  GIVING RESULT.
IF RESULT = 1
  PERFORM EXCEPTION-HANDLER.
*
*           USING DBPUTBOOLEAN
*
MOVE "EMP-SEC-CLEAR" TO ITEM-NAME.
MOVE "TRUE" TO VAR-1.
CALL "DBPUTBOOLEAN OF DMINTERPRETER"
  USING DATA-SET-NAME,ITEM-NAME,VAR-1
  GIVING RESULT.
IF RESULT = 1
  PERFORM EXCEPTION-HANDLER.
```

## COBOL74 Application Program Using the Interpretive Interface (cont.)

---

```

*****
*                RECORD SETUP IN USER WORK AREA                *
*****
* EMP   EMP   EMP   EMP   EMP   EMP   EMP   *
* NO   LNAME  FNAME  SSN   DATE-HIRED  SEC-CLEAR  SALARY  *
*-----*
* 11  STACK  CATHY  573230911  811223      TRUE    9001.90 *
*****

```

```

*****
*                STORING A RECORD                               *
*****

```

```

CALL "DBBEGINTRANSACTION OF DMINTERPRETER"
    USING P-AUDIT
    GIVING RESULT.
IF RESULT = 1
    PERFORM EXCEPTION-HANDLER.
CALL "DBSTORE OF DMINTERPRETER"
    USING DATA-SET-NAME
    GIVING RESULT.
IF RESULT = 1
    PERFORM EXCEPTION-HANDLER.
CALL "DBENDTRANSACTION OF DMINTERPRETER"
    USING P-AUDIT
    GIVING RESULT.
IF RESULT = 1
    PERFORM EXCEPTION-HANDLER.

```

```

*****
*                RE-CREATING A RECORD                           *
*****

```

```

CALL "DBRECREATE OF DMINTERPRETER"
    USING DATA-SET-NAME, SPACE-1
    GIVING RESULT.
IF RESULT = 1
    PERFORM EXCEPTION-HANDLER.

MOVE "PUT REAL EMP EMP-NO" TO DATA-REQUEST.
MOVE "12" TO DATA-BUFFER.
CALL "DBDATA OF DMINTERPRETER"
    USING DATA-REQUEST, DATA-BUFFER
    GIVING RESULT.
IF RESULT = 1
    PERFORM EXCEPTION-HANDLER.

```

## COBOL74 Application Program Using the Interpretive Interface (cont.)

---

```
MOVE "PUT REAL EMP EMP-SSN" TO DATA-REQUEST.
MOVE "549927328" TO DATA-BUFFER.
CALL "DBDATA OF DMINTERPRETER"
  USING DATA-REQUEST,DATA-BUFFER
  GIVING RESULT.
IF RESULT = 1
  PERFORM EXCEPTION-HANDLER.

MOVE "PUT REAL EMP EMP-DATE-HIRED" TO DATA-REQUEST.
MOVE "811228" TO DATA-BUFFER.
CALL "DBDATA OF DMINTERPRETER"
  USING DATA-REQUEST,DATA-BUFFER
  GIVING RESULT.
IF RESULT = 1
  PERFORM EXCEPTION-HANDLER.

MOVE "PUT STRING EMP EMP-LNAME" TO DATA-REQUEST.
MOVE "WOLVERTON" TO DATA-BUFFER.
CALL "DBDATA OF DMINTERPRETER"
  USING DATA-REQUEST,DATA-BUFFER
  GIVING RESULT.
IF RESULT = 1
  PERFORM EXCEPTION-HANDLER.

MOVE "EMP-FNAME" TO ITEM-NAME.
MOVE "LAURA" TO VAR-1.
CALL "DBPUTDISPLAY OF DMINTERPRETER"
  USING DATA-SET-NAME,ITEM-NAME,VAR-1
  GIVING RESULT.
IF RESULT = 1
  PERFORM EXCEPTION-HANDLER.
```

## COBOL74 Application Program Using the Interpretive Interface (cont.)

---

```
*****
*          SETUP OF RE-CREATED RECORD IN USER WORK AREA          *
*****
* EMP      EMP      EMP      EMP      EMP      EMP      EMP      *
* NO      LNAME     FNAME     SSN      DATE-HIRED  SEC-CLEAR  SALARY  *
*-----*-----*-----*-----*-----*-----*-----*
* 12  WOLVERTON  LAURA  549927328  811228      TRUE      9001.90 *
*****
```

```
CALL "DBBEGINTRANSACTION OF DMINTERPRETER"
    USING P-AUDIT
    GIVING RESULT.
IF RESULT = 1
    PERFORM EXCEPTION-HANDLER.
CALL "DBSTORE OF DMINTERPRETER"
    USING DATA-SET-NAME
    GIVING RESULT.
IF RESULT = 1
    PERFORM EXCEPTION-HANDLER.
CALL "DBENDTRANSACTION OF DMINTERPRETER"
    USING P-AUDIT
    GIVING RESULT.
IF RESULT = 1
    PERFORM EXCEPTION-HANDLER.
```

```
*****
*          SETTING DATA SET TO THE BEGINNING                    *
*****
```

```
CALL "DBSET OF DMINTERPRETER"
    USING DATA-SET-NAME, BEGIN-1
    GIVING RESULT.
IF RESULT = 1
    PERFORM EXCEPTION-HANDLER.
```

```
*****
*          FINDING A RECORD                                       *
*****
```

```
CALL "DBFIND OF DMINTERPRETER"
    USING FIRST-1, DATA-SET-NAME, CONDITION-1
    GIVING RESULT.
IF RESULT = 1
    PERFORM EXCEPTION-HANDLER.
```

## COBOL74 Application Program Using the Interpretive Interface (cont.)

---

```
*****  
*                               LOCKING A RECORD                               *  
*****
```

```
CALL "DBLOCK OF DMINTERPRETER"  
  USING FIRST-1, DATA-SET-NAME, CONDITION-1  
  GIVING RESULT.  
IF RESULT = 1  
  PERFORM EXCEPTION-HANDLER.
```

```
*****  
*                               VERIFYING ITEMS IN RECORD                               *  
*****
```

```
*  
*                               USING DBDATA ENTRY POINT                               *  
*
```

```
MOVE "GET REAL EMP EMP-NO" TO DATA-REQUEST.  
CALL "DBDATA OF DMINTERPRETER"  
  USING DATA-REQUEST,DATA-BUFFER  
  GIVING RESULT.  
IF RESULT = 1  
  PERFORM EXCEPTION-HANDLER.
```

```
MOVE "GET REAL EMP EMP-SALARY" TO DATA-REQUEST.  
CALL "DBDATA OF DMINTERPRETER"  
  USING DATA-REQUEST,DATA-BUFFER  
  GIVING RESULT.  
IF RESULT = 1  
  PERFORM EXCEPTION-HANDLER.
```

```
MOVE "GET REAL EMP EMP-SSN" TO DATA-REQUEST.  
CALL "DBDATA OF DMINTERPRETER"  
  USING DATA-REQUEST,DATA-BUFFER  
  GIVING RESULT.  
IF RESULT = 1  
  PERFORM EXCEPTION-HANDLER.
```

```
MOVE "GET REAL EMP EMP-DATE-HIRED" TO DATA-REQUEST.  
CALL "DBDATA OF DMINTERPRETER"  
  USING DATA-REQUEST,DATA-BUFFER  
  GIVING RESULT.  
IF RESULT = 1  
  PERFORM EXCEPTION-HANDLER.
```

```
MOVE "GET STRING EMP EMP-LNAME" TO DATA-REQUEST.  
CALL "DBDATA OF DMINTERPRETER"  
  USING DATA-REQUEST,DATA-BUFFER  
  GIVING RESULT.  
IF RESULT = 1  
  PERFORM EXCEPTION-HANDLER.
```

## COBOL74 Application Program Using the Interpretive Interface (cont.)

---

```
*
*      USING DBGETDISPLAY ENTRY POINT
*
MOVE "EMP-FNAME" TO ITEM-NAME.
CALL "DBGETDISPLAY OF DMINTERPRETER"
  USING DATA-SET-NAME,ITEM-NAME,VAR-1
  GIVING RESULT.
IF RESULT = 1
  PERFORM EXCEPTION-HANDLER.
*
*      USING DBGETBOOLEAN
*
MOVE "EMP-SEC-CLEAR" TO ITEM-NAME.
CALL "DBGETBOOLEAN OF DMINTERPRETER"
  USING DATA-SET-NAME,ITEM-NAME,VAR-1
  GIVING RESULT.
IF RESULT = 1
  PERFORM EXCEPTION-HANDLER.

*****
*
*      FREEING A RECORD
*
*****

CALL "DBFREE OF DMINTERPRETER"
  USING DATA-SET-NAME
  GIVING RESULT.
IF RESULT = 1
  PERFORM EXCEPTION-HANDLER.

*****
*
*      CAUSING AN EXCEPTION
*
*****

CALL "DBFIND OF DMINTERPRETER"
  USING FIRST-1, DATA-SET-NAME, SPACE-1
  GIVING RESULT.
IF RESULT = 1
  PERFORM EXCEPTION-HANDLER.
CALL "DBFIND OF DMINTERPRETER"
  USING PRIOR-1, DATA-SET-NAME, SPACE-1
  GIVING RESULT.
IF RESULT = 1
  PERFORM EXCEPTION-HANDLER.
```

## COBOL74 Application Program Using the Interpretive Interface (cont.)

---

```
*****
*                               USING DBVERB                               *
*****
*
*       USING DBVERB ON A SET STATEMENT
*
    MOVE "SET EMP TO BEGINNING" TO DATA-REQUEST.
    CALL "DBVERB OF DMINTERPRETER"
        USING DATA-REQUEST
        GIVING RESULT.
    IF RESULT = 1
        PERFORM EXCEPTION-HANDLER.
*
*       USING DBVERB ON A SEARCH STATEMENT
*
    MOVE "FIND FIRST EMP AT EMP-NO=11" TO DATA-REQUEST.
    CALL "DBVERB OF DMINTERPRETER"
        USING DATA-REQUEST
        GIVING RESULT.
    IF RESULT = 1
        PERFORM EXCEPTION-HANDLER.

*****
*                               CLOSING THE DATABASE                       *
*****

    CALL "DBCLOSE OF DMINTERPRETER"
        GIVING RESULT.
    IF RESULT = 1
        PERFORM EXCEPTION-HANDLER.

    CLOSE REMOTEFILE.
    STOP RUN.

EXCEPTION-HANDLER.

    CALL "DBSTATUS OF DMINTERPRETER"
        USING EXCEPTION-WORD.
    CALL "DBEXCEPTIONNAME OF DMINTERPRETER"
        USING EXCEPTION-WORD, EXCEPTION-NAME.
    MOVE EXCEPTION-NAME TO REMOTE-REC.
    WRITE REMOTE-REC.
    CALL "DBEXCEPTIONTEXT OF DMINTERPRETER"
        USING EXCEPTION-WORD, EXCEPTION-MSG.
    MOVE EXCEPTION-MSG TO REMOTE-REC.
    WRITE REMOTE-REC.

END-OF-PROGRAM.
EXIT PROGRAM.
```

## FORTRAN77 Application Program Using the Interpretive Interface

```

*****
* This program uses the FORTRAN77 declaration of library entry points *
* from the DATABASE/DMINTERPRETER symbolic file rather than          *
* explicitly declaring each entry point.                               *
*****
*
$ RESET FREE
*
*****
*          FORTRAN77 LIBRARY AND ENTRY POINT DECLARATIONS          *
*****

      BLOCK GLOBALS
      LIBRARY DMI(TITLE="DMINTERPRETER/EMPJOB.")
      END

$ INCLUDE "DATABASE/DMINTERPRETER" 20200000-20249999

*****
*          SUBROUTINE EXCEPT          *
*****

      SUBROUTINE EXCEPT

$ INCLUDE "DATABASE/DMINTERPRETER" 20290000-20299999

      CHARACTER      MSG      *70
      LOGICAL        RESULT

      RESULT = DBSTAT( )

      CALL DBEXCN(RESULT,MSG)
      WRITE(6,200) MSG

      CALL DBEXCT(RESULT,MSG)
      WRITE(6,200) MSG
200  FORMAT(1X,A)
      RETURN
      END

$ INCLUDE "DATABASE/DMINTERPRETER" 20290000-20299999

```



# FORTRAN77 Application Program Using the Interpretive Interface (cont.)

```
*****  
*                               START OF MAIN PROGRAM                               *  
*****
```

```
CHARACTER DSNAME      *4      /"EMP"/,  
*      FIRST          *10     /"FIRST"/,  
*      PRIOR          *10     /"PRIOR"/,  
*      NEXT           *10     /"NEXT"/,  
*      BEGIN          *9      /"BEGINNING"/,  
*      AUDIT          *10     /"AUDIT"/,  
*      UPDATE         *6      /"UPDATE"/,  
*      SPACE          *1      /" "/,  
*      COND           *9      /"EMP-NO=11"/,  
*      DATREQ         *40     ,  
*      DATBUF         *22     ,  
*      ITMNAM         *13     ,  
*      MSG            *70     ,  
*      EXPR3          *22
```

```
LOGICAL  RSLT,  
*        EXPR4
```

```
DOUBLE PRECISION  EXPR2
```

```
REAL  EXPR1
```

```
*****  
*                               OPENING THE DATABASE                               *  
*****
```

```
RSLT = DBOPEN(UPDATE)  
IF (RSLT) CALL EXCEPT
```

```
*****  
*                               DELETING A RECORD                               *  
*****
```

```
RSLT = DBBTR(AUDIT)  
IF (RSLT) CALL EXCEPT
```

```
RSLT = DBDEL(FIRST,DSNAME,COND)  
IF (RSLT) CALL EXCEPT
```

```
RSLT = DBETR(AUDIT)  
IF (RSLT) CALL EXCEPT
```

## FORTRAN77 Application Program Using the Interpretive Interface (cont.)

---

```
*****
*                                     CREATING A RECORD                       *
*****

      RSLT = DBCR(DSNAME,SPACE)
      IF (RSLT) CALL EXCEPT
*****
*                                     FILLING IN A RECORD                       *
*****

*
*      USING DBPUTREAL ENTRY POINT
*
      ITMNAM = "EMP-NO"
      EXPR1 = 11
      RSLT = DBPUTR(DSNAME,ITMNAM,EXPR1)
      IF (RSLT) CALL EXCEPT

      ITMNAM = "EMP-SALARY"
      EXPR1 = 4500.95
      RSLT = DBPUTR(DSNAME,ITMNAM, 2 * EXPR1)
      IF (RSLT) CALL EXCEPT

*
*      USING DBPUTDOUBLE ENTRY POINT
*
      ITMNAM = "EMP-SSN"
      EXPR2 = 573230911
      RSLT = DBPUTD(DSNAME,ITMNAM, EXPR2 - 11)
      IF (RSLT) CALL EXCEPT

*
*      USING DBDATA ENTRY POINT
*
      DATREQ = "PUT REAL EMP EMP-DATE-HIRED"
      DATBUF = "811223"
      RSLT = DBDATA(DATREQ,DATBUF)
      IF (RSLT) CALL EXCEPT

      DATREQ = "PUT CHARACTER EMP EMP-LNAME"
      DATBUF = "STACK"
      RSLT = DBDATA(DATREQ,DATBUF)
      IF (RSLT) CALL EXCEPT

*
*      USING DBPUTCHARACTER ENTRY POINT
*
      ITMNAM = "EMP-FNAME"
      EXPR3 = "CATHY"
      RSLT = DBPUTC(DSNAME,ITMNAM,EXPR3(1:5) // "LINA")
      IF (RSLT) CALL EXCEPT
```

# FORTRAN77 Application Program Using the Interpretive Interface (cont.)

```

*
*   USING DBPUTLOGICAL ENTRY POINT
*
  ITMNAM = "EMP-SEC-CLEAR"
  EXPR4 = .TRUE.
  RSLT = DBPUTL(DSNAME,ITMNAM,.NOT. EXPR4)
  IF (RSLT) CALL EXCEPT

*****
*                               RECORD SETUP IN USER WORK AREA                               *
*****
*
*   EMP   EMP   EMP   EMP   EMP   EMP   EMP   *
*   NO   LNAME  FNAME  SSN   DATE-HIRED  SEC-CLEAR  SALARY  *
*-----*
*  22   STACK  CATHY  573230900  811223   FALSE     9001.9  *
*****

*****
*                               STORING A RECORD                                       *
*****

  RSLT = DBBTR(AUDIT)
  IF (RSLT) CALL EXCEPT

  RSLT = DBSTORE(DSNAME)
  IF (RSLT) CALL EXCEPT

  RSLT = DBETR(AUDIT)
  IF (RSLT) CALL EXCEPT

*****
*                               RE-CREATING A RECORD                               *
*****

  RSLT = DBRECR(DSNAME,SPACE)
  IF (RSLT) CALL EXCEPT

  ITMNAM = "EMP-NO"
  EXPR1 = 12
  RSLT = DBPUTR(DSNAME,ITMNAM,EXPR1)
  IF (RSLT) CALL EXCEPT

  ITMNAM = "EMP-SSN"
  EXPR2 = 549273281
  RSLT = DBPUTD(DSNAME,ITMNAM, EXPR2 - 12)
  IF (RSLT) CALL EXCEPT

  DATREQ = "PUT STRING EMP EMP-LNAME"
  DATBUF = "BLACK"
  RSLT = DBDATA(DATREQ,DATBUF)
  IF (RSLT) CALL EXCEPT

```

## FORTTRAN77 Application Program Using the Interpretive Interface (cont.)

---

```
ITMNAM = "EMP-FNAME"  
EXPR3 = "SARAH"  
RSLT = DBPUTC(DSNAME,ITMNAM,EXPR3(1:5) // "LEE")  
IF (RSLT) CALL EXCEPT
```

```
RSLT = DBBTR(AUDIT)  
IF (RSLT) CALL EXCEPT
```

```
RSLT = DBSTORE(DSNAME)  
IF (RSLT) CALL EXCEPT
```

```
RSLT = DBETR(AUDIT)  
IF (RSLT) CALL EXCEPT
```

```
*****  
*                               SETTING DATA SET TO THE BEGINNING                               *  
*****
```

```
RSLT = DBSET(DSNAME,BEGIN)  
IF (RSLT) CALL EXCEPT
```

```
*****  
*                               FINDING A RECORD                                           *  
*****
```

```
RSLT = DBFIND(FIRST,DSNAME,COND)  
IF (RSLT) CALL EXCEPT
```

```
*****  
*                               LOCKING A RECORD                                           *  
*****
```

```
RSLT = DBLOCK(FIRST,DSNAME,COND)  
IF (RSLT) CALL EXCEPT
```

```
*****  
*                               VERIFYING ITEMS IN A RECORD                               *  
*****
```

```
*  
*  
*  
*  
*
```

```
    USING DBGETREAL ENTRY POINT
```

```
ITMNAM = "EMP-NO"  
RSLT = DBGETR(DSNAME,ITMNAM,EXPR1)  
IF (RSLT) CALL EXCEPT
```

```
ITMNAM = "EMP-SALARY"  
RSLT = DBGETR(DSNAME,ITMNAM,EXPR1)  
IF (RSLT) CALL EXCEPT
```

## FORTRAN77 Application Program Using the Interpretive Interface (cont.)

---

```
*
*
*      USING DBGETDOUBLE ENTRY POINT
*
ITMNAM = "EMP-SSN"
RSLT = DBGETD(DSNAME,ITMNAM,EXPR2)
IF (RSLT) CALL EXCEPT
*
*
*      USING DBDATA ENTRY POINT
*
DATREQ = "GET REAL EMP EMP-DATE-HIRED"
RSLT = DBDATA(DATREQ,DATBUF)
IF (RSLT) CALL EXCEPT

DATREQ = "GET CHARACTER EMP EMP-LNAME"
RSLT = DBDATA(DATREQ,DATBUF)
IF (RSLT) CALL EXCEPT
*
*
*      USING DBGETCHARACTER ENTRY POINT
*
ITMNAM = "EMP-FNAME"
RSLT = DBGETC(DSNAME,ITMNAM,EXPR3)
IF (RSLT) CALL EXCEPT
*
*
*      USING DBGETLOGICAL ENTRY POINT
*
ITMNAM = "EMP-SEC-CLEAR"
RSLT = DBGETL(DSNAME,ITMNAM,EXPR4)
IF (RSLT) CALL EXCEPT

*****
*                               FREEING A RECORD                               *
*****

RSLT = DBFREE(DSNAME)
IF (RSLT) CALL EXCEPT

*****
*                               CAUSING AN EXCEPTION                               *
*****

RSLT = DBSET(DSNAME,BEGIN)
IF (RSLT) CALL EXCEPT

RSLT = DBFIND(PRIOR,DSNAME,SPACE)
IF (RSLT) CALL EXCEPT
```

## FORTRAN77 Application Program Using the Interpretive Interface (cont.)

---

```
*****
*                                     USING DBVERB                               *
*****
*
*      USING DBVERB ON A SET STATEMENT
*
RSLT = DBVERB("SET EMP TO BEGINNING")
IF (RSLT) CALL EXCEPT
*
*      USING DBVERB ON A SEARCH STATEMENT
*
RSLT = DBVERB("FIND FIRST EMP AT EMP-NO=11")
IF (RSLT) CALL EXCEPT
*
*      USING DBVERB ON A STATUS STATEMENT
*
RSLT = DBVERB("SET EMP TO BEGINNING")
IF (RSLT) CALL EXCEPT
RSLT = DBVERB("FIND PRIOR EMP")
RSLT = DBVERB("DMSTATUS")
CALL DBEXCN(RSLT,MSG)
CALL DBEXCT(RSLT,MSG)

*****
*                                     CLOSING THE DATABASE                       *
*****

RSLT = DBCLOSE( )
IF (RSLT) CALL EXCEPT

END
```

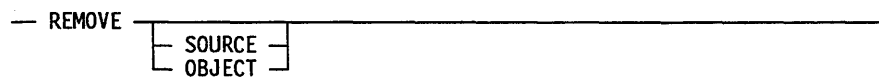


# Appendix D

## Understanding Railroad Diagrams

### What Are Railroad Diagrams?

Railroad diagrams are diagrams that show you the rules for putting words and symbols together into commands and statements that the computer can understand. These diagrams consist of a series of paths that show the allowable structure, constants, and variables for a command or a statement. Paths show the order in which the command or statement is constructed. Paths are represented by horizontal and vertical lines. Many railroad diagrams have a number of different paths you can take to get to the end of the diagram. For example:



If you follow this railroad diagram from left to right, you will discover three acceptable commands. These commands are

- REMOVE
- REMOVE SOURCE
- REMOVE OBJECT

If all railroad diagrams were this simple, this explanation could end here. However, because the allowed ways of communicating with the computer can be complex, railroad diagrams sometimes must also be complex.

Regardless of the level of complexity, all railroad diagrams are visual representations of commands and statements. Railroad diagrams are intended to

- Show the mandatory items.
- Show the user-selected items.
- Present the order in which the items must appear.
- Show the number of times an item can be repeated.
- Show the necessary punctuation.

To familiarize you with railroad diagrams, this explanation describes the elements of the diagrams and provides examples.

Some of the actual railroad diagrams you will encounter might be more complex. However, all railroad diagrams, simple or complex, follow the same basic rules. They



## Understanding Railroad Diagrams

---

all consist of paths that represent the allowable structure, constants, and variables for commands and statements.

By following railroad diagrams, you can easily understand the correct syntax for commands and statements. Once you become proficient in the use of railroad notation, the diagrams serve as quick references to the commands and statements.

### Constants and Variables

A constant is an item that cannot be altered. You must enter the constant as it appears in the diagram, either in full or as an allowable abbreviation. If a constant is partially underlined, you can abbreviate the constant by entering only the underlined letters. In addition to the underlined letters, any of the remaining letters can be entered. If no part of the constant is underlined, the constant cannot be abbreviated. Constants can be recognized by the fact that they are never enclosed in angle brackets (< >) and are in uppercase letters.

A variable is an item that represents data. You can replace the variable with data that meets the requirements of the particular command or statement. When replacing a variable with data, you must follow the rules defined for the particular command or statement. Variables appear in railroad diagrams enclosed in angle brackets.

In the following example, BEGIN and END are constants while <statement list> is a variable. The constant BEGIN can be abbreviated since it is partially underlined. Valid abbreviations for BEGIN are BE, BEG, and BEGI.

— BEGIN —<statement list>— END —————|

### Constraints

Constraints are used in a railroad diagram to control progression through the diagram. Constraints consist of symbols and unique railroad diagram line paths. They include

- Vertical bars
- Percent signs
- Right arrows
- Required items
- User-selected items
- Loops
- Bridges

A description of each item follows.

## Vertical Bar

The vertical bar symbol (|) represents the end of a railroad diagram and indicates the command or statement can be followed by another command or statement.

— SECONDWORD — ( —<arithmetic expression>— ) —————|

## Percent Sign

The percent sign (%) represents the end of a railroad diagram and indicates the command or statement must be on a line by itself.

— STOP —————%

## Right Arrow

The right arrow symbol (>) is used when the railroad diagram is too long to fit on one line and must continue on the next. A right arrow appears at the end of the first line and another right arrow appears at the beginning of the next line.

— SCALERIGHT — ( —<arithmetic expression>— , —————>  
-><arithmetic expression>— ) —————|

## Required Items

A required item can be either a constant, a variable, or punctuation. A required item appears as a single entry, by itself or with other items, on a horizontal line. Required items can also exist on horizontal lines within alternate paths or nested (lower-level) diagrams. If the path you are following contains a required item, you must enter the item in the command or statement; the required item cannot be omitted.

In the following example, the word EVENT is a required constant and <identifier> is a required variable:

— EVENT —<identifier>—————|

## User-Selected Items

User-selected items appear one below the other in a vertical list. You can choose any one of the items from the list. If the list also contains an empty path (solid line), none of the choices are required. A user-selected item can be either a constant, a variable, or punctuation. In the following railroad diagram, either the plus sign (+) or minus sign (-) can be entered before the required variable <arithmetic expression>, or the symbols can be disregarded because the diagram also contains an empty path.

+
-

 <arithmetic expression>—————|

## Understanding Railroad Diagrams

---

### Loop

A loop represents an item or group of items that you can repeat. A loop can span all or part of a railroad diagram. It always consists of at least two horizontal lines, one below the other, connected on both sides by vertical lines. The top line is a right-to-left path that contains information about repeating the loop.

Some loops include a return character. A return character is a character — often a comma (,) or semicolon (;) — required before each repetition of a loop. If there is no return character, the items must be separated by one or more blank spaces.

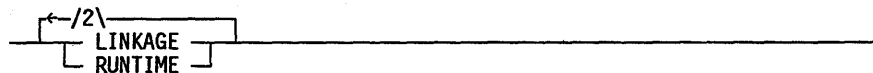
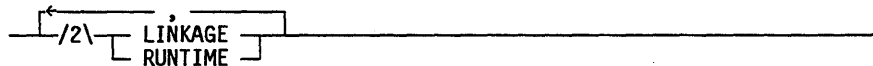


### Bridge

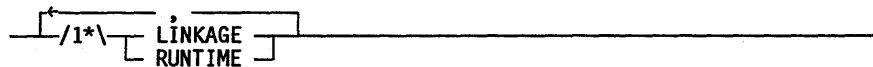
Sometimes a loop also includes a bridge, which is used to show the maximum number of times the loop can be repeated. The bridge can precede the contents of the loop, or it can precede the return character (if any) on the upper line of the loop.

The bridge determines the number of times you can cross that point in the diagram. The bridge is an integer enclosed in sloping lines (/ \). Not all loops have bridges. Those that do not can be repeated any number of times until all valid entries have been used.

In the first bridge example, you can enter LINKAGE or RUNTIME no more than two times. In the second bridge example, you can enter LINKAGE or RUNTIME no more than three times.



In some bridges an asterisk (\*) follows the number. The asterisk means that you must select one item from the group.



The following figure shows the types of constraints used in railroad diagrams.

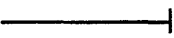

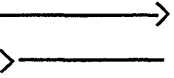
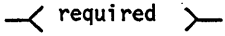
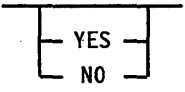
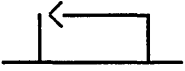
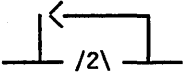
SYMBOL/PATH	EXPLANATION
	Vertical bar. Indicates that the command or statement can be followed by another command or statement.
	Percent sign. Indicates that the command or statement must be on a line by itself.
	Right arrow. Indicates that the diagram occupies more than one line.
	Required items. Indicates the constants, variables, and punctuation that must be entered in a command or statement.
	User-selected items. Indicates the items that appear one below the other in a vertical list. You select which item or items to include.
	A loop. Indicates an item or group of items that can be repeated.
	A bridge. Indicates the maximum number of times a loop can be repeated.

Figure D-1. Railroad Constraints

## Following the Paths of a Railroad Diagram

The paths of a railroad diagram lead you through the command or statement from beginning to end. Some railroad diagrams have only one path, while others have several alternate paths. The following railroad diagram indicates there is only one path that requires the constant LINKAGE and the variable <linkage mnemonic> :

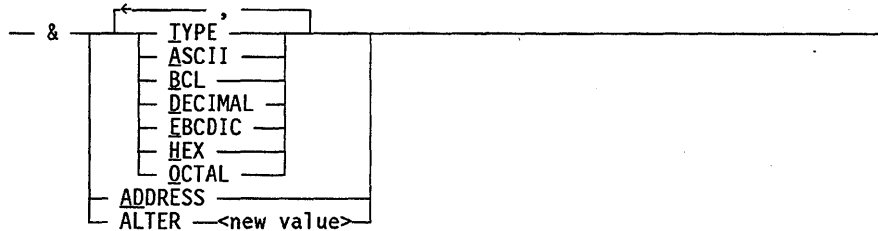
— LINKAGE —<linkage mnemonic>—————|

Alternate paths provide choices in the construction of commands and statements. Alternate paths are provided by loops, user-selected items, or a combination of both. More complex railroad diagrams can consist of many alternate paths, or nested (lower-level) diagrams, that show a further level of detail.

For example, the following railroad diagram consists of a top path and two alternate paths. The top path includes an ampersand (&) and the constants (that are

## Understanding Railroad Diagrams

user-selected items) in the vertical list. These constants are within a loop that can be repeated any number of times until all options have been selected. The first alternate path requires the ampersand and the required constant ADDRESS. The second alternate path requires the ampersand followed by the required constant ALTER and the required variable <new value>.

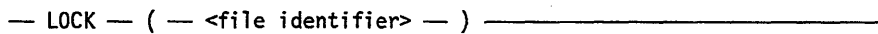


## Railroad Diagram Examples with Sample Input

The following examples show five railroad diagrams and possible command and statement constructions based on the paths of these diagrams.

### Example 1

<lock statement>



#### Sample Input

LOCK (FILE4)

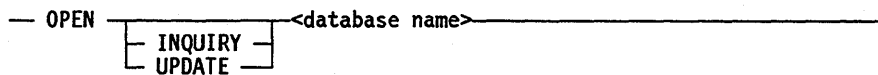
#### Explanation

LOCK is a constant and cannot be altered. Because no part of the word is underlined, the entire word must be entered.

The parentheses are required punctuation and FILE4 is a sample file identifier.

### Example 2

<open statement>



#### Sample Input

OPEN DATABASE1

#### Explanation

The constant OPEN is followed by the variable DATABASE1, which is a database name.

The railroad diagram shows two user-selected items, INQUIRY and UPDATE. However, because there is an empty path (solid line), these entries are not required.

OPEN INQUIRY DATABASE1

The constant OPEN is followed by the user-selected constant INQUIRY and the variable DATABASE1.

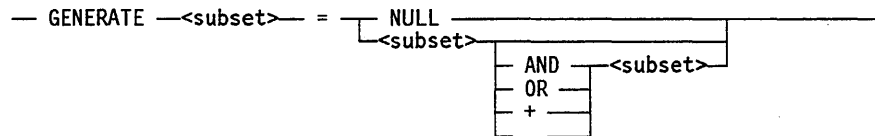
*continued*

*continued*

Sample Input	Explanation
OPEN UPDATE DATABASE1	The constant OPEN is followed by the user-selected constant UPDATE and the variable DATABASE1.

### Example 3

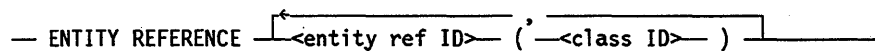
<generate statement>



Sample Input	Explanation
GENERATE Z = NULL	The GENERATE constant is followed by the variable Z, an equal sign (=), and the user-selected constant NULL.
GENERATE Z = X	The GENERATE constant is followed by the variable Z, an equal sign, and the user-selected variable X.
GENERATE Z = X AND B	The GENERATE constant is followed by the variable Z, an equal sign, the user-selected variable X, the AND command (from the list of user-selected items in the nested path), and a third variable, B.
GENERATE Z = X + B	The GENERATE constant is followed by the variable Z, an equal sign, the user-selected variable X, the plus sign (from the list of user-selected items in the nested path), and a third variable, B.

### Example 4

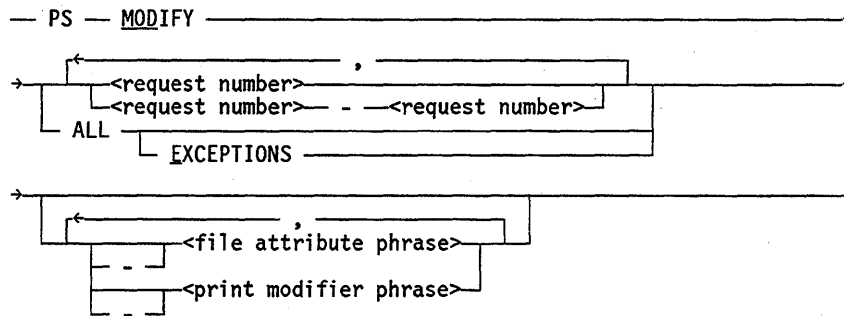
<entity reference declaration>



Sample Input	Explanation
ENTITY REFERENCE ADVISOR1 (INSTRUCTOR)	The required item ENTITY REFERENCE is followed by the variable ADVISOR1 and the variable INSTRUCTOR. The parentheses are required.
ENTITY REFERENCE ADVISOR1 (INSTRUCTOR), ADVISOR2 (ASST_INSTRUCTOR)	Because the diagram contains a loop, the pair of variables can be repeated any number of times.

# Understanding Railroad Diagrams

## Example 5



### Sample Input

PS MODIFY 11159

PS MODIFY 11159,11160,11163

PS MOD 11159-11161 DESTINATION =  
"LP7"

PS MOD ALL EXCEPTIONS

### Explanation

The constants PS and MODIFY are followed by the variable 11159, which is a request number.

Because the diagram contains a loop, the variable 11159 can be followed by a comma, the variable 11160, another comma, and the final variable 11163.

The constants PS and MODIFY are followed by the user-selected variables 11159-11161, which are request numbers, and the user-selected variable DESTINATION = "LP7", which is a file attribute phrase. Note that the constant MODIFY has been abbreviated to its minimum allowable form.

The constants PS and MODIFY are followed by the user-selected constants ALL and EXCEPTIONS.

# Glossary

In this glossary, definitions taken from outside sources are preceded by an abbreviation enclosed in parentheses. Unless otherwise noted, all definitions that pertain to COBOL also pertain to COBOL74 and COBOL85.

## A

### **abort**

To terminate an active program or session abnormally and, sometimes, to attempt to restart it.

### **ABSN**

*See* audit block serial number.

### **access**

(1) To perform an action on an object. Possible actions depend on the type of object; for example, interrogating or assigning a value to a variable, reading from or writing to a file, or invoking a procedure. (2) A logical index structure that defines the physical ordering of records in direct, ordered, and random data sets. An access functions like a set, but no physical file is associated with an access.

### **access mode**

The manner in which records are to be operated on within a file. The two possible access modes are random and sequential.

### **Accessroutines**

Routines that perform all physical and logical management of a database and allow many users to access the database concurrently. Each data management statement that a user language program executes invokes a portion of the Accessroutines to perform all file management functions that the statement requires.

### **active query**

In database management, a query that the system can process. All queries activated within transaction state are deactivated at the end of transaction state.

### **ACTUALNAME clause**

A clause that permits the renaming of a library entry point within a user program by using a different procedure identifier to link to the original library entry point name. This clause is used in ALGOL, FORTRAN77, and PL/1 application programs when a procedure is declared to be an entry point of a particular library.

### **ADDS**

*See* Advanced Data Dictionary System.

### **Advanced Data Dictionary System (ADDS)**

A software product that allows for the centralized definition, storage, and retrieval of data descriptions.



## Glossary

---

### **afterimage**

The image of a database record in the audit trail after an update operation is performed.

### **ALGOL**

Algorithmic language. A structured, high-level programming language that provides the basis for the stack architecture of the Unisys A Series systems. ALGOL was the first block-structured language developed in the 1960s and served as a basis for such languages as Pascal and Ada. It is still used extensively on A Series systems, primarily for systems programming.

### **alpha data item of usage Kanji**

A database or file item that is declared in either the Data and Structure Definition Language (DASDL) or a file description as being ALPHA USAGE IS KANJI.

### **alpha item**

A data item that stores alphanumeric information (letters, numbers, special characters, and blanks) as EBCDIC characters. An alpha item cannot be used in calculations.

### **alphabetic character**

A character that belongs to the following set of letters in either uppercase or lowercase: A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z, and the space (blank).

### **alphanumeric character**

Any character in the computer's character set.

### **ANSI**

American National Standards Institute. A nongovernmental, nonprofit organization that is the central body for coordinating voluntary standards in the United States. ANSI also serves as the United States member of the International Standards Organization (ISO).

### **arithmetic expression**

An expression containing any of the following: a numeric variable, a numeric elementary item, a numeric literal, identifiers and literals separated by arithmetic operators, two arithmetic expressions separated by an arithmetic operator, or an arithmetic expression enclosed in parentheses.

### **arithmetic function**

A function containing calculations that produce a numeric result based on one or more records.

### **attribute setting entry point**

In the DMINTERPRETER library, an entry point that restricts the number of times the DMINTERPRETER library can call the Accessroutines to complete a find or lock operation.

### **audit block serial number (ABSN)**

A number associated with a block in the audit file. The first block in the first audit file is noted as ABSN 0. The ABSN is incremented by one through the last audit file.

**audit trail**

A file produced by the Accessroutines that contains various control records and a sequence of before-update and after-update record images resulting from changes to the database. The audit trail is used to recover the database and supply restart information to programs after a hardware or software failure has occurred.

**audited database**

A database that stores a record of changes (called the audit trail), which can be used for database recovery if a hardware or software failure occurs.

**automatic subset**

A subset declared with a condition that specifies which members of the data set are to be included in the subset. Entries are automatically inserted into or removed from the subset when records are added to or deleted from the data set.

**B**

**back out**

To undo changes made against a database and to roll back the progress of one or more transactions to a previously consistent state.

**beforeimage**

The image of a database record in the audit trail before an update operation is performed.

**binary search**

A quick method of searching for records that are in a known sequence by successively halving the area to be searched.

**block**

(1) A group of physically adjacent records that can be transferred to or from a physical device as a group. (2) A program, or a part of a program, that is treated by the processor as a discrete unit. Examples are a procedure in ALGOL, a procedure or function in Pascal, a subroutine or function in FORTRAN, or a complete COBOL program.

**Boolean**

Pertaining to variables, data items, and attributes having a value of TRUE or FALSE.

**Boolean item**

A data item that stores information having a logical value of TRUE or FALSE.

**C**

**call**

A programmatic request for another procedure or program to execute.

**CALL statement**

In COBOL, the *CALL nonnumeric literal* statement is the only construct that enables a program to call a library entry point. The CALL statement can pass required

## Glossary

---

parameters in an optional USING clause and receive a result in an optional GIVING clause.

### **CANDE**

*See* Command and Edit.

### **CHANGE ATTRIBUTE statement**

In COBOL, the statement that is the only construct that can be used to change library attributes. When programming for the interpretive interface, this statement is used to change the TITLE attribute of the library to the actual name of the library object code file.

### **character**

The actual or coded representation of a digit, letter, or special symbol in display form.

### **COBOL**

Common Business-Oriented Language. A widely used, procedure-oriented language intended for use in solving problems in business data processing. The main characteristics of COBOL are the easy readability of programs and a considerable degree of machine independence. COBOL is the most widely used procedure-oriented language.

### **COBOL74**

A version of the COBOL language that is compatible with the American National Standard X3.23-1974.

### **COBOL85**

The latest version of the COBOL language. This version is compatible with the American National Standard X3.23-1985.

### **code file**

*See* object code file, source file.

### **Command and Edit (CANDE)**

A time-sharing message control system (MCS) that enables a user to create and edit files, and to develop, test, and execute programs, interactively.

### **Communications Management System (COMS)**

A general message control system (MCS) that controls online environments on A Series systems. COMS can support the processing of multiprogram transactions, single-station remote files, and multistation remote files.

### **compile time**

The time during which a compiler analyzes program text and generates an object code file.

### **complex key**

A key composed of multiple key items.

### **COMS**

*See* Communications Management System.

**condition**

In COBOL, a status for which a truth value can be determined at execution time. The term *condition* (condition-1, condition-2, and so forth) implies a conditional expression consisting of either a simple condition optionally enclosed in parentheses or a combined condition consisting of a combination of simple conditions, logical operators, and parentheses, for which a truth value can be determined. *See also* selection expression.

**control file**

A file containing data file coordination information, audit control information, and dynamic database parameter values.

**control item**

A count item, population item, or record-type item.

**controlpoint**

A feature that limits the amount of audit information scanned by halt/load and Abort recovery. The controlpoint is performed when the last process leaves transaction state. Halt/load and Abort recovery do not process any audit images prior to the next to last controlpoint in the audit.

**count item**

A control item that contains a system-maintained count of the number of counted links that refer to a record.

**current path**

A logical reference into a data set or set from which the current record was retrieved.

**current record**

The actual data set record that a program is currently referencing. Each data set has a current record, which is contained in the user work area.

**current record area**

*See* user work area.

**D****DASDL**

*See* Data and Structure Definition Language.

**Data and Structure Definition Language (DASDL)**

The language used to describe a database logically and physically, and to specify criteria to ensure the integrity of data stored in the database. DASDL is the source language that is input to the DASDL compiler, which creates or updates the database description file from the input.

**data independence**

In data management, the property that establishes the ability to change the structural format of a database without requiring changes to or recompilation of unaffected application programs that use the database.

## Glossary

---

### **data item**

A field in a database record or transaction format that contains a particular type of information.

### **data management**

(1) The operating system function of placing and retrieving data in storage and protecting its security and integrity. (2) Data administration.

### **Data Management System (DMS)**

The system responsible for storing and retrieving data while protecting data security and integrity.

### **Data Management System II (DMSII)**

A specialized system software package used to describe a database and maintain the relationships among the data elements in the database.

### **data set**

A collection of related data records stored in a file on a random-access storage device. A data set is similar to a conventional file. It contains data items and has logical and physical properties similar to files. However, unlike conventional files, data sets can contain other data sets, sets, and subsets.

### **data set record**

A record contained in a data set, which can be accessed through a DMSII statement.

### **data transfer entry point**

A type of entry point peculiar to the DMINTERPRETER library whereby an item needed from a data set record is moved individually to or from a variable or expression that has been declared in the application program.

### **data type**

An interpretation applied to a string of bits. Data types can be classified as *structured* or *scalar*. Structured data types are collections of individual data items of the same or different data types, such as arrays and records. Scalar data types include real, integer, double precision, complex, logical (also called Boolean), character, pointer, and label. Most programming languages provide a declaration statement or a standard convention to indicate the data type of a variable.

### **database (DB)**

An integrated, centralized system of data files and program utilities designed to support an application. The data sets and associated index structures are defined by a single description. Ideally, all the permanent data pertinent to a particular application resides in a single database. The database is considered a global entity that several applications can access and update concurrently.

### **database administrator (DBA)**

The person or group of people responsible for planning, designing, implementing, and maintaining a database.

### **database definition**

A description of the logical and physical structures of a database.

**database equation**

The equation that refers to three operations: the specification of database titles during compilation, the run-time manipulation of database titles, and the creation of a Work Flow Language (WFL) task equation that overrides compiled-in titles by implicitly assigning a value to the DATABASE task attribute.

**database management system (DBMS)**

The software used to store, retrieve, update, report on, and protect data in a database.

**database name**

The unique identifier of a particular database. The rules for constructing a database name are the same as those for constructing a file name.

**database stack (DBS)**

A stack that contains all the information necessary for the Access routines to manage a database.

**DB**

*See* database.

**DBA**

*See* database administrator.

**DBMS**

*See* database management system.

**DBS**

*See* database stack.

**deadlock**

In data management, a situation in which two or more programs have locked records and are also attempting to lock records held by each other.

**declaration**

A programming language construct used to identify an object, such as a type or variable, to the compiler. A declaration can be used to associate a data type with the object so that the object can be used in a program.

**description file**

The file produced by the Data and Structure Definition Language (DASDL) or Transaction Formatting Language (TFL) compiler that contains information used when compiling all tailored software and all DMSII user-language programs for a particular database or transaction base.

**direct data set**

A collection of related data records stored in a file. These records are maintained in key value order. One unsigned numeric data item in the record is designated as the key item.

**directory**

(1) In Data Management System II (DMSII), a file with the layout for each field of the record that it describes. A directory describes the layout of records within a file. (2) In

## Glossary

---

the Advanced Data Dictionary System (ADDS), a unique identifier by which one or more entities can be grouped.

### **directory name**

A name used to refer to a group of files whose file names are identical to the directory name, except that the file names have at least one additional node following the directory name.

### **disjoint**

Pertaining to a data set, set, or subset when it is not contained in another data set. *Contrast with* embedded.

### **DMINTERPRETER library**

The object code file within the interpretive interface that enables an application program to access a DMSII database.

### **DMS**

*See* Data Management System.

### **DMSII**

*See* Data Management System II.

### **DMSII recovery**

A database routine that is initiated after a hardware, software, or operations failure while a database is in the update mode. DMSII recovery backs out any partially completed transactions by applying audit-trail images to the database to restore it to its proper state. It also passes restart information to the programs accessing the database.

### **DMSUPPORT library**

An object code file containing the procedures that are tailored for a particular database.

### **double precision**

Pertaining to an arithmetic value that is represented internally as a signed-magnitude mantissa and exponent and is contained in two words.

## **E**

### **EBCDIC**

Extended Binary Coded Decimal Interchange Code. An 8-bit code representing 256 graphic and control characters that are the native character set of most mainframe systems.

### **embedded**

Pertaining to a data set, set, or subset contained within another data set. A record of an embedded structure must be accessed through the *master* data set in which it is embedded. *Contrast with* disjoint.

### **end of Kanji (EOK) character**

A character that signals the end of a Kanji character string.

**entry point**

A procedure or function that is a library object.

**EOK**

See end of Kanji character.

**ERGO**

See Extended Retrieval with Graphic Output.

**exception**

In data management, an error result returned to an application program by the data management software, explaining the reason a requested database operation was not performed.

**exception handling entry point**

In the DMINTERPRETER library, an entry point that aids the programmer in handling and interpreting exceptions and errors.

**exception result**

*Synonym for result descriptor and exception word.*

**exception word**

*Synonym for result descriptor and exception result.*

**execution time**

The time during which an object code file is executed. *Synonym for run time and, in COBOL, object time.*

**export**

To send or carry to an outside entity the original significance or function.

**expression**

A combination of operands and operators that results in the generation of a single value.

**Extended Retrieval with Graphic Output (ERGO)**

A program for Data Management System II (DMSII) database and file access, and for report generation.

**F**

**field item**

A data item that can contain an unsigned integer or Boolean values.

**Field Trouble Report (FTR)**

An obsolete term; see User Communication Form.

**file equation**

A mechanism for specifying the values of file attributes when a program is compiled or executed. A file equation implicitly assigns a value to the FILECARDS task attribute.



## Glossary

---

### **file-equate**

To specify the file title and other file parameters that are different from the defaults provided by the source program.

### **FORTRAN**

Formula Translation. A high-level, structured programming language intended primarily for scientific use.

### **FORTRAN77**

A version of the FORTRAN language that is compatible with the ANSI X3.9-1978 standard.

### **FTR**

An acronym for the obsolete term Field Trouble Report. *See* User Communication Form.

## **G**

### **GIVING clause**

In COBOL, an optional clause in the *CALL non-numeric literal* statement that specifies a variable as the destination for the integer value

### **global data item**

A data item, group item, or population item that is not a part of any data set. Global data items generally contain information such as control totals, hash totals, and populations that apply to the entire database.

### **group item**

A collection of data items that can be viewed as a single data item.

## **H**

### **halt/load**

A system-initialization procedure that temporarily halts the system and loads the master control program (MCP) from a disk to main memory.

### **hashing algorithm**

An algorithm used in constructing and maintaining hash tables, which modify record keys to produce the addresses of the record keys in a structure.

## **I**

### **ID**

*See* identifier.

### **identifier (ID)**

(1) A label. (2) One node of a file name.

### **import**

To bring in from an outside source and still retain the original significance or function.

**IN LIBRARY statement**

In FORTRAN77, the statement that associates the logical expression name of an entry point with the library that is known to the application program through the library identifier.

**INCLUDE**

A compiler control option that enables the user to insert an entire file, or sequence range within a file, into a program.

**index sequential structure**

A structure that is a collection of tables arranged hierarchically.

**input/output mapping clause**

A clause that can be used with data management operations to assign the value of a database item to a program variable (input) and/or to assign the value of a program variable to a database item (output).

**inquiry mode**

A database mode in which data can be read but not updated.

**integer**

A whole number.

**internal file name**

The name used to declare a logical file in a program. The internal name of a file is given by the value of its INTNAME file attribute. Work Flow Language (WFL) file equation statements can reference the file by implicitly or explicitly specifying an INTNAME value that matches the INTNAME attribute of a file in a program.

**interpretive interface**

A set of conventions for passing information using the DMINTERPRETER library.

**INTNAME**

See internal file name.

**intrinsic**

A system-supplied program routine for common mathematical and other operations that is loaded onto the system separately. An intrinsic can be invoked by the operating system or user programs.

**intrinsic data set**

A data set created by system designers as part of a Data Management System II (DMSII) software package, as opposed to a data set created by a user.

**invocation**

(1) The syntax used to initiate execution of software. (2) The act that transfers control to the start of a specified procedure, initializes any parameters, and begins the execution of the statements of the procedure. Invocations are of two kinds: entrances and initiations.

**invoke**

(1) To cause to be executed. (2) To cause to be brought into main storage.

## Glossary

---

### item

A field in a database record that contains an individual piece of information and can be referenced by name.

## J

### job

An independent process. The job of a particular task is the independent process that is the eldest ancestor of that task.

## K

### Kanji

The standard Japanese character set for information exchange. Each Kanji character is 2 bytes (16 bits) in length and takes two positions on a form image.

### Kanji alpha string

A string, delimited by quotation marks, that is further delimited by a start of Kanji (SOK) character (48"2B") and an end of Kanji (EOK) character (48"2C").

### Kanji character literal

A character string bounded on the left by the separator NC" and on the right by a quotation mark ("). The string contains Kanji characters between the beginning and ending quotation marks.

### key

(1) A field used to locate or identify a record in an indexed file. (2) A field in a record that is used to sort a file. (3) *See also* key item.

### key field

*See* key.

### key item

A data item or group item that serves as a retrieval key for a set, subset, or access.

## L

### language extension

A syntactical addition to a standard language compiler that enables the compiler to interface directly with Unisys products.

### library

(1) A collection of one or more named routines or library objects that are stored in a file and can be accessed by other programs. (2) A program that exports objects for use by user programs.

### library directory

A memory structure associated with a library process stack that describes the objects exported by the library process.

**library identifier**

An internal name within an application program that represents the library code file being accessed.

**linear search**

A method for searching a set of information by examining each element of the set one at a time.

**link item**

A field that enables one data set record to refer to another.

**literal**

A character string whose value is implied by the ordered set of characters that compose the string.

**logical database**

A collection of structures declared in the Data and Structure Definition Language (DASDL) that provide a view of the database, enforce structure-level security, and achieve data independence. When a logical database is declared in DASDL, the data sets, sets, subsets, and remaps to be included in the database are listed.

**logical expression**

In FORTRAN77, a rule for computing a value corresponding to *.TRUE.* or *.FALSE.* . Logical expressions consist of any valid combination of logical operands, logical operators, and parentheses.

**logical function**

In FORTRAN, a function that returns a value of *.TRUE.* or *.FALSE.* .

## M

**major key**

The first key in a complex key.

**manual subset**

A subset that has no condition specifying which data set records are to be included in the subset. The user must add and delete manual subset entries, using the *INSERT* and *REMOVE* statements.

**master**

A data set or record that contains one or more embedded data sets or records. In DMSII, *synonym for master, parent, owner.*

**master control program (MCP)**

The central program of the A Series operating system. The term applies to any master control program that Unisys may release for A Series systems.

**MCP**

*See master control program.*

## Glossary

---

### **MCS**

See message control system.

### **member**

A record of a data set.

### **message control system (MCS)**

A program that controls the flow of messages between terminals, application programs, and the operating system. MCS functions can include message routing, access control, audit and recovery, system management, and message formatting.

### **metatoken**

An item that appears in syntax notation as a variable item.

### **minor key**

Any key in a complex key that is not the major (first) key.

### **mnemonic**

(1) An abbreviation or acronym that is used to assist the human memory. (2) A programmer-supplied word associated with a specific function name. (3) A character or group of characters intended to serve as a mnemonic.

## **N**

### **nonkey item**

An item in the data set being indexed that is not a key item in the selected set.

### **null string**

An empty or zero-length string.

### **null value**

The value contained in an item that does not contain valid information.

### **numeric item**

A data item whose description restricts its contents to a value represented by numeric characters.

## **O**

### **object code file**

A file produced by a compiler when a program is compiled successfully. The file contains instructions in machine-executable object code.

### **object time**

In COBOL, the time during which an object program is executed. *Synonym for run time, execution time.*

**OCCURS clause**

In a Data and Structure Definition Language (DASDL) data item declaration, a clause that establishes an ordered (subscripted) collection of data items with identical attributes.

**ordered**

Pertaining to an item maintained in a user-specified sequence.

**owner**

*See master.*

**P**

**parent**

A process that owns the critical block of a dependent process. If the parent exits the critical block before the dependent process terminates, the dependent process is discontinued. *See also master.*

**path**

(1) The route that must be traced from a directory to a subdirectory, or through a series of subdirectories, to find a file. (2) A specific location within the logical ordering of a data set, set, subset, or access.

**physical database**

An entire database as it is stored on a disk. Whereas a logical database represents only parts of a database to be used for limited purposes, a physical database is an entire database.

**population**

For disjoint data sets, the number of records in the data set. For embedded data sets, the population is the number of records in the embedded data set owned by the current master.

**Q**

**qualification**

The specification of the data set that owns an item. Qualification is usually used when several data sets contain an item with the same name.

**quiet point**

A time during which there are no transactions in progress for a database. *See also syncpoint.*

**R**

**railroad diagram**

A graphic representation of the syntax of a command or statement.

## Glossary

---

### **RDS**

*See* restart data set.

### **real item**

A data item that stores signed or unsigned, fractional or whole values in single-precision, floating-point form.

### **real number**

Any number, including fractions and whole numbers.

### **rebuild**

In database management, a recovery process in which the entire database is loaded from one or more sets of dump tapes. The recovery process then applies the audit trail after-update record images to move the database forward in time.

### **record**

(1) A group of logically related items of data in a file that are treated as a unit. (2) The data read from or written to a file in one execution of a read or write statement in a program.

### **record-type item**

Either of the following: a binary integer value used in conjunction with variable-format data sets to identify the variable-format part, if any, the record contains; or a control item that contains the format number for records in variable-format data sets.

### **recovery**

In data management, a procedure that is initiated following a hardware, software, or operations failure while the database is in update mode. Recovery backs out any partially completed transactions by applying audit-trail images to the database to restore it to a consistent state. In addition, recovery passes restart information to the programs accessing the database.

### **remap**

A logical data set that redefines a physical data set by omitting, reordering, or renaming the items.

### **remote file**

A file with the **KIND** attribute specified as **REMOTE**. A remote file enables object programs to communicate interactively with a terminal.

### **reorganization**

The process of reordering or reformatting data sets, sets, or subsets. Reorganization can restore space in files, reorder data sets for more efficient retrieval, and reformat data set records when items are added, deleted, or changed.

### **restart data set (RDS)**

A data set containing restart records that application programs can access to recover database information after a system failure.

### **restart record**

A record containing user-defined information that enables a user program to restart in response to a particular condition.

**result descriptor**

A 48-bit word that is the standard Data Management System II (DMSII) exception word. A result descriptor includes the exception category and subcategory, and the structure number involved. When an exception occurs, bit 0 in the 48-bit word is set to 1. When an operation is successful, the 48-bit word contains all zeros.

**rollback**

The recovery of a database or transaction base to a consistent state at an earlier point in time.

**RPG**

Report Program Generator. A high-level, commercially oriented programming language used most frequently to produce reports based on information derived from data files.

**run time**

The time during which an object code file or user interface system (UIS) is executed. *Synonym for* execution time and, in COBOL, object time.

**run-time error**

An error occurring during the execution of a program, which causes the system software to terminate execution of that program abnormally.

**S**

**savepoint**

An user-determined point in a program that is located between a beginning transaction marker and its corresponding end transaction marker. While executing the transaction, processing can be specifically backed out to this point.

**schema**

The outline or description of a database. The schema acts as a map for the host system to use when performing any functions on the database or when accessing the database.

**selection expression**

The entire complement of selection criteria used in a FIND, LOCK, or DELETE statement to locate a data set record. The definition of a selection expression encompasses both the select options (FIRST, NEXT, LAST, and PRIOR) and all the variations for the key conditions. *See also* condition.

**Semantic Information Manager (SIM)**

The basis of the InfoExec environment. SIM is a database management system used to describe and maintain associations among data by means of subclass-superclass relationships and linking attributes.

**serial search**

A method for searching a set of information by examining each element of the set one at a time.

**set**

A file of indexes that refers to all the records of a single data set. Sets are automatically maintained by the system. Sets permit access to the records of a data set in some logical



## Glossary

---

sequence and are normally used to optimize certain types of retrievals of the data set records.

### **SIM**

*See* Semantic Information Manager.

### **SOK**

*See* start of Kanji character.

### **source file**

(1) A file in which a source program is stored. (2) A file containing instructions written in a programming language.

### **standard entry point**

In the DMINTERPRETER library, an entry point that performs a function corresponding to a function specified in a Data Management System II (DMSII) user language statement.

### **start of Kanji (SOK) character**

A character that signals the beginning of a Kanji character string.

### **string**

A connected sequence or group of characters.

### **structure**

A data set, set, subset, access, or remap.

### **subitem**

An item that is a member of a group item.

### **subscript**

A number that is an index into an array.

### **subset**

An index structure that is identical to a set, except that the subset need not contain a record for every record of the data set. A set must index every record in its associated data set, whereas a subset can index zero, one, several, or all data set records. A subset might or might not be automatically maintained by Data Management System II (DMSII).

### **syncpoint**

A point in time when no program is in a transaction state.

### **syntax**

The rules or grammar of a language.

## **T**

### **TFL**

*See* Transaction Formatting Language.

**timestamp**

An encoded, 48-bit numerical value for the time and date. Various timestamps are maintained by the system for each disk file. Timestamps note the time and date a file was created, last altered, and last accessed.

**TITLE**

A file attribute whose value is the external name of a file. By default, this value is the value of the INTNAME attribute. For a file whose KIND attribute is equal to DISK or PACK, the TITLE attribute can be assigned a value of the form *<file name> ON <family name>*; thus, the values of the TITLE and FILENAME attributes, both of which specify the external file name, can be different.

**TPS**

See transaction processing system.

**transaction**

(1) The transfer of one message from a terminal or host program to a receiving host program, the processing carried out by the receiving host program, and the return of an answer to the sender. (2) In data management, a sequence of operations grouped by a user program because the operations constitute a single logical change to the database.

**Transaction Formatting Language (TFL)**

The Unisys language used to write source files that are compiled to produce description files for transaction bases.

**transaction point**

A point that is explicitly assigned in a program between a begin transaction statement and an end transaction statement so that the programmer is able to cancel or partially cancel a transaction that has not yet completed processing.

**transaction processing system (TPS)**

A Unisys system that provides methods for processing a high volume of transactions, keeps track of all input transactions that access the database, enables the user to batch data for later processing, and enables transactions to be processed on a database that resides on a remote system.

**transaction state**

The period in a user-language program between a begin transaction operation and an end transaction operation.

**U**

**UCF**

See User Communication Form.

**unordered**

Referring to files, data sets, sets, and subsets that are not maintained in a user-specified order.

**update**

To delete, insert, or modify information in a database or transaction base.

## Glossary

---

### **update mode**

A database or file access mode in which data can be inserted, deleted, or modified.

### **User Communication Form (UCF)**

A form used by Unisys customers to report problems and express comments about Unisys products to support organizations.

### **user work area**

A memory area in a user program where data records are constructed, accessed, or modified. The Accessroutines maintain one user work area for each data set or remap invoked by a program.

### **USING clause**

In COBOL, an optional clause in the *CALL nonnumeric literal* statement that identifies the parameters being passed to a library entry point.

### **utility**

A system software program that performs commonly used functions.

## V

### **variable**

An object in a program whose value can be changed during program execution.

### **variable format**

A record format that consists of two parts: a fixed part and a variable-format part. A single record description exists for the fixed part. The variable-format part can describe several variable parts. An individual record is constructed by using the fixed part alone, or by joining the fixed part with one of the variable parts.

### **variable-format data set**

A data set with records that can have several different formats. All records in such a data set, known as variable-format records, consist of a common fixed part and optionally include one of the variable parts declared in the Data and Structure Definition Language (DASDL).

## W

### **WFL**

See Work Flow Language.

### **WFL job**

(1) A Work Flow Language (WFL) program, or the execution of such a program. (2) A collection of WFL statements that enable the user to run programs or tasks.

### **WFL job deck**

See job.

**word**

A unit of computer memory. On A Series systems, a word consists of 48 bits used for storage plus tag bits used to indicate how the word is interpreted.

**Work Flow Language (WFL)**

A Unisys language used for constructing jobs that compile or run programs on A Series systems. WFL includes variables, expressions, and flow-of-control statements that offer the programmer a wide range of capabilities with regard to task control.

**Z**

**ZIP**

A statement that initiates the Work Flow Language (WFL) compiler. A ZIP is commonly used to initiate compilations automatically.



# Bibliography

- A Series ALGOL Programming Reference Manual, Volume 1: Basic Implementation* (form 8600 0098). Unisys Corporation.
- A Series ALGOL Programming Reference Manual, Volume 2: Product Interfaces* (form 8600 0734). Unisys Corporation.
- A Series ALGOL Test and Debug System (TADS) Programming Guide* (form 1169539). Unisys Corporation.
- A Series C Programming Reference Manual* (form 3950 8775). Unisys Corporation.
- A Series CANDE Configuration Reference Manual* (form 8600 1344). Unisys Corporation.
- A Series CANDE Operations Reference Manual* (form 8600 1500). Unisys Corporation.
- A Series COBOL ANSI-68 Programming Reference Manual* (form 8600 0320). Unisys Corporation.
- A Series COBOL ANSI-74 Programming Reference Manual, Volume 1: Basic Implementation* (form 8600 0296). Unisys Corporation.
- A Series COBOL ANSI-74 Programming Reference Manual, Volume 2: Product Interfaces* (form 8600 0130). Unisys Corporation.
- A Series COBOL ANSI-74 Test and Debug System (TADS) Programming Guide* (form 1169901). Unisys Corporation.
- A Series COBOL ANSI-85 Programming Reference Manual, Volume 1: Basic Implementation* (form 8600 1518). Unisys Corporation.
- A Series COBOL ANSI-85 Programming Reference Manual, Volume 2: Product Interfaces* (form 8600 1526). Unisys Corporation.
- A Series Data Management Functional Overview* (form 8600 0239). Unisys Corporation.
- A Series DMSII Application Program Interfaces Programming Guide* (form 5044225). Unisys Corporation. Formerly *A Series DMSII User Language Interface Programming Guide*.
- A Series DMSII Data and Structure Definition Language (DASDL) Programming Reference Manual* (form 8600 0213). Unisys Corporation.
- A Series DMSII Documents Index* (form 8600 1443). Unisys Corporation.

## Bibliography

---

- A Series DMSII Technical Overview* (form 5044191). Unisys Corporation.
- A Series DMSII Transaction Processing System (TPS) Programming Guide* (form 1164043). Unisys Corporation.
- A Series DMSII Utilities Operations Guide* (form 8600 0759). Unisys Corporation.
- A Series Documentation Library Overview* (form 8600 0361). Unisys Corporation.
- A Series Editor Operations Guide* (form 8600 0551). Unisys Corporation.
- A Series Extended Retrieval with Graphic Output (ERGO) Operations Guide* (form 8600 0205). Unisys Corporation.
- A Series FORTRAN77 Programming Reference Manual* (form 3950 8759). Unisys Corporation.
- A Series FORTRAN77 Test and Debug System (TADS) Programming Guide* (form 1222667). Unisys Corporation.
- A Series Mark 3.9 Software Release Capabilities Overview* (form 8600 0015). Unisys Corporation.
- A Series Pascal Programming Reference Manual, Volume 1: Basic Implementation* (form 8600 0080). Unisys Corporation.
- A Series Pascal Programming Reference Manual, Volume 2: Product Interfaces* (form 8600 1294). Unisys Corporation.
- A Series PL/1 Reference Manual* (form 1169620). Unisys Corporation.
- A Series Railroad Diagram Reference Card* (form 5044266). Unisys Corporation.
- A Series Report Program Generator (RPG) Programming Reference Manual, Volume 1: Basic Implementation* (form 8600 0544). Unisys Corporation.
- A Series Report Program Generator (RPG) Programming Reference Manual, Volume 2: Product Interfaces* (form 8600 0742). Unisys Corporation.
- A Series Report Program Generator (RPG) Programming Template* (form 1195302). Unisys Corporation.
- A Series System Messages Support Reference Manual* (form 8600 0429). Unisys Corporation.
- A Series Systems Functional Overview* (form 8600 0353). Unisys Corporation.
- A Series Task Attributes Programming Reference Manual* (form 8600 0502). Unisys Corporation. Formerly *A Series Work Flow Administration and Programming Guide*.

*A Series Task Management Programming Guide* (form 8600 0494). Unisys Corporation.  
Formerly *A Series Work Flow Administration and Programming Guide*.

*A Series Work Flow Language (WFL) Programming Reference Manual*  
(form 8600 1047). Unisys Corporation.

*The DMSII Primer, Volume 1*. Sonoma, CA: Gregory Publishing Company, 1982.





# Index

## A

- ABORT category
  - exception and error subcategories, B-3
  - major category number, B-2
- abort transaction operations, 3-48
- aborting transactions, 3-48, 3-53
- access methods, defining in DBSTRUCTURE data set, 7-5
- access modes, for open operations, 3-6
- ACCESS value, as entry for SET-SUBTYPE, 7-5
- accessing a DMINTERPRETER library
  - in ALGOL programs, 2-2
  - in COBOL programs, 2-8, 2-9
  - in FORTRAN77 programs, 2-13
- accessing databases, using the DMINTERPRETER library, 1-4
- accessing entry points, 1-5
- accessing library entry points
  - in ALGOL programs, 2-3
  - in COBOL programs, 2-11
  - in FORTRAN77 programs, 2-14
- accessing the interpretive interface, 2-1
  - in ALGOL programs, 2-2
  - in COBOL programs, 2-8
  - in FORTRAN77 programs, 2-12
- accessing user work areas, 4-1
- Accessroutines
  - as run-time interface, 1-1
  - exceptions and errors returned, B-1
  - restricting number of calls to, 6-1
- ACTUALNAME clause, renaming library entry points
  - in ALGOL programs, 2-6
  - in FORTRAN77 programs, 2-17
- adding new records, 3-9
- ALGOL
  - aborting transactions, 3-49
  - accessing entry points, 1-5, 2-2
  - accessing library entry points, 2-3
  - accessing the interpretive interface, 2-2
  - ACTUALNAME clause, using, 2-6
  - beginning transaction state, 3-46
  - canceling transactions to savepoints, 3-54
  - changing libraries, 2-2
  - closing databases, 3-72
  - constructing data entry points dynamically, 4-62
  - creating new records, 3-11
  - declaring entry points, 2-3
  - declaring the DMINTERPRETER library, 2-2
  - deleting records, 3-59
  - ending transaction state, 3-70
  - entry points, table of, 2-4
  - exception handling, 2-7
  - executing language extensions, 3-77
  - exported names of entry points, 2-4
  - finding records, 3-27
  - identifying types of exceptions, 5-9
  - invoking entry points, 2-7
  - library declaration, 2-2
  - locking records, 3-32
  - locking structures, 3-35
  - moving character strings to variables, 4-8
  - moving double-precision values to variables, 4-23
  - moving Kanji character strings to variables, 4-13
  - moving numeric values to variables, 4-19
  - opening databases, 3-7
  - placing double-precision values into data items, 4-49
  - placing numeric values into data items, 4-44
  - placing strings into data items, 4-34
  - placing strings into Kanji alpha items, 4-39
  - programming considerations
    - accessing entry points, 1-5, 2-2
    - as preferred application programming language, 1-8

## Index

---

- entry point names, 1-6
- re-creating records, 3-64
- renaming entry points, 2-4
- restricting calls to the Accessroutines, 6-2, 6-3
- retrieving Boolean or field items, 4-28
- returning exceptions, 5-4
- returning text of exception messages, 5-6
- sample program, C-5
- saving transaction points, 3-51
- securing records, 3-40
- securing structures, 3-43
- setting Boolean values, 4-53
- setting current path, 3-15
- setting null values, 4-58
- storing records, 3-67
- table of entry points, 2-4
- TITLE attribute, using, 2-2
- unlocking records, 3-19
- unlocking structures, 3-22
- \$INCLUDE compiler control option, use of, 2-4
- ALGOLABORTTRANSACTION, aborting transactions, 3-48
- ALGOLBEGINTRANSACTION, beginning transaction state, 3-45
- ALGOLCANCELTRPOINT, canceling transactions to savepoints, 3-53
- ALGOLCLOSE, closing databases, 3-72
- ALGOLCREATE, creating new records, 3-9
- ALGOLDATA, constructing data entry points dynamically, 4-60
- ALGOLDELETE, deleting a record, 3-56
- ALGOLENDTRANSACTION, ending transaction state, 3-69
- ALGOLEXCEPTIONNAME, identifying types of exceptions, 5-8
- ALGOLEXCEPTIONTEXT, returning text of exception message, 5-5
- ALGOLFIND, finding records, 3-24
- ALGOLFREE, freeing records, 3-17
- ALGOLFREESTR, freeing structures, 3-21
- ALGOLGETBOOLEAN, retrieving Boolean or field items, 4-26
- ALGOLGETDOUBLE, moving numeric values to variables
  - double precision, 4-21
- ALGOLGETKANJI, retrieving Kanji character strings, 4-11
- ALGOLGETREAL, moving numeric values to variables, 4-16
- ALGOLGETSTRING, moving character strings to variables, 4-6
- ALGOLLOCK, locking records, 3-29
- ALGOLLOCKSTR, locking structures, 3-34
- ALGOLOPEN, opening a database, 3-6
- ALGOLPUTBOOLEAN, setting data items to Boolean values, 4-51
- ALGOLPUTDOUBLE
  - placing double-precision values into data items, 4-46
- ALGOLPUTKANJI, placing strings into Kanji alpha items, 4-36
- ALGOLPUTNULL, setting null values, 4-56
- ALGOLPUTREAL, placing numeric values into data items, 4-41
- ALGOLPUTSTRING, placing strings into data items, 4-31
- ALGOLRECREATE, re-creating records, 3-62
- ALGOLSAVETRPOINT, saving transaction points, 3-50
- ALGOLSECURE, securing records, 3-37
- ALGOLSECURESTR, securing structures, 3-42
- ALGOLSET, setting the current path, 3-13
- ALGOLSETLIMIT, restricting calls on the Accessroutines, 6-1
- ALGOLSTATUS, returning exceptions, 5-3
- ALGOLSTORE, storing records, 3-66
- ALGOLVERB, executing language extensions, 3-74
- ALPHA entry, for ITEM-SUBTYPE, 7-6
  - alpha items
    - moving character strings to variables, 4-6
    - moving Kanji character strings to variables, 4-11
    - placing strings into data items, 4-31
    - placing strings into Kanji alpha items, 4-36
- application program interfaces, types of, 1-1
- < arithmetic expression >
  - placing double-precision values into data items, 4-48
  - placing numeric values into data items, 4-43
- attribute setting entry points, 6-1
  - purpose of, 1-4
  - table of, 6-1
- AUDIT value
  - setting when beginning transaction state, 3-46
  - setting when ending transaction state, 3-69

- < audit >
    - beginning transaction state, 3-46
    - ending transaction state, 3-69
  - audited databases
    - aborting transactions, 3-48
    - beginning transaction state, 3-45
    - canceling transactions back to savepoints, 3-53
    - deleting records, 3-56
    - ending transaction state, 3-69
    - saving transaction points, 3-50
    - storing records, 3-66
  - AUDITERERROR category
    - exception and error subcategories, B-3
    - major category number, B-2
  - < auditsync >, ending transaction state, 3-69
- B**
- basics of the interpretive interface, 1-1
  - begin transaction operations, 3-45
    - guidelines, 1-9
  - begin transaction state, 3-45
  - BEGINNING value, in setting current path, 3-14
  - BINARY value, as entry for SET-SEARCH-METHOD, 7-5
  - blanks
    - creating new records, 3-10
    - re-creating records, 3-62
  - blanks, use in interpretive interface
    - application programs, 1-8
  - BOOLEAN entry, for ITEM-SUBTYPE, 7-6
  - < Boolean expression >, setting Boolean values in ALGOL, 4-53
  - Boolean procedures, declaring ALGOL entry points, 2-3
  - Boolean values
    - retrieving, 4-26
    - setting, 4-51
  - BUILDINQ program
    - capabilities, A-1
    - correcting inconsistencies, A-4
    - files used and produced by, A-1
    - generating a DMINTERPRETER library, A-5
    - generating the DBSTRUCTURE data set, 7-1
    - relationship to a DMINTERPRETER library, 1-4
    - using NOZIP value, A-13
    - using ZIP option, A-5
- C**
- CALL statement, COBOL nonnumeric literal statement, 2-11
  - calls, restricting number to the
    - Accessroutines, 6-1
  - cancel transaction point operations, 3-53
  - canceling transactions back to savepoints, 3-53
  - categories of exceptions and errors, B-2
  - CHANGE construct, invoking
    - DMINTERPRETER library in COBOL, 2-10
  - close database operations, 3-72
  - CLOSEERROR category
    - exception and error subcategories, B-4
    - major category number, B-2
  - closing databases, 3-72
  - COBOL
    - aborting transactions, 3-49
    - accessing entry points, 1-6, 2-8
    - accessing library entry points, 2-11
    - accessing the interpretive interface, 2-8
    - beginning transaction state, 3-47
    - CALL statement, 2-11
    - canceling transactions to savepoints, 3-55
    - CHANGE construct, 2-10
    - changing libraries, 2-10
    - closing databases, 3-73
    - constructing data entry points dynamically, 4-63
    - creating new records, 3-11
    - deleting records, 3-60
    - ending transaction state, 3-71
    - entry points, table of, 2-8
    - executing language extensions, 3-78
    - exported names of entry points, 2-8
    - finding records, 3-28
    - GIVING clause, 2-12
    - identifying types of exceptions, 5-10
    - invoking entry points, 2-11
    - invoking the DMINTERPRETER library, 2-9
    - locking records, 3-33
    - locking structures, 3-36
    - moving character strings to variables, 4-9
    - moving double-precision values to variables, 4-21
    - moving numeric values to variables, 4-16

## Index

---

- opening databases, 3-8
  - passing parameters, 2-11
  - placing double-precision values into data items, 4-46, 4-49
  - placing numeric values into data items, 4-41, 4-44
  - placing strings into data items, 4-34
  - placing strings into Kanji alpha items, 4-39
  - programming considerations
    - accessing entry points, 1-6, 2-8
    - entry point names, 1-6
  - re-creating records, 3-64
  - restricting calls to the Accessroutines, 6-2, 6-3
  - retrieving Boolean or field items, 4-29
  - returning exceptions, 5-4
  - returning text of exception messages, 5-7
  - saving transaction points, 3-52
  - securing records, 3-41
  - securing structures, 3-44
  - setting Boolean values, 4-54
  - setting current path, 3-15
  - setting null values, 4-59
  - storing records, 3-68
  - table of entry points, 2-8
  - unlocking records, 3-19
  - unlocking structures, 3-23
  - using TITLE attribute, 2-10
- COBOL74**
- aborting transactions, 3-49
  - accessing entry points, 1-6, 2-8
  - accessing library entry points, 2-11
  - accessing the interpretive interface, 2-8
  - beginning transaction state, 3-47
  - CALL statement, 2-11
  - canceling transactions to savepoints, 3-55
  - CHANGE construct, 2-10
  - changing libraries, 2-10
  - closing databases, 3-73
  - constructing data entry points dynamically, 4-63
  - creating new records, 3-11
  - deleting records /inx>, 3-60
  - ending transaction state, 3-71
  - entry points, table of, 2-8
  - executing language extensions, 3-78
  - exported names of entry points, 2-8
  - finding records /inx>, 3-28
  - GIVING clause, 2-12
  - identifying types of exceptions, 5-10
  - invoking entry points, 2-11
  - invoking the DMINTERPRETER library, 2-9
  - locking records /inx>, 3-33
  - locking structures /inx>, 3-36
  - moving character strings to variables, 4-9
  - moving double-precision values to variables, 4-21
  - moving Kanji character strings to variables, 4-14
  - moving numeric values to variables, 4-16
  - opening databases, 3-8
  - passing parameters, 2-11
  - placing double-precision values into data items, 4-46, 4-49
  - placing numeric values into data items, 4-41, 4-44
  - placing strings into data items, 4-34
  - placing strings into Kanji alpha items, 4-39
  - programming considerations
    - accessing entry points, 1-6, 2-8
    - entry point names, 1-6
  - re-creating records, 3-64
  - restricting calls to the Accessroutines, 6-2, 6-3
  - retrieving Boolean or field items, 4-29
  - returning exceptions, 5-4
  - returning text of exception messages, 5-7
  - sample program, C-15
  - saving transaction points, 3-52
  - securing records /inx>, 3-41
  - securing structures /inx>, 3-44
  - setting Boolean values, 4-54
  - setting current path /inx>, 3-15
  - setting null values, 4-59
  - storing records, 3-68
  - table of entry points, 2-8
  - unlocking records, 3-19
  - unlocking structures, 3-23
  - using TITLE attribute, 2-10
- COBOL85**
- aborting transactions, 3-49
  - accessing entry points, 1-6, 2-8
  - accessing library entry points, 2-11
  - accessing the interpretive interface, 2-8
  - beginning transaction state, 3-47
  - CALL statement, 2-11
  - canceling transactions to savepoints, 3-55
  - CHANGE construct, 2-10
  - changing libraries, 2-10
  - closing databases, 3-73
  - constructing data entry points dynamically, 4-63

- creating new records, 3-11
  - deleting records, 3-60
  - ending transaction state, 3-71
  - entry points, table of, 2-8
  - executing language extensions, 3-78
  - exported names of entry points, 2-8
  - finding records, 3-28
  - GIVING clause, 2-12
  - identifying types of exceptions, 5-10
  - invoking entry points, 2-11
  - invoking the DMINTERPRETER library, 2-9
  - locking records, 3-33
  - locking structures, 3-36
  - moving character strings to variables, 4-9
  - moving double-precision values to variables, 4-21
  - moving Kanji character strings to variables, 4-14
  - moving numeric values to variables, 4-16
  - opening databases, 3-8
  - passing parameters, 2-11
  - placing double-precision values into data items, 4-46, 4-49
  - placing numeric values into data items, 4-41, 4-44
  - placing strings into data items, 4-34
  - placing strings into Kanji alpha items, 4-39
  - programming considerations
    - accessing entry points, 1-6, 2-8
    - entry point names, 1-6
  - re-creating records, 3-64
  - restricting calls to the Accessroutines, 6-2, 6-3
  - retrieving Boolean or field items, 4-29
  - returning exceptions, 5-4
  - returning text of exception messages, 5-7
  - saving transaction points, 3-52
  - securing records, 3-41
  - securing structures, 3-44
  - setting Boolean values, 4-54
  - setting current path, 3-15
  - setting null values, 4-59
  - storing records, 3-68
  - table of entry points, 2-8
  - unlocking records, 3-19
  - unlocking structures, 3-23
  - using TITLE attribute, 2-10
  - coding of standard entry points, sequence, 3-1
  - COMPACT data set, as entry for DATASET-SUBTYPE, 7-4
  - compile-time interface, 1-1
  - compilers that can use the interpretive interface, 1-1
  - < condition >
    - deleting records, 3-58
    - finding records, 3-26
    - locking records, 3-31
    - securing records, 3-39
  - constructing data transfers during program execution, 4-60
  - copy operations, 3-62
  - copying records, through re-creation operation, 3-61
  - COUNT entry, for ITEM-SUBTYPE, 7-6
  - count items
    - transferring double-precision values to variables, 4-21
    - transferring values to variables, 4-16
  - create operations, 3-9
  - < create statement >, executing through entry point, 3-75
  - creating new records, 3-9
  - current paths, setting, 3-13
- D**
- DASDL, (See Data Structure and Definition Language (DASDL))
  - data items, tables of, 7-2
  - data management statements, executing, 3-74
  - < data request >, constructing transfers dynamically, 4-60
  - < data set qualified name >
    - creating new records, 3-10
    - deleting records, 3-58
    - finding records, 3-26
    - freeing data set records, 3-18
    - freeing structures, 3-21
    - locking records, 3-31
    - locking structures, 3-35
    - moving character strings to variables, 4-7
    - moving double-precision values to variables, 4-22
    - moving Kanji character strings to variables, 4-12
    - moving numeric values to variables, 4-17
    - placing double-precision values into data items, 4-47
    - placing numeric values into data items, 4-42

- placing string into Kanji alpha items, 4-37
- placing strings into data items, 4-32
- re-creating records, 3-62
- retrieving Boolean or field items, 4-27
- securing records, 3-39
- securing structures, 3-43
- setting Boolean values, 4-52
- setting current path, 3-14
- setting null values, 4-57
- storing records, 3-66
- data set records, freeing, 3-17
- < data set >
  - locking structures, 3-34
  - securing structures, 3-42
- data sets
  - defining types of in DBSTRUCTURE data set, 7-4
  - setting current paths, 3-13
- Data Structure and Definition Language (DASDL)
  - database for sample programs, C-2
  - describing the DBSTRUCTURE data set, 7-2
  - using ZIP, A-5
- data transfer entry points, 4-1
  - ALGOLDATA, 4-60
  - ALGOLGETBOOLEAN, 4-26
  - ALGOLGETDOUBLE, 4-21
  - ALGOLGETKANJI, 4-11
  - ALGOLGETREAL, 4-16
  - ALGOLGETSTRING, 4-6
  - ALGOLPUTBOOLEAN, 4-51
  - ALGOLPUTDOUBLE, 4-46
  - ALGOLPUTKANJI, 4-36
  - ALGOLPUTNULL, 4-56
  - ALGOLPUTREAL, 4-41
  - ALGOLPUTSTRING, 4-31
  - DBDATA, 4-60
  - DBGETBOOLEAN, 4-26
  - DBGETDISPLAY, 4-6
  - DBGETDOUBLE, 4-21
  - DBGETKANJI, 4-11
  - DBGETREAL, 4-16
  - DBPUTBOOLEAN, 4-51
  - DBPUTDISPLAY, 4-31
  - DBPUTDOUBLE, 4-46
  - DBPUTKANJI, 4-36
  - DBPUTNULL/inx>, 4-56
  - DBPUTREAL, 4-41
  - exceptions, 4-4
  - FORTTRAN77DATA, 4-60
  - FORTTRAN77GETBOOLEAN, 4-26
  - FORTTRAN77GETCHARACTER, 4-6
  - FORTTRAN77GETDOUBLE, 4-21
  - FORTTRAN77GETKANJI, 4-11
  - FORTTRAN77GETREAL, 4-16
  - FORTTRAN77PUTCHARACTER, 4-31
  - FORTTRAN77PUTDOUBLE, 4-46
  - FORTTRAN77PUTKANJI, 4-36
  - FORTTRAN77PUTLOGICAL, 4-51
  - FORTTRAN77PUTNULL, 4-56
  - FORTTRAN77PUTREAL, 4-41
  - purpose of, 1-4
  - table of, 4-1
- data transfer tasks
  - performed by entry points to the DMINTERPRETER library, 4-4
- DATA value, as entry for ITEM-KEY-CLASS, 7-8
- < data-name >, invoking COBOL entry points, 2-11
- < data >, constructing transfers dynamically, 4-61
- DATABASE card, A-15
- database definition for
  - sample programs, C-2
- database equation, using, 1-9
- < database name >
  - finding records, 3-26
  - freeing global data records, 3-18
  - locking records, 3-31
  - securing records, 3-39
  - storing records, 3-67
- < database statement >, executing through
  - entry point, 3-74
- database structure, determining, 7-1
- DATABASE/DMINTERPRETER
  - ensuring consistency with program symbolic, A-4
  - generating DMINTERPRETER library, A-1
  - use in FORTRAN77, 2-14
- DATABASE/DMINTERPRETER symbolic file
  - providing declarations of library entry points, 1-5
  - providing exported names of library entry points, 1-6
  - use in ALGOL programs, 2-4
  - use in COBOL programs, 2-8
  - use in FORTRAN77 programs, 2-14
- DATABASE/PROPERTIES, generating
  - DMINTERPRETER library, A-1
- databases

- aborting transactions, 3-48
- beginning transaction state, 3-45
- canceling transactions back to savepoints, 3-53
- closing, 3-72
- declaring, 2-1
- ending transaction state, 3-69
- invoking, 2-1
- opening, 2-1, 3-6
- saving transaction points, 3-50
- DATAERROR category
  - exception and error subcategories, B-4
  - major category number, B-2
- DATASET value, as entry for DB-TYPE, 7-3
- DATASET-SUBTYPE item, in
  - DBSTRUCTURE data set, 7-4
- DB-ID item, in DBSTRUCTURE data set, 7-3
- DB-NAME item, in DBSTRUCTURE data set, 7-3
- DB-OWNER item, in DBSTRUCTURE data set, 7-3
- DB-TYPE item, in DBSTRUCTURE data set, 7-3
- DB-TYPE of DATASET, DBSTRUCTURE items, 7-4
- DB-TYPE of ITEM, DBSTRUCTURE data set items, 7-5
- DB-TYPE of LINK, DBSTRUCTURE items, 7-8
- DB-TYPE of SET, DBSTRUCTURE items, 7-4
- DBABORTTRANSACTION, aborting transactions, 3-48
- DBBEGINTRANSACTION, beginning transaction state, 3-45
- DBCANCELTRPOINT, canceling transactions to savepoints, 3-53
- DBCLOSE, closing databases, 3-72
- DBCREATE, creating new records, 3-9
- DBDATA, constructing data entry points dynamically, 4-60
- DBDELETE, deleting a record, 3-56
- DBENDTRANSACTION, ending transaction state, 3-69
- DBEXCEPTIONNAME, identifying types exceptions, 5-8
- DBEXCEPTIONTEXT, returning text of exception message, 5-5
- DBFIND, finding records, 3-24
- DBFREE, freeing records, 3-17
- DBFREESTR, freeing structures, 3-21
- DBGET entry points, 4-3
- DBGET operations
  - constructing data entry points dynamically, 4-60
  - moving character strings to variables, 4-6
  - moving double-precision values to variables, 4-21
  - moving Kanji strings to variables, 4-11
  - moving numeric values to variables, 4-16
  - retrieving Boolean or field items, 4-26
  - using with DBSTRUCTURE data set, 7-1
- DBGETBOOLEAN, retrieving Boolean or field items, 4-26
- DBGETDISPLAY, moving character strings to variables, 4-6
- DBGETDOUBLE, moving double-precision values to variables, 4-21
- DBGETKANJI, retrieving Kanji character strings, 4-11
- DBGETREAL, moving numeric values to variables, 4-16
- DBLOCK, locking records, 3-29
- DBLOCKSTR, locking structures, 3-34
- DBOPEN, opening a database, 3-6
- DBPUT entry points, 4-3
- DBPUT operations
  - constructing data entry points dynamically, 4-60
  - placing double-precision values into data items, 4-46
  - placing numeric values into data items, 4-41
  - placing strings into data items, 4-31
  - placing strings into Kanji alpha items, 4-36
  - setting data items to Boolean values, 4-51
  - setting data items to null values, 4-56
- DBPUTBOOLEAN, setting data items to Boolean values, 4-51
- DBPUTDISPLAY, placing strings into data items, 4-31
- DBPUTDOUBLE, placing double-precision values into data items, 4-46
- DBPUTKANJI, placing strings into Kanji alpha items, 4-36
- DBPUTNULL, setting null values, 4-56
- DBPUTREAL, placing numeric values into data items, 4-41
- DBRECREATE, re-creating records, 3-62
- DBSAVETRPOINT, saving transaction points, 3-50
- DBSECURE, securing records, 3-37
- DBSECURESTR, securing structures, 3-42



## Index

---

- DBSET, setting the current path, 3-13
- DBSETLIMIT, restricting calls on the
  - Accessroutines, 6-1
- DBSTATUS, returning exceptions, 5-3
- DBSTORE, storing records, 3-66
- DBSTRUCTURE data set
  - DASDL description, 7-2
  - defining access method, 7-5
  - generating, 7-1
  - generating via BUILDINQ program, 7-1
  - items, 7-4
  - items for DB-TYPE of DATASET, 7-4
  - items for DB-TYPE of ITEM, 7-5
  - items for DB-TYPE of LINK, 7-8
  - items for DB-TYPE of SET, 7-4
  - provided by DMINTERPRETER library, 7-1
  - types of data sets, 7-4
  - types of entities, 7-3
  - types of sets, 7-5
  - using with ERGO, 7-1
- DBSTRUCTURE data sets
  - specifying use through BUILDINQ program, A-10
  - use with ERGO, A-10
- DBVERB, executing language extensions, 3-74
- DEADLOCK category
  - exception and error subcategories, B-5
  - major category number, B-2
- deadlocks
  - exceptions and errors, B-5
- deadly embrace, (See deadlocks)
- declaring a DMINTERPRETER library
  - in ALGOL programs, 2-2
  - in FORTRAN77 programs, 2-13
- declaring a subset of the entry points
  - in ALGOL programs, 2-5
  - in FORTRAN77 programs, 2-16
- declaring databases
  - purpose of, 2-1
- declaring entry points, 1-5
  - in ALGOL programs, 2-3
    - through procedure declaration, 2-5
    - through release tape, 2-4
  - in FORTRAN77 programs
    - through logical functions, 2-16
    - through release tape, 2-14
  - purpose of, 2-1
- declaring library entry points
  - in FORTRAN77 programs, 2-14
- delete operations, 3-56
- deleting data records, 3-56
- deleting records
  - before recreating a record, 3-61
- DESCRIPTION/< database name >
  - in DMINTERPRETER library, A-1
- determining access mode, in BUILDINQ program, A-11
- DIRECT data set, as entry for DATASET-SUBTYPE, 7-4
- DIRECT value
  - as entry for SET-SEARCH-METHOD, 7-5
- < direction >
  - deleting records, 3-57
  - finding a record, 3-25
  - locking records, 3-30
  - securing records, 3-38
- DMIDIRECTORY/< database name >
  - ensuring consistency with program symbolic, A-4
  - generating DMINTERPRETER library, A-1
- DMINTERPRETER card, A-15
- DMINTERPRETER library
  - attribute setting entry points, 6-1
  - BUILDINQ program, relationship to, A-1
  - data transfer entry points, 4-3
  - database equation, using, 1-9
  - DBGGET entry points, 4-3
  - DBPUT entry points, 4-3
  - declaring, 2-1
  - exception handling entry points, 5-1
  - files produced when generating, A-1
  - files used to generate, A-1
  - generating, 1-4, A-1
    - from remote terminal, A-5
    - from WFL job deck, A-15
    - using NOZIP value, A-13
    - using ZIP, A-5
  - invoking, 2-1
  - providing DBSTRUCTURE data set, 7-1
  - purpose of, 1-4
  - standard entry points, 3-1
  - structure switching, 1-8
  - tasks performed by entry points
    - attribute setting, 6-1
    - data transfer, 4-4
    - exception handling, 5-1
  - types of entry points, 1-4
  - use of global items, 1-9
- DMINTERPRETER library entry points
  - declaring, 2-1
  - guidelines for sequencing entry points, 1-8

- invoking, 2-1
  - naming and renaming, 1-6
  - returning exceptions, 1-7
  - types of, 1-4
  - using parameters, 1-7
  - DMINTERPRETER/< database name >,
    - generating DMINTERPRETER, A-1
  - DMSII application program interfaces, 1-1
  - DMSII libraries, using, 1-4
  - DMSUPPORT library
    - purpose of, 1-4
    - use by exception handling entry points, 1-7
    - use in exceptions, 5-1
  - DUPLICATES category
    - exception and error subcategories, B-5
    - major category number, B-2
  - duplicating records, 3-61
- E**
- < EBCDIC string >
    - declaring DMINTERPRETER library in ALGOL, 2-3
    - declaring entry points, 2-6
  - EBCDIC strings, using in DBSTRUCTURE data set, 7-8
  - empty string
    - creating new records, 3-10
    - re-creating records, 3-62
  - end transaction operations, 3-69
    - programming consideration, 1-9
  - ending transaction state, 3-69
  - ENDING value, in setting current path, 3-14
  - entering compilation queue, in BUILDINQ program, A-12
  - entering DMINTERPRETER code file name, in BUILDINQ program, A-12
  - entering the name of a logical database, in BUILDINQ program, A-11
  - entering transaction state, 3-45
  - entities, defining types of in DBSTRUCTURE data set, 7-3
  - entry point declarations
    - in ALGOL programs, 2-3
    - in COBOL programs, 2-11
    - in FORTRAN77 programs, 2-14
  - entry points
    - ALGOL, listing of, 2-4
    - attribute setting, 6-1
    - COBOL, listing of, 2-8
    - coding sequence, 3-1
    - constructing transfers during program execution, 4-60
    - data transfer, 4-1, 4-4
    - DBGET, 4-3
    - DBPUT, 4-3
    - declaring, 2-1
    - declaring in ALGOL programs, 2-3, 2-7
    - declaring in FORTRAN77 programs, 2-14, 2-18
    - exception handling, 5-1
    - exporting, 1-4
    - FORTRAN77, listing of, 2-14
    - guidelines for sequencing entry points, 1-8
    - identifying types of exceptions, 5-8
    - importing, 1-4
    - invoking, 2-1
    - invoking in ALGOL programs, 2-7
    - invoking in COBOL programs, 2-11
    - invoking in FORTRAN77 programs, 2-18
    - moving character strings to variables, 4-6
    - moving data in and out of a user work area, 4-3
    - moving double-precision values to variables, 4-21
    - moving numeric values to variables, 4-16
    - placing double-precision values into data items, 4-46
    - placing numeric values into data items, 4-41
    - placing strings into data items, 4-31
    - placing strings into Kanji alpha items, 4-36
    - restricting calls to the Accessroutines, 6-2
    - retrieving Boolean values, 4-26
    - retrieving Kanji alpha character strings, 4-11
    - returning exceptions, 1-7, 5-3
    - returning text of exception messages, 5-5
    - setting data items to Boolean values, 4-51
    - setting data items to null values, 4-56
    - standard, 3-1
    - types of, 1-4
    - using parameters, 1-7
    - using with DBSTRUCTURE data set, 7-1
  - ERGO, (See Extended Retrieval with Graphic Output)
  - error handling, in interpretive interface, 5-1
  - errors
    - major categories by number, B-2
    - subcategories, B-2
  - examples
    - aborting transactions

## Index

---

- ALGOL, 3-49
- COBOL, 3-49
- FORTRAN77, 3-49
- beginning transaction state
  - ALGOL, 3-46
  - COBOL, 3-47
  - FORTRAN77, 3-47
- canceling transactions to savepoints
  - ALGOL, 3-54
  - COBOL, 3-55
  - FORTRAN77, 3-55
- closing databases
  - ALGOL, 3-72
  - COBOL, 3-73
  - FORTRAN77, 3-73
- constructing data entry points dynamically
  - ALGOL, 4-62
  - COBOL, 4-63
  - FORTRAN77, 4-63
- creating new records
  - ALGOL, 3-11
  - COBOL, 3-11
  - FORTRAN77, 3-12
- declaring library entry point
  - ALGOL, 2-7
  - FORTRAN77, 2-18
- deleting records
  - ALGOL, 3-59
  - COBOL, 3-60
  - FORTRAN77, 3-60
- ending transaction state
  - ALGOL, 3-70
  - COBOL, 3-71
  - FORTRAN77, 3-71
- executing language extensions
  - ALGOL, 3-77
  - COBOL, 3-78
  - FORTRAN77, 3-79
- finding records
  - ALGOL, 3-27
  - COBOL, 3-28
  - FORTRAN77, 3-28
- identifying types of exceptions
  - ALGOL, 5-9
  - COBOL, 5-10
  - FORTRAN77, 5-10
- invoking an entry point
  - ALGOL, 2-7
  - COBOL, 2-12
  - FORTRAN77, 2-18
- linking to a DMINTERPRETER library
  - ALGOL, 2-3
  - COBOL, 2-10
  - FORTRAN77, 2-14
- locking records
  - ALGOL, 3-32
  - COBOL, 3-33
  - FORTRAN77, 3-33
- locking structures
  - ALGOL, 3-35
  - COBOL, 3-36
  - FORTRAN77, 3-36
- moving character strings to variables
  - ALGOL, 4-8
  - COBOL, 4-9
  - FORTRAN77, 4-10
- moving double-precision values to variables
  - ALGOL, 4-23
  - COBOL, 4-24
  - FORTRAN77, 4-25
- moving Kanji character strings to variables
  - ALGOL, 4-13
  - COBOL, 4-14
  - FORTRAN77, 4-15
- moving numeric values to variables
  - ALGOL, 4-19
  - COBOL, 4-20
  - FORTRAN77, 4-20
- opening databases
  - ALGOL, 3-7
  - COBOL, 3-8
  - FORTRAN77, 3-8
- placing double-precision into data items
  - ALGOL, 4-49
  - COBOL, 4-49
- placing double-precision values into data items
  - FORTRAN77, 4-50
- placing numeric values into data items
  - ALGOL, 4-44
  - COBOL, 4-44
  - FORTRAN77, 4-45
- placing strings into data items
  - ALGOL, 4-34
  - COBOL, 4-34
  - FORTRAN77, 4-35
- placing strings into Kanji alpha items
  - ALGOL, 4-39
  - COBOL, 4-39
  - FORTRAN77, 4-40
- re-creating records
  - ALGOL, 3-64
  - COBOL, 3-64
  - FORTRAN77, 3-65

- restricting calls to the Accessroutines
  - ALGOL, 6-3
  - COBOL, 6-3
  - FORTRAN77, 6-4
- retrieving Boolean or field items
  - ALGOL, 4-28
  - COBOL, 4-29
  - FORTRAN77, 4-30
- returning exceptions
  - ALGOL, 5-4
  - COBOL, 5-4
  - FORTRAN77, 5-4
- returning text of exception messages
  - ALGOL, 5-6
  - COBOL, 5-7
  - FORTRAN77, 5-7
- saving transaction points
  - ALGOL, 3-51
  - COBOL, 3-52
  - FORTRAN77, 3-52
- securing records
  - ALGOL, 3-40
  - COBOL, 3-41
  - FORTRAN77, 3-41
- securing structures
  - ALGOL, 3-43
  - COBOL, 3-44
  - FORTRAN77, 3-44
- setting Boolean values
  - ALGOL, 4-53
  - COBOL, 4-54
  - FORTRAN77, 4-55
- setting current path
  - ALGOL, 3-15
  - COBOL, 3-15
- setting null values
  - ALGOL, 4-58
  - COBOL, 4-59
  - FORTRAN77, 4-59
- storing records
  - ALGOL, 3-67
  - COBOL, 3-68
  - FORTRAN77, 3-68
- unlocking records
  - ALGOL, 3-19
  - COBOL, 3-19
  - FORTRAN77, 3-20
- unlocking structures
  - ALGOL, 3-22
  - COBOL, 3-23
  - FORTRAN77, 3-23
- using a WFL deck to generate the DMINTERPRETER library, A-16
- exception categories, 5-8
- exception handling
  - category of exception, 5-8
  - in ALGOL programs, 2-7
  - in COBOL programs, 2-12
  - in FORTRAN77 programs, 2-18
  - returning exception result, 5-3
  - text of exception message, 5-5
  - using the interpretive interface, 5-1
- exception handling entry points, 5-1
  - ALGOLEXCEPTIONNAME, 5-8
  - ALGOLEXCEPTIONTEXT, 5-5
  - ALGOLSTATUS, 5-3
  - DBEXCEPTIONNAME, 5-8
  - DBEXCEPTIONTEXT, 5-5
  - DBSTATUS, 5-3
  - FORTRAN77EXCEPTIONNAME, 5-8
  - FORTRAN77EXCEPTIONTEXT, 5-5
  - FORTRAN77STATUS, 5-3
  - purpose of, 1-4
  - table of, 5-1
- exception handling tasks, performed by entry points, 5-1
- exception message, in interpretive interface, 5-5
- exception word, returned through entry point, 5-3
- exceptions
  - major categories by number, B-2
  - subcategories, B-2
- executing language extensions, 3-74
- explicitly freeing
  - records, 3-17
  - structures, 3-21
- exported names
  - in ALGOL programs, 2-4
  - in COBOL programs, 2-8
  - in FORTRAN77 programs, 2-14
  - renaming, 1-6
- exporting entry points, 1-4
  - renaming, 1-6
- < expression >
  - placing double-precision values into data items, 4-48
  - placing numeric values into data items, 4-43
  - placing values into character strings, 4-33
  - placing values into Kanji strings, 4-38
  - setting Boolean values, 4-53
- Extended Retrieval with Graphic Output

## Index

---

using DBSTRUCTURE data set, 7-1, A-10

## F

### FATALERROR category

exception and error subcategories, B-6  
major category number, B-2

### FIELD entry, for ITEM-SUBTYPE, 7-6

### field items

retrieving Boolean values, 4-26  
transferring double-precision values to variables, 4-21  
transferring values to variables, 4-16

### find operations, 3-24

restricting linear search, 6-1  
using DBSTRUCTURE data set, 7-1

### finding records, 3-24

### FINDLIMIT, restricting linear searches, 6-1

### FIRST, as direction for

deleting records, 3-57  
locking records, 3-30  
retrieving records, 3-25  
securing records, 3-38

### fixed-format record

creating new records, 3-10  
recreating records, 3-63

### FORTRAN77

aborting transactions, 3-49  
accessing entry points, 1-5, 2-12  
accessing library entry points, 2-14  
accessing the interpretive interface, 2-12  
ACTUALNAME clause, using, 2-17  
beginning transaction state, 3-47  
canceling transactions to savepoints, 3-55  
changing libraries, 2-13  
closing databases, 3-73  
constructing data entry points dynamically, 4-63  
creating new records, 3-12  
declaring entry points, 2-14  
declaring the DMINTERPRETER library, 2-13  
deleting records, 3-60  
ending transaction state, 3-71  
entry points, table of, 2-14  
exception handling, 2-18  
executing language extensions, 3-79  
exported names of entry points, 2-14  
finding records, 3-28  
identifying types of exceptions, 5-10  
invoking entry points, 2-18

### LIBRARY statement (declaration), 2-13

locking records, 3-33  
locking structures, 3-36  
moving character strings to variables, 4-10  
moving double-precision values to variables, 4-25  
moving Kanji character strings to variables, 4-15  
moving numeric values to variables, 4-20  
opening databases, 3-8  
placing double-precision values into data items, 4-50  
placing numeric values into data items, 4-45  
placing strings into data items, 4-35  
placing strings into Kanji alpha items, 4-40  
programming considerations  
accessing entry points, 1-5, 2-12  
entry point names, 1-6  
re-creating records, 3-65  
renaming exported entry points, 2-14  
restricting calls to the Accessroutines, 6-2, 6-4  
retrieving Boolean or field items, 4-30  
returning exceptions, 5-4  
returning text of exception message, 5-7  
saving transaction points, 3-52  
securing records, 3-41  
securing structures, 3-44  
setting Boolean values, 4-55  
setting current path, 3-16  
setting null values, 4-59  
storing records, 3-68  
table of entry points, 2-14  
unlocking records, 3-20  
unlocking structures, 3-23  
using TITLE attribute, 2-13  
\$INCLUDE compiler control option, use of, 2-14  
FORTRAN77 sample program, C-25  
FORTRAN77ABORTTRANSACTION,  
aborting transactions, 3-48  
FORTRAN77BEGINTRANSACTION,  
beginning transaction state, 3-45  
FORTRAN77CANCELTRPOINT, canceling  
transaction to savepoint, 3-53  
FORTRAN77CLOSE, closing databases,  
3-72, 3-73  
FORTRAN77CREATE, creating new records,  
3-9  
FORTRAN77DATA, constructing data entry  
points dynamically, 4-60

FORTRAN77DELETE, deleting a record, 3-56

FORTRAN77ENDTRANSACTION, ending transaction state, 3-69

FORTRAN77EXCEPTIONNAME, identifying types of exceptions, 5-8

FORTRAN77EXCEPTIONTEXT, returning text of exception message, 5-5

FORTRAN77FIND, finding records, 3-24

FORTRAN77FREE, freeing records, 3-17

FORTRAN77FREESTR, freeing structures, 3-21

FORTRAN77GETBOOLEAN, retrieving Boolean or field items, 4-26

FORTRAN77GETCHARACTER, moving character string to variable, 4-6

FORTRAN77GETDOUBLE, moving double precision to a variable, 4-21

FORTRAN77GETREAL, moving numeric values to variables, 4-16

FORTRAN77KANJI, retrieving Kanji character strings, 4-11

FORTRAN77LOCK, locking records, 3-29

FORTRAN77LOCKSTR, locking structures, 3-34

FORTRAN77OPEN, opening a database, 3-6

FORTRAN77PUTCHARACTER, placing strings into data items, 4-31

FORTRAN77PUTDOUBLE, placing double-precision values into data items, 4-46

FORTRAN77PUTKANJI, placing strings into Kanji alpha items, 4-36

FORTRAN77PUTLOGICAL, setting data items to Boolean values, 4-51

FORTRAN77PUTNULL, setting null values, 4-56

FORTRAN77PUTREAL, placing numeric values into data items, 4-41

FORTRAN77RECREATE, re-creating records, 3-62

FORTRAN77SAVETRPOINT, saving transaction points, 3-50

FORTRAN77SECURE, securing records, 3-37

FORTRAN77SECURESTR, securing structures, 3-42

FORTRAN77SET, setting the current path, 3-13

FORTRAN77SETLIMIT, restricting calls to Accessroutines, 6-1

FORTRAN77STATUS, returning exceptions, 5-3

FORTRAN77STORE, storing records, 3-66

FORTRAN77VERB, executing language extensions, 3-74

free operations, 3-17, 3-21

freeing

- records, 3-17
- structures, 3-21

## G

generating a DMINTERPRETER library, A-1

- ensuring consistency, A-4
- files used, A-1
- from remote terminals, A-5, A-13

generating the DBSTRUCTURE data set, 7-1

getting field items, 4-26

GIVING clause, supplying parameters for COBOL results, 2-12

global data records

- deleting, 3-56
- freeing, 3-17

GLOBAL data set, as entry for DATASET-SUBTYPE, 7-4

global data, in DBSTRUCTURE data set, 7-4

global items, in DMINTERPRETER library stack, 1-9

GROUP entry, for ITEM-SUBTYPE, 7-6

group items

- moving character strings to variables, 4-6
- placing strings into data items, 4-31

group keys, in DBSTRUCTURE data set, 7-3

## H

HASH value, as entry for SET-SEARCH-METHOD, 7-5

< hyphen name >

- moving character strings to variables, 4-7
- moving double-precision value to a variable, 4-23
- moving Kanji character strings to variables, 4-12
- moving numeric values to variables, 4-18
- placing double-precision values into data items, 4-48

## Index

---

- placing numeric values into data items, 4-43
  - placing strings into data items, 4-32
  - placing strings into Kanji alpha items, 4-37
  - retrieving Boolean or field items, 4-28
  - setting Boolean values, 4-52
  - setting null values, 4-57
- I**
- < identifier >
    - declaring DMINTERPRETER library in ALGOL, 2-3
    - invoking DMINTERPRETER library in COBOL, 2-10
  - identifying types of exceptions, 5-8
  - implicitly freeing
    - records, 3-17
    - structures, 3-21
  - importing entry points, 1-4
  - < index set qualified name >
    - finding records, 3-26
    - locking records, 3-31
    - securing records, 3-39
    - setting current path, 3-14
  - index sets
    - finding records, 3-26
    - locking records, 3-31
    - securing records, 3-39
    - setting current paths, 3-13
  - initiating the BUILDINQ program, A-5
  - inquiry access, 3-6
  - INQUIRY value, in opening databases, 3-7
  - < integer >
    - creating new records, 3-10
    - moving character strings to variables, 4-7
    - moving double-precision values to variables, 4-22
    - moving Kanji character strings to variables, 4-12
    - moving numeric values to variables, 4-18
    - placing double-precision values into data items, 4-47
    - placing numeric values into data items, 4-42
    - placing strings into data items, 4-32
    - placing strings into Kanji alpha items, 4-37
    - re-creating records, 3-62
    - retrieving Boolean or field items, 4-27
    - setting Boolean values, 4-52
    - setting null values, 4-57
  - INTEGRITYERROR category
    - exception and error subcategories, B-6
    - major category number, B-2
  - interspersing consecutive calls on structures, 1-8
  - INTLIBERROR
    - messages specific to interpretive interface, 5-1
  - INTLIBERROR category
    - exception and error subcategories, B-7
    - major category number, B-2
  - INUSE category
    - exception and error subcategories, B-8
    - major category number, B-2
  - invoking databases, purpose of, 2-1
  - invoking DMINTERPRETER library
    - in COBOL, 2-9
  - invoking entry points
    - in ALGOL programs, 2-7
    - in COBOL programs, 2-11
    - in FORTRAN77 programs, 2-18
    - purpose of, 2-1
  - IOERROR category
    - exception and error subcategories, B-9
    - major category number, B-2
  - < item name >
    - moving character strings to variables, 4-7
    - moving double-precision values to variables, 4-22
    - moving Kanji character strings to variables, 4-12
    - moving numeric values to variables, 4-18
    - placing double-precision values into data items, 4-47
    - placing numeric values into data items, 4-42
    - placing strings into data items, 4-32
    - placing strings into Kanji alpha items, 4-37
    - retrieving Boolean or field items, 4-27
    - setting Boolean values, 4-52
    - setting null values, 4-57
  - ITEM value, as entry for DB-TYPE, 7-3
  - ITEM-KEY-CLASS item, in DBSTRUCTURE data set, 7-8
  - ITEM-OCCURS item, in DBSTRUCTURE data set, 7-6
  - ITEM-OWNER-GROUP item, in DBSTRUCTURE data set, 7-7
  - ITEM-RECORD-TYPE item, in DBSTRUCTURE data set, 7-8
  - ITEM-REQUIRED item, in DBSTRUCTURE data set, 7-7

ITEM-SCALE-FACTOR item, in  
 DBSTRUCTURE data set, 7-7  
 ITEM-SIGNED item, in DBSTRUCTURE  
 data set, 7-7  
 ITEM-SIZE item, in DBSTRUCTURE data  
 set, 7-7  
 ITEM-SUBSCRIPTS item, in  
 DBSTRUCTURE data set, 7-6  
 ITEM-SUBTYPE item, in DBSTRUCTURE  
 data set, 7-6  
 ITEM-USAGE item, in DBSTRUCTURE  
 data set, 7-8  
 items in DBSTRUCTURE data set, 7-4

## K

Kanji strings  
 moving to variables, 4-11  
 placing strings into alpha items, 4-36  
 using in DBSTRUCTURE data set, 7-8  
 KEYCHANGED category  
 exception and error subcategories, B-10  
 major category number, B-2

## L

language extensions, executing, 3-75  
 LAST, as direction for  
 deleting records, 3-57  
 locking records, 3-30  
 retrieving records, 3-25  
 securing records, 3-38  
 LDBNAME card, A-15  
 leaving transaction state, 3-69  
 libraries, declaring and invoking, 2-1  
 library declaration, ALGOL, 2-2  
 <library declaration>, declaring  
 DMINTERPRETER in ALGOL, 2-2  
 <library entry point declaration>  
 in ALGOL, 2-6  
 in FORTRAN77, 2-16  
 library generation, DMINTERPRETER, A-1  
 <library ID>  
 declaring ALGOL entry points, 2-6  
 declaring DMINTERPRETER library in  
 ALGOL, 2-2  
 declaring DMINTERPRETER library in  
 FORTRAN77, 2-13  
 declaring FORTRAN77 entry points, 2-17

invoking COBOL entry points, 2-11  
 invoking DMINTERPRETER library in  
 COBOL, 2-10  
 LIBRARY statement, in FORTRAN77, 2-13  
 <library statement>, declaring  
 DMINTERPRETER library, 2-13  
 <limit type>, specifying type of linear  
 search, 6-1  
 <limit value>, specifying length of linear  
 search, 6-1  
 LIMITERROR category  
 exception and error subcategories, B-10  
 major category number, B-2  
 limiting find, lock, and secure operations, 6-1  
 limiting number of calls to Accessroutines,  
 6-1  
 linear searches, restricting, 6-1  
 LINEAR value, as entry for  
 SET-SEARCH-METHOD, 7-5  
 LINK value, as entry for DB-TYPE, 7-3  
 LINK-OCCURS item, in DBSTRUCTURE  
 data set, 7-8  
 LINK-TO-DATASET item, in  
 DBSTRUCTURE data set, 7-8  
 <link>  
 deleting records, 3-58  
 finding records, 3-27  
 locking records, 3-31  
 securing records, 3-39  
 lock operations  
 interpretive interface, 3-29  
 restricting linear search, 6-1  
 lock structure operations  
 interpretive interface, 3-34  
 locking  
 records, 3-29  
 structures, 3-34  
 <logical expression>, setting Boolean values  
 in FORTRAN77, 4-53  
 <logical function name>, invoking  
 FORTRAN77 entry points, 2-16  
 LOGICAL FUNCTION, declaring  
 FORTRAN77 entry points, 2-16  
 logical functions, declaring FORTRAN77  
 entry points, 2-16

## M

main interpretive interface software  
 components, 1-2



## Index

---

major categories for exceptions and errors,  
table of, B-2

MAJOR value, as entry for  
ITEM-KEY-CLASS, 7-8

manipulating databases, using standard entry  
points, 3-1

MINOR value, as entry for  
ITEM-KEY-CLASS, 7-8

modifying  
nonrequired items, 3-61  
required items, 3-61

moving  
character strings to variables, 4-6  
data in and out of a DMSII user work area,  
4-3  
double-precision values to variables, 4-21  
numeric value to variables  
double-precision, 4-21  
single-precision, 4-16

**N**

NEXT, as direction for  
deleting records, 3-57  
locking records, 3-30  
retrieving records, 3-25  
securing records, 3-38

NOAUDIT value  
setting when beginning transaction state,  
3-46  
setting when ending transaction state,  
3-69

NONE value, as entry for  
ITEM-KEY-CLASS, 7-8

<nonnumeric literal>, invoking  
DMINTERPRETER library, 2-10

nonrequired items, modifying, 3-61

NORECORD category  
exception and error subcategories, B-10  
major category number, B-2

NOTFOUND category  
exception and error subcategories, B-11  
major category number, B-2

NOTLOCKED category  
exception and error subcategories, B-12  
major category number, B-2

NOUPDATE card, A-16

NOZIP value, generating the  
DMINTERPRETER library, A-13

null values, setting, 4-56

NUMBER entry, for ITEM-SUBTYPE, 7-6

number items  
in DBSTRUCTURE data set, 7-7  
signed number in DBSTRUCTURE data  
set, 7-7  
transferring double-precision values to  
variables, 4-21  
transferring values to variables, 4-16

## O

OCCURS clause, in DBSTRUCTURE data  
set, 7-6, 7-8

open disposition, setting access mode for open  
operation, 3-6

<open disposition>, in opening DMSII  
database, 3-7

open operations, 3-6

OPENERERROR category  
exception and error subcategories, B-12  
major category number, B-2

opening databases, 3-6  
accessing a database, 2-1  
UPDATE versus INQUIRY access mode,  
3-7

operations  
aborting transactions, 3-48, 3-53  
beginning transaction state, 3-45  
canceling transactions points, 3-53  
closing DMSII databases, 3-72  
creating new records, 3-9

DBGET  
constructing data entry points  
dynamically, 4-60  
moving character strings to variables,  
4-6  
moving double-precision values to  
variables, 4-21  
moving Kanji strings to variables, 4-11  
moving numeric values to variables,  
4-16  
retrieving Boolean or field items, 4-26

DBPUT  
constructing data entry points  
dynamically, 4-60  
placing double-precision values into data  
items, 4-46  
placing numeric values into data items,  
4-41  
placing strings into data items, 4-31  
placing strings into Kanji alpha items,  
4-36

- setting data items to Boolean values, 4-51
    - setting data items to null values, 4-56
  - deleting data records, 3-56
  - ending transaction state, 3-69
  - entering transaction state, 3-45
  - executing language extensions, 3-74
  - finding records, 3-24
  - freeing records, 3-17
  - freeing structures, 3-21
  - locking records, 3-29
  - locking structures, 3-34
  - opening databases, 3-6
  - re-creating records, 3-61
  - reserving space for new records, 3-9
  - restricting number of calls to
    - Accessroutines, 6-1
  - returning exception word, 5-3
  - returning text of exception message, 5-5
  - returning type of exception message, 5-8
  - saving transaction points, 3-50
  - securing records, 3-37
  - securing structures, 3-42
  - setting the current path, 3-13
  - storing records, 3-66
- P**
- < parameter >, invoking FORTRAN77 entry points, 2-16
  - parameters, using in entry points, 1-7
  - passing parameters, in COBOL languages, 2-11
  - placing
    - double-precision values into data items, 4-46
    - numeric values into data items, 4-41
    - strings into data items, 4-31
    - strings into Kanji alpha items, 4-36
  - POPULATION entry, for ITEM-SUBTYPE, 7-6
  - population items
    - transferring double-precision values to variables, 4-21
    - transferring values to variables, 4-16
  - PRIOR, as direction for
    - deleting records, 3-57
    - locking records, 3-30
    - retrieving records, 3-25
    - securing records, 3-38
  - procedure declaration, for ALGOL entry points, 2-5, 2-6
  - < procedure heading >, declaring ALGOL entry points, 2-6
  - < procedure ID >
    - declaring ALGOL entry points, 2-6
    - invoking COBOL entry points, 2-11
  - programming considerations and guidelines, 1-7, 2-1
    - for standard entry points, 3-3
    - reducing run-time overhead, 1-8
    - sequencing entry points, 1-8
  - protected state, 3-45
- Q**
- QUEUE card
    - remote generation of DMINTERPRETER, A-14
    - WFL generation of DMINTERPRETER, A-15
- R**
- Railroad diagrams, explanation of, D-1
  - RANDOM data set, as entry for DATASET-SUBTYPE, 7-4
  - random retrieval, 3-24
  - re-create operations, 3-62
  - re-creating records, 3-61
  - READONLY category
    - exception and error subcategories, B-15
    - major category number, B-2
  - REAL entry, for ITEM-SUBTYPE, 7-6
  - real items
    - in DBSTRUCTURE data set, 7-7
    - transferring double-precision values to variables, 4-21
    - transferring values to variables, 4-16
  - < record statement >, executing through entry point, 3-75
  - < record type >
    - creating new records, 3-10
    - re-creating records, 3-63
  - record-type items
    - transferring double-precision values to variables, 4-21
    - transferring values to variables, 4-16
  - records

## Index

---

- creating, 3-9
  - deleting, 3-56
  - finding, 3-24
  - freeing, 3-17
  - locking, 3-29
  - re-creating, 3-61
  - securing, 3-37
  - storing, 3-66
  - RECOVER option, use of NOZIP value, A-13
  - reducing overhead guidelines, 1-8
  - release tape, declaring entry points for
    - ALGOL programs, 2-4
    - FORTRAN77 programs, 2-14
    - supported COBOL languages, 2-8
  - releasing
    - records, 3-17
    - structures, 3-21
  - remote generation of DMINTERPRETER library, A-5, A-13
  - remote terminal, generating
    - DMINTERPRETER library, A-5, A-13
  - renaming entry points, 1-6
    - in ALGOL, 2-4, 2-6
    - in FORTRAN77, 2-14, 2-17
  - renaming hyphenated database names, in BUILDINQ program, A-11
  - reporting errors, using DMSUPPORT library, 1-4
  - required items
    - explanation of, 7-2
    - modifying, 3-61
  - reserving space for new records, 3-9
  - RESTART data set, as entry for DATASET-SUBTYPE, 7-4
  - restart data sets
    - when beginning transaction state, 3-45
    - when ending transaction state, 3-69
  - restricting
    - calls to the Accessroutines, 6-1
    - linear searches, 6-1
    - number of calls to the Accessroutines, 6-1
  - result descriptor, as DMSII exception word, 5-1
  - retrieving
    - Boolean items, 4-26
    - field items, 4-26
    - Kanji alpha character strings, 4-11
  - returning
    - exception word operations, 5-3
    - exceptions, 1-7, 5-3
    - text of exception messages, 5-5
    - type of exception, 5-8
  - run-time interface, 1-1
  - run-time libraries, purpose of, 1-4
- ## S
- sample programs
    - ALGOL, C-5
    - COBOL74, C-15
    - DASDL source for, C-2
    - FORTRAN77, C-25
  - save transaction point operations, 3-50
    - < savepoint >
      - canceling transactions to savepoints, 3-53
      - saving transaction points, 3-50
    - saving transaction points, 3-50
    - < search statement >, executing through entry point, 3-76
  - secure operations
    - interpretive interface, 3-37
    - restricting linear search, 6-1
  - secure structure operations
    - interpretive interface, 3-42
  - securing
    - records, 3-37
    - structures, 3-42
  - SECURITYERROR category
    - exception and error subcategories, B-16
    - major category number, B-2
  - selecting data sets, in BUILDINQ program, A-10
  - selecting the database, in BUILDINQ program, A-5
  - selecting view of database, in BUILDINQ program, A-6
  - < selection expression >
    - deleting records, 3-58
    - finding records, 3-27
    - locking records, 3-31
    - securing records, 3-39
  - sequence of interpretive interface tasks, guidelines, 1-8
  - sequence range in
    - DATABASE/DMINTERPRETER
      - for ALGOL declarations and entry points, 2-4
      - for FORTRAN77 declarations and entry points, 2-14
  - sequential retrieval, 3-24
  - set operations, 3-13
    - using with DBSTRUCTURE data set, 7-1

- < set statement > , executing through entry point, 3-75
- SET value , as entry for DB-TYPE, 7-3
- SET value, as entry for SET-SUBTYPE, 7-5
- SET-SEARCH-METHOD item, in
  - DBSTRUCTURE data set, 7-5
- SET-SPANS-DATASET item, in
  - DBSTRUCTURE data set, 7-5
- SET-SUBTYPE item, in DBSTRUCTURE data set, 7-5
- sets, defining types of in DBSTRUCTURE data set, 7-5
- setting
  - Boolean values, 4-51
  - current paths, 3-13
  - data items to null values, 4-56
  - variables to a character string, 6-1
- STANDARD data set, as entry for DATASET-SUBTYPE, 7-4
- standard entry points, 3-1
  - ALGOABORTTRANSACTION, 3-48
  - ALGOBEGINTRANSACTION, 3-45
  - ALGOCANCELTRPOINT, 3-53
  - ALGOCLOSE, 3-72
  - ALGOLCREATE, 3-9
  - ALGOLDELETE, 3-56
  - ALGOENDTRANSACTION, 3-69
  - ALGOLFIND, 3-24
  - ALGOLFREE, 3-17
  - ALGOLFREESTR, 3-21
  - ALGOLLOCK, 3-29
  - ALGOLLOCKSTR, 3-34
  - ALGOOPEN, 3-6
  - ALGOLRECREATE, 3-62
  - ALGOLSAVETRPOINT, 3-50
  - ALGOLSECURE, 3-37
  - ALGOLSECURESTR, 3-42
  - ALGOLSET, 3-13
  - ALGOLSTORE, 3-66
  - ALGOLVERB, 3-74
  - DBABORTTRANSACTION, 3-48
  - DBBEGINTRANSACTION, 3-45
  - DBCANCELTRPOINT, 3-53
  - DBCLOSE, 3-72, 3-73
  - DBCREATE, 3-9
  - DBDELETE, 3-56
  - DBENDTRANSACTION, 3-69
  - DBFIND, 3-24
  - DBFREE, 3-17
  - DBFREESTR, 3-21
  - DBLOCK, 3-29
  - DBLOCKSTR, 3-34
  - DBOPEN, 3-6
  - DBRECREATE, 3-62
  - DBSAVETRPOINT, 3-50
  - DBSECURE, 3-37
  - DBSECURESTR, 3-42
  - DBSET, 3-13
  - DBSTORE, 3-66
  - DEVERB, 3-74
  - FORTTRAN77ABORTTRANSACTION, 3-48
  - FORTTRAN77BEGINTRANSACTION, 3-45
  - FORTTRAN77CANCELTRPOINT, 3-53
  - FORTTRAN77CLOSE, 3-72, 3-73
  - FORTTRAN77CREATE, 3-9
  - FORTTRAN77DELETE, 3-56
  - FORTTRAN77ENDTRANSACTION, 3-69
  - FORTTRAN77FIND, 3-24
  - FORTTRAN77FREE, 3-17
  - FORTTRAN77FREESTR, 3-21
  - FORTTRAN77LOCK, 3-29
  - FORTTRAN77LOCKSTR, 3-34
  - FORTTRAN77OPEN, 3-6
  - FORTTRAN77RECREATE, 3-62
  - FORTTRAN77SAVETRPOINT, 3-50
  - FORTTRAN77SECURE, 3-37
  - FORTTRAN77SECURESTR, 3-42
  - FORTTRAN77SET, 3-13
  - FORTTRAN77STORE, 3-66
  - FORTTRAN77VERB, 3-74
  - purpose of, 1-4
  - table of, 3-1
- < status statement > , executing through entry point, 3-76
- store operations, 3-66
- storing records, 3-66
- string variables, use of preinitialized, 1-8
- < string >
  - declaring DMINTERPRETER library in FORTRAN77, 2-13
  - declaring FORTRAN77 entry points, 2-17
- STRUCTURE data set, as entry for DATASET-SUBTYPE, 7-4
- < structure >
  - finding records, 3-26
  - freeing records, 3-18
  - locking records, 3-30
  - moving character strings to variables, 4-7
  - moving double-precision values to variables, 4-22
  - moving Kanji character strings to variables, 4-12

## Index

---

- moving numeric values to variables, 4-17
- placing double-precision values into data items, 4-47
- placing numeric values into data items, 4-42
- placing strings into data items, 4-32
- placing strings into Kanji alpha items, 4-37
- retrieving Boolean or field items, 4-27
- securing records, 3-38
- setting Boolean values, 4-52
- setting current path, 3-14
- setting null values, 4-56
- storing records, 3-66
- structures
  - locking, 3-34
  - securing, 3-42
- subcategories for exceptions and errors, tables of, B-2
- SUBSET value, as entry for SET-SUBTYPE, 7-5
- summary table of exceptions and errors, B-18
- symbolic name, in FORTRAN77, 2-13
  - <symbolic name>, declaring
    - DMINTERPRETER library, 2-13
- SYNC value, setting when ending transaction state, 3-69
- syncpoints, setting when ending transaction state, 3-69
- SYSTEM/BUILDING, generating
  - DMINTERPRETER library, A-1
- SYSTEM/INTERFACE, generating
  - DMINTERPRETER library, A-1
- SYSTEMERROR category
  - exception and error subcategories, B-16
  - major category number, B-2

## T

- table of exceptions and errors, B-18
- <target>, setting current path, 3-14
- TITLE attribute
  - declaring DMINTERPRETER library in
    - ALGOL, 2-2
    - FORTRAN77, 2-13
  - invoking DMINTERPRETER library in
    - COBOL, 2-10
- transaction state
  - aborting, 3-48
  - beginning, 3-45
  - canceling, 3-53

- deleting records, 3-56
- ending, 3-69
- saving a transaction point, 3-50
- storing records, 3-66
- <transaction statement>, executing through entry point, 3-75
- transferring data
  - constructing transfers during program execution, 4-60
  - moving character strings to variables, 4-6
  - moving double-precision values to variables, 4-21
  - moving Kanji alpha character strings to variables, 4-11
  - moving numeric values to variables, 4-16
  - placing double-precision values into data items, 4-46
  - placing numeric values into data items, 4-41
  - placing strings into data items, 4-31
  - placing strings into Kanji alpha items, 4-36
  - retrieving Boolean values, 4-26
  - setting data items to Boolean values, 4-51
  - setting data items to null values, 4-56
- TYPE entry, for ITEM-SUBTYPE, 7-6
- types of
  - application program interfaces, 1-1
  - data sets in DBSTRUCTURE data set, 7-4
  - entities in DBSTRUCTURE, 7-3
  - sets in DBSTRUCTURE data set, 7-5

## U

- <underscore name>
  - moving character strings to variables, 4-7
  - moving double-precision values to variables, 4-22
  - moving Kanji character strings to variables, 4-12
  - moving numeric values to variables, 4-18
  - placing double-precision values into data items, 4-47
  - placing numeric values into data items, 4-42
  - placing strings into data items, 4-32
  - placing strings into Kanji alpha items, 4-37
  - retrieving Boolean or field items, 4-27
  - setting Boolean values, 4-52
  - setting null values, 4-57
- unlocking
  - records, 3-17

structures, 3-21  
UNORDERED data set, as entry for  
    DATASET-SUBTYPE, 7-4  
update access, 3-6  
UPDATE value, in opening databases, 3-7  
use of DASDL options  
    ZIP, A-5  
user work area  
    deleting records, 3-56  
    finding records, 3-24  
using BUILDINQ program, A-4  
USING clause, supplying parameters for  
    COBOL entry points, 2-11  
using the release tape  
    declaring ALGOL entry points, 2-4  
    declaring COBOL entry points, 2-8  
    declaring FORTRAN77 entry points, 2-14

\$INCLUDE compiler control option  
    reasons to use, 1-5  
    use in ALGOL programs, 2-4  
    use in FORTRAN77 programs, 2-14

## V

<variable name >  
    moving character strings to variables, 4-8  
    moving double-precision values to  
        variables, 4-23  
    moving Kanji strings to variables, 4-13  
    moving numeric values to variables, 4-18  
    retrieving Boolean values, 4-28  
variable-format record  
    creating new records, 3-10  
    recreating records, 3-63  
verification message, in BUILDINQ program,  
    A-12  
VERSIONERROR category  
    exception and error subcategories, B-18  
    major category number, B-2

## W

Work Flow Language  
    generating the DMINTERPRETER library,  
        A-15

## Z

ZIP option, generating the  
    DMINTERPRETER library, A-5









86000155-000