UNISYS

A Series
COBOL ANSI-74
Programming Reference Manual
Volume 2: Product Interfaces

Release 3.9.0

Priced Item

September 1991

U S America
8600 0130-000

UNISYS

Product Information Announcement

○ New Release ○ Revision ● Update ○ New Mail Code

Title

A Series COBOL ANSI-74 Programming Reference Manual, Volume 2: Product Interfaces

This Product Information Announcement announces the release of Update 1 to the *A Series COBOL ANSI-74 Programming Reference Manual, Volume 2: Product Interfaces,* dated September 1991, relative to the Mark 3.9.0 System Software Release.

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 the Semantic Information Manager (SIM) products. This manual is written for programmers who are familiar with COBOL74 programming language concepts and terms.

Update 1 includes varied technical changes and clarifications that have developed since publication of the 3.9.0 version of this manual.

Changes to the text are indicated by vertical bars in the margins of the replacement pages.

Remove	Insert
iii through iv	iii through iv
vii through x	vii through x
xiii through xxiv	xiii through xxiv
1–5 through 1–6	1–5 through 1–6
2–1 through 2–2	2–1 through 2–2
2–5 through 2–10	2–5 through 2–10
2–13 through 2–14	2–13 through 2–14
2–19 through 2–20	2–19 through 2–20
3–3 through 3–6	3–3 through 3–6
3–9 through 3–10	3–9 through 3–10B
3–23 through 3–26	3–23 through 3–26
3–47 through 3–48	3–47 through 3–48
3–55 through 3–56	3–55 through 3–56
	4–18A through 4–18B
4–51 through 4–52	4–51 through 4–52B
6–1 through 6–26	6–1 through 6–30
7–1 through 7–8	7–1 through 7–8
7–17 through 7–20	7–17 through 7–20
8–1 through 8–6	8–1 through 8–2B
	8–3 through 8–6
8–13 through 8–14	8–13 through 8–14

continued

Announcement only:

Announcement and attachments:
AS121

System: A Series
Release: Mark 4.0.0 July 1992

Part number: 8600 0130–010

Remove	Insert
8–37 through 8–40	8–37 through 8–40
8–57 through 8–58	8–57 through 8–58B
8–65 through 8–72	8–65 through 8–72
8–77 through 8–78	8–77 through 8–78
Bibliography–1 through 2	Bibliography–1 through 2
Index–1 through 32	Index–1 through 32

Retain this Product Information Announcement as a record of changes made to the base publication.

To order additional copies of this document

- United States customers call Unisys Direct at 1-800-448-1424
- All other customers contact your Unisys Subsidiary Librarian
- Unisys personnel use the Electronic Literature Ordering (ELO) system

UNISYS

A Series
COBOL ANSI-74
Programming Reference Manual
Volume 2: Product Interfaces

Printed on recycled paper

Release 3.9.0 September 1991

Priced Item

U S America
8600 0130–000

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 through iv	–010
v through vi	–000
vii through x	–010
xi	–000
xii	Blank
xiii through xix	–010
xx	Blank
xxi	–010
xxii	Blank
xxiii	–010
xxiv	Blank
1–1 through 1–4	–000
1–5 through 1–6	–010
1–7 through 1–8	–000
2–1 through 2–2	–010
2–3 through 2–4	–000
2–5 through 2–10	–010
2–11 through 2–12	–000
2–13 through 2–14	–010
2–15 through 2–18	–000
2–19 through 2–20	–010
2–21 through 2–22	–000
3–1 through 3–2	–000
3–3 through 3–6	–010
3–7 through 3–8	–000
3–9 through 3–10A	–010
3–10B	Blank
3–11 through 3–22	–000
3–23 through 3–26	–010
3–27 through 3–46	–000
3–47 through 3–48	–010
3–49 through 3–54	–000
3–55 through 3–56	–010
3–57 through 3–61	–000
3–62	Blank
4–1 through 4–18	–000
4–18A	–010
4–18B	Blank
4–19 through 4–50	–000

continued

Page Status

continued

Page	Issue
4–51 through 4–52A	–010
4–52B	Blank
4–53 through 4–82	–000
5–1 through 5–51	–000
5–52	Blank
6–1 through 6–29	–010
6–30	Blank
7–1 through 7–8	–010
7–9 through 7–16	–000
7–17 through 7–20	–010
7–21 through 7–34	–000
8–1 through 8–2A	–010
8–2B	Blank
8–3 through 8–6	–010
8–7 through 8–12	–000
8–13 through 8–14	–010
8–15 through 8–36	–000
8–37 through 8–40	–010
8–41 through 8–56	–000
8–57 through 8–58A	–010
8–58B	Blank
8–59 through 8–64	–000
8–65 through 8–72	–010
8–73 through 8–76	–000
8–77 through 8–78	–010
8–79 through 8–84	–000
A–1 through A–5	–000
A–6	Blank
B–1	–000
B–2	Blank
Glossary–1 through 31	–000
Glossary–32	Blank
Bibliography–1 through 2	–010
Index–1 through 31	–010
Index–32	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

This manual is the second volume of a two-volume reference set. The *A Series COBOL ANSI-74 Programming Reference Manual, Volume 1: Basic Implementation* provides the syntax and general usage of standard COBOL74. Volume 2 contains information on using Unisys standard COBOL74 to write application programs that interface with the following products:

- Advanced Data Dictionary System (ADDS)
- Communications Management System (COMS)
- Data Management System II (DMSII)
- DMSII transaction processing system (TPS)
- Screen Design Facility (SDF)
- Screen Design Facility Plus (SDF Plus)
- Semantic Information Manager (SIM)

SDF is a member of the InterPro™ (Interactive Productivity) family of products. ADDS and SIM are members of the InfoExec™ (Information Executive) family of products.

Scope

For each product, Volume 2 presents information on

- The purposes of COBOL74 interfaces with the product
- The product features and functions that an interface can manipulate
- The uses of the language extensions of a product
- The means by which each language extension is used
- The information necessary for writing COBOL74 programs that use combined products

The Unisys standard COBOL74 covered in the manual is implemented for use on A Series systems. Unisys standard COBOL74 is based on, and compatible with, the American National Standard, X3.23-1974.

InfoExec and InterPro are trademarks of Unisys Corporation.

COBOL is an industry language and is not the property of any company or group of companies, or of any organization or group of organizations.

No warranty, expressed or implied, is made by any contributor or by the Conference on Data Systems Language (CODASYL) as to the accuracy and functioning of the programming system and language. Moreover, no responsibility is assumed by any contributor, or by the committee in connection therewith.

The authors and copyright holders of the copyrighted material used herein have specifically authorized the use of this material, in whole or in part, in the COBOL specifications. These authors or copyright holders are the following:

- FLOW-MATIC, programming for the UNIVAC I and II, Data Automation Systems, copyrighted in 1958, by Sperry Rand Corporation
- IBM Commercial Translator, form No. F 28-8013, copyrighted in 1959 by the International Business Machines Corporation
- FACT, DSI 27 A5260-2760, copyrighted in 1960 by Minneapolis-Honeywell

Such authorization extends to the reproduction and use of COBOL specifications in programming manuals or similar publications.

Audience

The information in this manual is intended for application programmers who are programming in COBOL74 and require the capabilities of one or more products.

Prerequisites

To use this manual, you should be familiar with COBOL74, the product or products being used, and the programming concepts for the products.

How to Use This Manual

Throughout this manual, *Volume 1* refers to the *A Series COBOL ANSI-74 Programming Reference Manual, Volume 1: Basic Implementation.*

This manual is structured so that you only need to look at the sections that apply to your needs. The first section summarizes the extensions used with each product. After this introductory section, a separate section is provided for each product. This volume contains information that complements Volume 1 and the manuals that describe the interfaced products. For a discussion of those products, refer to the documentation for each product. A list of this documentation is provided in "Related Product Information" later in this preface. For the usage of Unisys Standard COBOL74 not specific to the products in this volume, refer to Volume 1.

Many of the discussions in this volume use COBOL74 code to illustrate an aspect of a product interface. Most examples do not include line numbers; it is assumed that the

first character of a line of source code is located in the appropriate column. Complete program examples have line numbers to provide continuity for programs that span several pages.

The glossary includes terms whose definitions help you to understand the functions and extensions described in this volume. For definitions of general Unisys Standard COBOL74 terms, refer to Volume 1. For definitions of product-specific terms, refer to the manual for that product.

The glossary defines and spells out each acronym found in this volume. Acronyms are also spelled out the first time they occur in the manual.

The syntax for the language extensions is described in COBOL notation; for a complete explanation of COBOL format notation, refer to Volume 1.

Unless otherwise specified, manuals referred to in the text are for A Series systems. The full title of a manual is used the first time the manual is referenced; shortened titles are used in subsequent references.

Organization

This volume is organized into eight sections. The first section is an introduction to program interfaces, and each subsequent section describes a program interface for a specific product. Appendixes are included to provide related information. A glossary, an index, and a bibliography are provided at the end of the volume. A brief description of the contents of the manual follows.

Section 1. Introduction to COBOL74 Program Interfaces

This section describes the concept of a COBOL74 interface with a product and introduces the product sections that follow. It includes information on the extensions covered in each product section, and the extensions that apply when products are used in combination.

Section 2. Using the ADDS Program Interface

This section describes how the ADDS interface invokes entities from the data dictionary, tracks programs, selects processing options, and sets entity status. It covers the requirements and options for using ADDS with other products, and presents the syntax, explanations, and program examples of the language extensions. It also references information on defining prefixes and synonyms and about entity tracking for programs.

Section 3. Using the COMS Program Interface

This section introduces the COMS interface and gives an overview of the extensions and their relationships to the product features. COMS provides a general message control system. This section describes the syntax and explanation for each extension and provides program examples. Information is also provided on using COMS with other products.

Section 4. Using the DMSII Program Interface

This section contains information on the extensions developed for the DMSII interface that invoke databases, use data management statements and database items, and handle exception conditions. This section also contains an explanation of the syntax and examples of extensions.

Section 5. Using the DMSII TPS Program Interface

This section outlines the features of TPS and describes the extensions that invoke a transaction base, create transaction records, use the transaction library entry points, and process transactions. The section also includes program samples.

Section 6. Using the SDF Program Interface

This section summarizes the features of SDF and explains the extensions that process forms, use message keys, and manipulate programmatic controls. The syntax and explanation of extensions are included as well as program examples. The section briefly discusses the use of other products with SDF.

Section 7. Using the SDF Plus Program Interface

This section contains the extensions developed for the SDF Plus interface. These extensions define a complete form-based user interface for COBOL74 application systems.

Section 8. Using the SIM Program Interface

This section describes the extensions for the SIM interface. The extensions manipulate data stored in a SIM database. The section provides information about declaring queries, performing transactions, using functions and expressions in a program, and handling exceptions. The text includes syntax, explanations, and examples.

Appendix A. Reserved Words

This appendix lists COBOL reserved words. A reserved word has a special meaning to COBOL and cannot be redefined by the programmer.

Appendix B. User-Defined Words

This appendix lists variable words or terms required in clauses or statements for which you must define names.

Related Product Information

***A Series COBOL ANSI-74 Programming Reference Manual, Volume 1: Basic Implementation* (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 Test and Debug System (TADS) Programming Guide* (1169901)**

This guide documents COBOL74 TADS, an interactive tool for testing and debugging COBOL74 programs and libraries. This guide is written for programmers familiar with COBOL74 programming language concepts and terms.

***A Series Communications Management System (COMS) Programming Guide* (8600 0650)**

The guide explains how to write online, interactive, and batch application programs that run under COMS. This guide is written for experienced applications programmers with knowledge of data communication subsystems.

***A Series DMSII Application Program Interfaces Programming Guide* (5044225)**

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 Transaction Processing System (TPS) Programming Guide* (1164043)**

This guide describes the various modules of TPS and provides information on the TPS library of transaction processing procedures. This guide is intended for experienced systems programmers who are familiar with DMSII.

***A Series InfoExec Advanced Data Dictionary System (ADDS) Operations Guide* (8600 0197)**

This guide describes InfoExec ADDS operations, such as creating and managing database descriptions. This guide is written for those who collect, organize, define, and maintain data and who are familiar with the Data Management System II (DMSII), the Semantic Information Manager (SIM), and with the Structured Query Language Database (SQLDB).

***A Series InfoExec Semantic Information Manager (SIM) Object Manipulation Language (OML) Programming Guide* (8600 0163)**

This guide describes how to interrogate and update SIM databases using SIM OML. Also described are two methods for processing queries: one method embeds calls on the SIM library in an application program and the other method uses the InfoExec Interactive

Query Facility (IQF). This guide is written for application programmers and experienced IQF and Workstation Query Facility (WQF) users.

***A Series InfoExec Semantic Information Manager (SIM) Programming Guide* (8600 1666)**

This guide describes how to use Unisys value-added language extensions to access InfoExec SIM databases from application programs written in COBOL74, Pascal, and ALGOL. This guide is written for programmers who know at least one of these programming languages thoroughly and who are familiar with SIM.

***A Series InfoExec Semantic Information Manager (SIM) Technical Overview* (8600 1674)**

This overview describes the SIM concepts on which the InfoExec data management system is based. This overview is written for end users, applications programmers, database designers, and database administrators.

***A Series Screen Design Facility (SDF) Operations and Programming Guide* (1185295)**

This guide explains how to install SDF. It gives detailed instructions on interactively defining fields, forms, and formlibraries, painting form images, and generating formlibraries. It also provides suggestions and examples for writing applications that use SDF features effectively. This guide is written for application programmers.

***A Series Screen Design Facility Plus (SDF Plus) Capabilities Manual* (8600 0270)**

This manual describes the capabilities and benefits of SDF Plus. It gives a general introduction to the product and explains the differences between SDF and SDF Plus. This manual is written for executive and data processing management.

***A Series Screen Design Facility Plus (SDF Plus) Installation and Operations Guide* (8600 0262)**

This guide explains how to use SDF Plus to create and maintain a user interface. It gives specific instructions for installing SDF Plus, using the SDF Plus forms, and installing and running a user interface created with SDF Plus.

***A Series Screen Design Facility Plus (SDF Plus) Technical Overview* (8600 0072)**

This overview provides the conceptual information needed to use SDF Plus effectively to create user interfaces.

***A Series System Software Utilities Operations Reference Manual* (8600 0460)**

This manual provides information on the system utilities, such as DCSTATUS, FILECOPY, and DUMPALL. This manual is written for applications programmers and operators.

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.

Contents

Section 8. Using the SIM Program Interface

Examples

Tables

Section 1
Introduction to COBOL74 Program Interfaces

A program interface comprises the syntax, conventions, and protocols of a programming language that are used to manipulate a software product. Interfaces are developed to aid you in writing applications that use the full functionality of Unisys products.

Using Program Interfaces for Specific Products

COBOL74 program interfaces for products are easy to use, flexible, and efficient. The interfaces are the Unisys COBOL74 extensions to ANSI-74 COBOL. These extensions make it easier to manipulate special product features.

COBOL74 program interfaces have been developed for each of the following products:

- Advanced Data Dictionary System (ADDS)
- Communications Management System (COMS)
- Data Management System II (DMSII)
- DMSII transaction processing system (TPS)
- Screen Design Facility (SDF)
- Screen Design Facility Plus (SDF Plus)
- Semantic Information Manager (SIM)

Each program interface is described in a section of this manual. Each section includes suggested ways to implement some of the features in your application of the product, including any requirements or options for using a combination of products.

Using Language Extensions for Specific Products

The extensions for Unisys standard COBOL74 comprise two groups:

- Extensions used only with specific products
- Extensions used with all Unisys products

The extensions used only with specific products are explained in this volume. The extensions used with all Unisys products are explained in Volume 1 of this manual; however, these general extensions might appear in this volume to provide context and to illustrate their use with specific products.

The following information briefly describes the extensions used only with specific products. The extensions for each product are presented alphabetically. Detailed discussions of the extensions are provided in the appropriate sections of this volume.

ADDS Extensions

The following extensions have been developed for the ADDS interface. Detailed information on the extensions is provided in Section 2, "Using the ADDS Program Interface."

Extension	Explanation
DICTIONARY compiler control option	Enables the compiler to access entities within a specified status.
DICTIONARY statement	Identifies a dictionary and provides optional information about a program to be tracked.
DIRECTORY clause	Uniquely identifies an entity in the data dictionary when used with an entity name and optional version number.
File description (FD)	Provides information about the physical structure and record names of a file.
FROM DICTIONARY clause	Invokes a structure from the dictionary.
INVOKE clause	Assigns an alias identifier to a data name.
INVOKE ALL clause	Invokes the entire structure, including record descriptions of a specified file from the dictionary.
SELECT statement	Includes a file definition from the dictionary.
Sort-merge file description (SD)	Provides information about the physical structure, identification, and record names of a file to be sorted.
VERSION clause	Uniquely identifies an entity in the dictionary when used with an entity name and optional directory.

COMS Extensions

The following extensions have been developed for the COMS interface. Detailed information on the extensions is provided in Section 3, "Using the COMS Program Interface."

Extension	Explanation
ACCEPT MESSAGE COUNT statement	Makes available the number of messages in a program application queue.
COMS header declaration	Provides information about the message for input or output.
DISABLE statement	Closes a COMS direct window to a station, or disconnects a station reached through a modem or a Communications Processor 2000.
ENABLE statement	Initializes the interface between COMS and a program, and opens a COMS direct window to a station not currently attached to COMS.

continued

continued

Extension	Explanation
RECEIVE statement	Makes a message and pertinent information available to the program from a queue maintained by the message control system (MCS).
SEND statement	Releases a message or message segment to one or more output queues maintained by the MCS.
VALUE clause	Accesses COMS service function routines and enables a mnemonic parameter to be passed to obtain a numeric result. The VALUE clause appears in the CALL statement.

DMSII Extensions

The following extensions have been developed for the DMSII interface. Detailed information on each extension is provided in Section 4, "Using the DMSII Program Interface."

Extension	Explanation
ABORT-TRANSACTION statement	Discards updates made in a transaction after a BEGIN-TRANSACTION statement.
ASSIGN statement	Establishes the relationship between a record in a data set and a record in the same or another data set.
BEGIN-TRANSACTION statement	Places a program in transaction state. This statement is used only with audited databases.
CANCEL TRANSACTION POINT statement	Discards all updates in a transaction to an intermediate transaction point or to the beginning of the transaction.
CLOSE statement	Closes a database unconditionally when further access is no longer required. A syncpoint is caused in the audit trail, and all locked records are freed.
COMPUTE statement	Provides an extension to the standard COMPUTE statement. This data management statement assigns a value to a Boolean item in the current record of a data set.
CREATE statement	Initializes the user work area of a data set record.
Database declaration	Provides information about one or more databases during compilation.
Database equation	Enables the database to be specified at run time, and allows access to databases under different usercodes and on packs not visible to a task.
Data management (DM) attributes	Enable read-only access to the count, record type, and population information in a record.
Data set reference entry	Specifies the structures that are to be invoked from the declared database.
DELETE statement	Finds a specific record, and then locks and deletes it.
DMERROR Use procedure	Handles exception conditions.

continued

continued

Extension	Explanation
DMSTATUS database status word	Indicates whether an exception condition has occurred and identifies the exception.
DMSTRUCTURE function	Determines the structure number of a data set, set, or subset programmatically. This data management structure function can be used to analyze exception condition results.
DMTERMINATE statement	Halts a program with a fault when an exception occurs that the program does not handle.
END-TRANSACTION statement	Takes a program out of transaction state. This data management statement is used only with audited databases.
FIND statement	Transfers a record to the work area associated with a data set or global data.
FREE statement	Unlocks the current record.
GENERATE statement	Creates a subset in one operation. All subsets must be disjoint bit vectors.
IF statement	Provides an extension to the standard IF statement. The statement tests an item to determine if it contains a NULL value.
INSERT statement	Places a record into a manual subset.
LOCK/MODIFY statement	Finds a record or structure, and locks it against concurrent modification by another user.
ON EXCEPTION option	Handles exception conditions. The clause is placed after certain data management statements to handle exception conditions.
OPEN statement	Opens a database for subsequent access and designates the access mode.
RECREATE statement	Partially initializes the user work area.
REMOVE statement	Finds a record, and then locks it and removes it from the subset.
SAVE TRANSACTION POINT statement	Provides an intermediate point in a transaction for audit.
SECURE statement	Locks a record in such a way that other programs can read the record but not update it.
Selection expression	Identifies a certain record in a data set. Selection expressions are used with FIND, LOCK, MODIFY, and DELETE statements.
SET statement	Alters the current path or changes the value of an item in the current record.
STORE statement	Places a new or modified record into a data set.

DMSII TPS Extensions

The following extensions have been developed for the DMSII TPS interface. More information on the extensions is provided in Section 5, "Using the DMSII TPS Program Interface."

Extension	Explanation
BEGIN-TRANSACTION statement	Places a program in transaction state. This statement is used only with audited databases.
Compile-time function	Provides access to constant properties of the transaction record formats at compile time.
CREATE statement	Defines and initializes the contents of a transaction record variable to a particular format.
END-TRANSACTION statement	Takes a program out of transaction state. This statement is used only with audited databases.
MID-TRANSACTION statement	Causes the compiler to generate calls on the SAVEINPUTTR procedure before the call on the DMSII procedure in Accessroutines.
OPEN statement	Provides the TRUPDATE option to open a database for subsequent access, designates the access mode, and enables the use of the BEGIN-TRANSACTION, END-TRANSACTION, and MID-TRANSACTION statements.
Transaction base declaration	Invokes a transaction base or subbase into a program.
Transaction record control item reference	Inquires about record control items in a transaction record for read-only access.
Transaction record reference entry	Accesses a transaction record item in a declared format and subformat.
Transaction record variable declaration	Associates the records for a given transaction with the transaction base.
USE statement	Passes transaction records as parameters to transaction library procedures.

SDF Extensions

The following extensions have been developed for the SDF interface. Detailed information about the extensions is provided in Section 6, "Using the SDF Program Interface." Related extensions are described in Section 2, "Using the ADDS Program Interface," and Section 3, "Using the COMS Program Interface."

Extension	Explanation
DICTIONARY statement	Identifies the dictionary to be used during compilation.
FORM-KEY function	Enables access to an internal binary number of a form name for programmatic uses. This function is required for using SDF with COMS.
FROM DICTIONARY clause	Invokes a formlibrary from the dictionary.
READ FORM statement	Reads specific and self-identifying forms.

continued

continued

Extension	Explanation
REDEFINES clause	Enables formlibraries invoked into a program to redefine the same record area as the first formlibrary.
SAME RECORD AREA clause	Enables all form descriptions in the formlibrary to be invoked as redefinitions of the first form in the formlibrary.
WRITE FORM statement	Writes forms from a station to a program.

SDF Plus Extensions

The following extensions have been developed for the SDF Plus interface. Detailed information about the extensions is provided in Section 7, "Using the SDF Plus Program Interface."

Extension	Explanation
DICTIONARY statement	Identifies the dictionary to be used during compilation.
Form record number attribute	Provides a means of performing I/O operations on form record libraries to enable individual form records to be specified at run time.
FROM DICTIONARY clause	Invokes an SDF Plus form record library from the dictionary.
GLOBAL clause	Allow references in subprograms to form record libraries declared in host programs.
READ FORM statement	Causes a form record to be read from a specified remote file and stored in a specified buffer.
REDEFINES clause	Enables multiple form record libraries to have the same record area.
SAME RECORD AREA clause	Enables all form record descriptions in the form record library to be invoked as redefinitions of the first form record description in the form record library.
SEPARATE RECORD AREA clause	Invokes each form record in the form record library as a separate data description with its own record area.
Transaction number attribute	Provides a means of performing I/O operations on form record libraries to enable individual transactions to be specified at run time.
WRITE FORM statement	Writes the contents of a form record to a specified remote file.
WRITE FORM TEXT statement	Causes the contents of a text array to be written to a remote file.

SIM Extensions

The following extensions have been developed for the SIM interface. Detailed information on the extensions is provided in Section 8, "Using the SIM Program Interface."

Extension	Explanation
ABORT-TRANSACTION statement	Causes a program to revert to the point in time just before the BEGIN-TRANSACTION statement was executed.
APPLY INSERT statement	Applies to the database all multiple-statement update INSERT statements between the START INSERT and APPLY INSERT statements.
APPLY MODIFY statement	Applies to the database all multiple-statement update MODIFY statements between the START MODIFY and APPLY MODIFY statements.
ASSIGN clause and ASSIGN statement	Updates single-valued database attributes.
BEGIN-TRANSACTION statement	Places a program in transaction state.
CALL SYSTEM statement	Obtains text that describes the current exception, and gives information about multiple exceptions. DMEXCEPTIONMSG, DMEXCEPTIONINFO, and DMNEXTEXCEPTION are the clauses used with this statement.
CANCEL TRANSACTION POINT statement	Discards all updates in a transaction to an intermediate transaction point or to the beginning of the transaction.
CLOSE statement	Closes a database when further access is not required.
Compound clause	Assigns or adds values to compound attributes. This clause is used with the ASSIGN or INCLUDE statement.
Database declaration	Specifies the name and type of database that is used in a query.
DELETE statement	Causes the entities in the specified class that meet the selection requirements to be deleted from the database.
DISCARD statement	Discards, or terminates, a currently active query.
DMERROR Use procedure	Handles exceptions.
DMSTATE statement	Indicates information about exceptions. This database status word contains a value associated with each program that accesses a database.
END-TRANSACTION statement	Takes a program out of transaction state.
Entity reference variable declaration	Declares an entity reference variable in a USAGE clause. This variable, which contains an explicit reference to a database entity, can be used to compare and assign entity-valued attribute values (EVAs) without selecting and retrieving the entities involved.
EXCLUDE clause and EXCLUDE statement	Removes values from either single-valued or multivalued attributes.

continued

continued

Extension	Explanation
Expressions	Include data management (DM) expressions, which determine values or entities, and selection expressions. A selection expression is a type of DM expression that identifies a set of entities upon which a query is to operate. Selection expressions can be either global or local.
Functions	Perform various activities. These functions include aggregate, time and date, symbolic, string, arithmetic, and special constructs.
INCLUDE clause and INCLUDE statement	Adds values to multivalued attributes (MVAs). This attribute assignment is in the form of a statement when used in a multiple-statement update.
INSERT statement	Creates an entity in the specified class by using the accompanying attribute assignments.
MODIFY statement	Updates existing entities in the specified class by using the accompanying attribute assignments.
ON EXCEPTION option	Handles exceptions.
OPEN statement	Opens a database for access and specifies the access mode.
Query declaration	Specifies the name of the query and the classes or items used in the query.
RESERVE clause	Notifies the compiler that SIM keywords are handled as reserved words for the program.
RETRIEVE statement	Retrieves the query and makes the entities available to the program.
SAVE TRANSACTION POINT statement	Provides an intermediate transaction point.
SELECT statement	Selects a set of entities from a database and associates the set with the query.
SET statement	Alters the level value in a retrieval involving transitive closure.
START INSERT statement	Describes the type of selection, if any, and associates a query name with the statement. This statement precedes a multiple-statement update insertion.
START MODIFY statement	Describes the selection and associates a query name with the statement. This statement precedes a multiple-statement update to the database.

Section 2
Using the ADDS Program Interface

The Advanced Data Dictionary System (ADDS) is a centralized storage system used to create and maintain data descriptions. ADDS is the Unisys repository for the A Series systems. With ADDS, you can manipulate data, define complex data structures, and update and report on entities or structures in the data dictionary.

The program interface for ADDS enables you to invoke entities such as files, records, record collections, and databases. It also provides the following options:

- Using the DICTIONARY compiler control option to include only entities with a particular status in the dictionary in your program
- Using the INVOKE clause to assign alias identifiers to file and data names to be used in the program
- Using the DICTIONARY statement with the PROGRAM-NAME option to allow tracking to occur where ADDS has already specified program tracking
- Allowing entity tracking to occur where ADDS has already specified both entity and program tracking

You can use ADDS to define Data Management System II (DMSII) databases. If you are running ADDS in the InfoExec environment with Semantic Information Manager (SIM), you can also use ADDS to define SIM databases.

For information on DMSII, see Section 4, "Using the DMSII Program Interface." For information on SIM, refer to Section 8, "Using the SIM Program Interface." For information on using the ADDS product, refer to the *InfoExec ADDS Operations Guide.*

If you have created formlibraries using the Screen Design Facility (SDF) and stored them in an ADDS dictionary, you can access these formlibraries just as you would other entities. Using SDF with ADDS provides some additional capabilities: form and formlibrary creation, automatic form painting using entities previously defined in ADDS, use of the ADDS prefixing and synonym features, and entity tracking for programs. For more information on SDF, refer to Section 6, "Using the SDF Program Interface." For information on ADDS prefixing, synonym, and entity tracking features, refer to the *InfoExec ADDS Operations Guide.*

If you have created form record libraries using the Screen Design Facility Plus (SDF Plus) and stored them in an ADDS dictionary, you can access these form record libraries just as you would other entities. For more information, refer to Section 7, "Using the SDF Plus Program Interface."

The information on the following pages explains how to write a program using the extensions developed for ADDS. Each extension is covered individually, with a description of its purpose or use, the syntax, an explanation, and an example. Two

sample programs are shown at the end of this section. For an alphabetized list of the extensions, refer to Section 1, "Introduction to COBOL74 Program Interfaces."

Accessing Entities with a Specific Status

The DICTIONARY compiler control option causes the COBOL74 compiler to access entities with a specified status. The use of this compiler control option is optional. Additional information on compiler control options is provided in Volume 1.

You can define a program entity in ADDS and use the DICTIONARY compiler control syntax in your program to restrict invocation of entities to those with a particular status.

Entities with historical status cannot be invoked by the COBOL74 compiler. You can invoke entities with historical status if you first change their status. Refer to the *InfoExec ADDS Operations Guide* for information on status and for the rules ADDS uses to search for an entity.

General Format

The general format for the DICTIONARY compiler control option is as follows:

```
DICTIONARY = { PRODUCTION }
             { TEST       }
```

Explanation of Format Elements

PRODUCTION

This format element ensures that only PRODUCTION status data dictionary entities are invoked. There is no default value.

TEST

This format element ensures that only TEST status data dictionary entities are invoked. There is no default value.

Example

The following example shows the DICTIONARY compiler control option used to invoke entities that are defined in ADDS with a PRODUCTION status:

```
$SET DICTIONARY = PRODUCTION
```

Identifying Specific Entities

The VERSION and DIRECTORY clauses uniquely identify an entity in the data dictionary. These clauses are assigned to an entity in an ADDS session. Refer to the *InfoExec ADDS Operations Guide* for information on entity default rules.

The VERSION and DIRECTORY clauses identify the following:

- A tracking program by using the PROGRAM-VERSION and PROGRAM-DIRECTORY clauses of the DICTIONARY statement
- A particular file in the dictionary when the clause is used in the SELECT statement
- A particular lower-level entity (such as a data item or record description) in the dictionary by using the data description FROM DICTIONARY clause

Although the VERSION and DIRECTORY clauses are optional, they provide additional information for identifying a particular entity. This is especially helpful if there are many duplicate items under different directories in the dictionary. If the VERSION and DIRECTORY clauses are not specified, ADDS follows its default rules to determine which entity to retrieve. See the *InfoExec ADDS Operations Guide* for information on ADDS default rules.

VERSION Clause

ADDS creates a unique version number for every entity that you define. The VERSION clause identifies the version number of the record description.

The general format for the VERSION clause is as follows:

```
[VERSION is literal-1]
```

Explanation of Format Element

literal-1

The version number is a numeric literal that can be up to 6 digits in length and can contain any value from 1 to 999999. Literal-1 must be a valid version number of the entity in the data dictionary.

Example

The following example specifies *2* as the version:

```
Ø1 SAMPLELIB FROM DICTIONARY
     VERSION IS 2
     DIRECTORY IS "JONES".
```

DIRECTORY Clause

The DIRECTORY clause specifies the directory under which the entity is stored in the data dictionary. Refer to the *InfoExec ADDS Operations Guide* for detailed information about directories.

The DIRECTORY syntax replaces the former USER clause. The USER syntax is still supported; however, Unisys recommends that you use the DIRECTORY syntax in your programs.

General Format

The general format of the DIRECTORY clause is as follows:

```
[DIRECTORY is literal-1]
```

Explanation of Format Element

literal-1

This literal describes the directory under which a data description is stored in the dictionary specified in the SPECIAL-NAMES paragraph. Literal-1 must be a nonnumeric literal of 1 to 17 alphabetic characters.

Example

In the following example, the version is *1* and the directory is specified as *Smith*:

```
SELECT ADDS-FILE FROM DICTIONARY
   VERSION IS 1
   DIRECTORY IS "SMITH".
```

Assigning Alias Identifiers

The optional INVOKE clause assigns an alias identifier to an entity name invoked from the dictionary. You then refer to the entity by its alias identifier throughout the rest of your program.

Assigning an alias is useful when, for example, you want to invoke the same record twice in your program. The alias enables you to use a unique qualifier.

In the program, the INVOKE clause is in the DATA DIVISION. Use the INVOKE clause to assign an alias as follows:

- In the SELECT statement, assigns an alias to a file
- In the data-description entry FROM DICTIONARY clause, assigns an alias to a lower-level entity such as a record or data item

See Also

See "Selecting a File" and "Invoking File Descriptions" later in this section for more information.

General Format

The general format of the INVOKE clause is as follows:

```
[data-name-1 {INVOKE}]
             {=     }
```

Explanation of Format Element

data-name-1

This data name must be at the 01-level. Once an alias is assigned, any reference to the data name in the program must specify data-name-1. All PROCEDURE DIVISION statements must also use this alias.

Example

In the following example, MY-INTERNAL-NAME is an alias identifier for the ADDS-ENTITY-NAME identifier:

```
Ø1  MY-INTERNAL-NAME INVOKE ADDS-ENTITY-NAME
          FROM DICTIONARY.
```

Identifying a Dictionary

You can use the DICTIONARY statement to identify the dictionary to be used during compilation. The dictionary is identified in the SPECIAL-NAMES paragraph of the program. Optional program clauses provide program tracking. If no dictionary is specified using the DICTIONARY statement, the compiler uses DATADICTIONARY by default.

Note: *A program can invoke only one dictionary. Therefore, if a program accesses both a SIM database (from a dictionary) and SDF forms, both must be in the same dictionary.*

General Format

The general format of the DICTIONARY statement is as follows:

```
[, DICTIONARY IS literal-1
 [, PROGRAM-NAME IS literal-2
 , PROGRAM-VERSION IS literal-3
 , PROGRAM-DIRECTORY IS literal-4] ]
```

Explanation of Format Elements

DICTIONARY

This statement specifies the function name of the dictionary library. The function name is the name equated to a library code file using the SL (support library) system command. The function name can be no longer than 17 characters.

literal-1

This format element designates the function name of the dictionary library. The function name can be 17 characters in length and optionally terminated by a period.

PROGRAM-NAME

This option specifies the name of the program entity defined in ADDS. The PROGRAM-NAME option is required if you desire tracking information to be sent to ADDS. Tracking is an ADDS feature. By defining a program entity in ADDS and setting the tracking attribute to YES, you can direct the dictionary to make a list (track) of the data structures and entities that are invoked in the DATA DIVISION and that are explicitly referenced in the PROCEDURE DIVISION of your program. ADDS will then associate this list with the program entity. More information on program tracking is provided in the *InfoExec ADDS Operations Guide*.

If the PROGRAM-NAME option appears but tracking is not set in ADDS, a warning is generated only if the WARNSUPR option is reset when the PROGRAM-NAME option appears.

If the PROGRAM-NAME option appears and tracking is set in ADDS, tracking information is sent to ADDS only if the program compiles correctly. Tracking information is not sent if syntax errors are encountered or if compilation is aborted.

literal-2

This format element designates a valid program name in the data dictionary.

PROGRAM-VERSION

This option specifies to which version of the program entity tracking information is sent.

literal-3

This format element designates a numeric literal that contains from 1 to 6 digits.

PROGRAM-DIRECTORY

This option specifies the directory of the program to be tracked.

literal-4

This format element designates a valid data dictionary directory.

See Also

See "Identifying Specific Entities" earlier in this section for information on the VERSION and DIRECTORY clauses.

Example

The following example shows DATADICTIONARY specified as the function name of the dictionary library. EXAMPLE-PROGRAM is the name of the program entity in ADDS to which tracking information is sent.

```
SPECIAL-NAMES.
  DICTIONARY IS "DATADICTIONARY"
      PROGRAM-NAME IS "EXAMPLE-PROGRAM"
      PROGRAM-VERSION IS 1
      PROGRAM-DIRECTORY IS "JOHNDOE".
```

Selecting a File

The SELECT statement includes files from a dictionary in your program. You can assign an alias identifier to a selected file. You can also use the VERSION and DIRECTORY clauses to identify the particular file. In your program, you include the SELECT statement in the FILE-CONTROL paragraph of the INPUT-OUTPUT SECTION.

General Format

The general format for the SELECT statement is as follows:

```
SELECT [file-name-1 {INVOKE} ] file-name-2 FROM DICTIONARY
       [           {=     } ]
[, VERSION IS literal-1] [, DIRECTORY IS literal-2] .
```

Explanation of Format Elements

file-name-1

This name designates the alias for the file selected from a dictionary.

INVOKE

This option specifies an alias for file-name-2.

file-name-2

This name designates the file selected from the dictionary. Subsequent references to this file name must use the file-name-1 alias.

FROM DICTIONARY

This clause obtains an entity from the dictionary.

VERSION IS literal-1

This format element specifies a numeric literal that identifies a version of the file. Literal-1 comprises from 1 to 6 numeric digits.

DIRECTORY IS literal-2

This format element specifies a nonnumeric literal that designates the directory under which the file is stored in the data dictionary.

Considerations for Use

If you run SDF with ADDS, you can use the SELECT statement to select a remote file from the dictionary by following these instructions:

1. Define the file in ADDS, and relate the formlibrary to the file. For information on defining files in ADDS, refer to the *InfoExec ADDS Operations Guide*.
2. Use the SELECT statement with the FROM DICTIONARY clause, and invoke the formlibrary in the file description statement of the program.

See Also

- For more information on using SDF with ADDS, refer to the *SDF Operations and Programming Guide* and to Example 2–2, "ADDS Sample Program with SDF," later in this section.
- Refer to Section 6, "Using the SDF Program Interface," for a discussion of using SDF extensions in your COBOL74 program.
- For information on the VERSION and DIRECTORY clauses, see "Identifying Specific Entities" earlier in this section.
- See "Assigning Alias Identifiers" earlier in this section for information on assigning an alias identifier to a selected file.

Example

The following example of the SELECT statement contains comments that explain the selection of a remote file:

```
 INPUT-OUTPUT SECTION.
 FILE-CONTROL.
    SELECT ADDS-FILE FROM DICTIONARY
        VERSION IS 1
        DIRECTORY  IS "*".
*--DICTIONARY DIRECTORY : *.
*--     ASSIGN TO DISK
*--     RESERVE 2 AREAS
*--     ORGANIZATION IS SEQUENTIAL
*--     .
    SELECT REMOTE-FILE FROM DICTIONARY.
*--DICTIONARY DIRECTORY : *.
*--     ASSIGN TO REMOTE
*--     .
 DATA DIVISION.
 FILE SECTION.
```

Invoking File Descriptions

The file description entries provide information about the physical structure, identification, and record names pertaining to a file. An FD entry identifies the beginning of a file description; an SD entry identifies the beginning of a sort-merge file description. The FD and SD entries are level indicators that invoke all file attributes named in the SELECT statement. Using the INVOKE ALL option, you can invoke the record descriptions as well as the file attributes.

File Description Entries

The file description entries are valid only for files previously selected using the SELECT statement. If you assigned an alias using the INVOKE option in the SELECT statement, then you must use that alias identifier for the file name.

If you do not use the INVOKE ALL option, one or more record description entries must follow the file description entry or sort-merge file description entry. No I/O statements (except RELEASE and RETURN) can be executed for the sort-merge file. Refer to Volume 1 for information on I/O statements.

General Formats

The formats for ADDS file descriptions are as follows:

Format 1

```
FD file-name [INVOKE ALL] .
```

Format 2

```
SD file-name [INVOKE ALL] .
```

Explanation of Format Elements

FD

This level indicator identifies the beginning of a file description and must precede the file name. FD stands for file description.

SD

This level indicator identifies the beginning of a file description and must precede the file name. SD refers to a sort-merge file description.

See Also

- For general information on the FILE SECTION and the file description entries in a program, refer to Volume 1.
- See "INVOKE ALL Option" later in this section for a description of that option.
- See "Assigning Alias Identifiers" earlier in this section for information on assigning an alias identifier to a selected file.
- See "Selecting a File" earlier in this section for information on the SELECT statement.

Examples

The following statements are examples of FD and SD statements:

```
FD REMOTE-FILE.

SD SORT-FILE     INVOKE ALL.
```

INVOKE ALL Option

If you used the SELECT statement to select a file, the INVOKE ALL option invokes form and record descriptions as well as file attributes associated with the file. The option appears in the file description in the DATA DIVISION of the program.

General Format

The general format of the INVOKE ALL option is as follows:

```
[INVOKE ALL]
```

Explanation of Format Element

INVOKE ALL

This option invokes all the record description, SDF form, and SDF Plus record descriptions defined for the file.

If the INVOKE ALL option is not specified in the file description, all file attributes are invoked; however, a selected set of record descriptions, SDF form or SDF Plus record descriptions can be invoked separately.

Example

The following code is an example of the INVOKE ALL option:

```
FILE SECTION.
FD ADDS-FILE    INVOKE ALL.
```

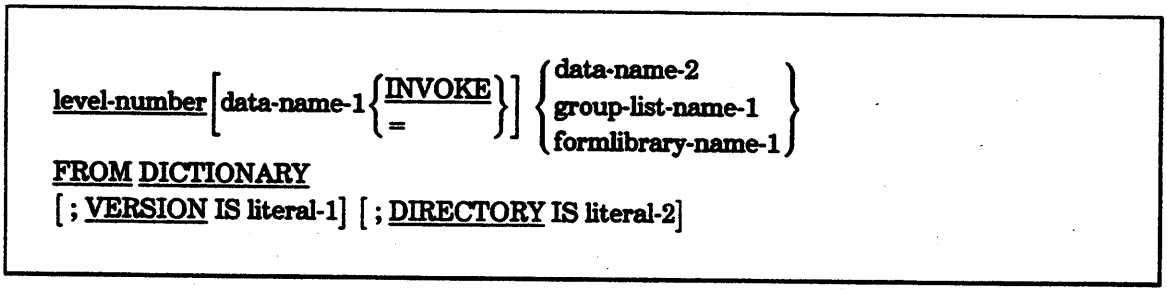
Invoking Data Descriptions

A data-description entry specifies the characteristics of a particular data item. You use the FROM DICTIONARY clause to obtain an entity from the dictionary. If you are using SDF with ADDS, you can also invoke formlibraries into your program. Section 6, "Using the SDF Program Interface," contains information on invoking formlibraries and using the optional REDEFINES and SAME RECORD AREA clauses.

The data-description entry is used within the FILE SECTION to invoke record descriptions, SDF Plus record descriptions, or SDF formlibraries for any file that has not been declared with an INVOKE ALL optional clause in its FD entry.

General Format

The general format for invoking data descriptions in ADDS is as follows:



Explanation of Format Elements

data-name-1

This data name must be at the 01-level. Once an alias is assigned, any reference to the data name in the program must specify data-name-1. All PROCEDURE DIVISION statements must also use this alias.

INVOKE

This option specifies an alias for data-name-2. INVOKE and the equal sign (=) are synonyms.

data-name-2

This format element identifies a dictionary structure that is a record description. The record description is at the 01-level and it is to be included in the FILE SECTION, WORKING-STORAGE SECTION, LINKAGE SECTION, or LOCAL-STORAGE SECTION.

group-list-name-1

This format element identifies a dictionary structure that is a collection of unrelated descriptions (at the 77-level or 01-level, or both), items, and records that are to be included in the WORKING-STORAGE SECTION, LINKAGE SECTION, or LOCAL-STORAGE SECTION. An alias cannot be specified for group-list-name-1.

formlibrary-name-1

This format element identifies a dictionary structure that is a collection of record descriptions describing forms. For information on formlibraries, which are created using Screen Design Facility (SDF), refer to the *SDF Operations and Programming Guide*. An alias cannot be specified for formlibrary-name-1.

VERSION IS literal-1

This format element specifies a numeric literal that identifies a version of the file. Literal-1 comprises from 1 to 6 digits.

DIRECTORY IS literal-2

This format element specifies a nonnumeric literal that identifies the directory under which the file is stored in the data dictionary.

Considerations for Use

When invoking data descriptions, the following rules apply:

- The 01-level records, SDF Plus form record libraries, or SDF formlibraries can be invoked within the FILE SECTION, the WORKING-STORAGE SECTION, the LINKAGE SECTION, or the LOCAL-STORAGE SECTION. Record collections, however, cannot be invoked within the FILE SECTION; they can be invoked only within the WORKING-STORAGE SECTION, the LINKAGE SECTION, or the LOCAL-STORAGE SECTION.
- Within the FILE SECTION, no data-description entry can follow a file description that includes the INVOKE ALL option under an FD entry.
- Within the FILE SECTION, if the file has been selected using the SELECT statement shown for ADDS, the record or the formlibrary must be associated with that file in the data dictionary.
- A record collection or formlibrary cannot be given an alias identifier using the INVOKE clause.

See Also

For information about the VERSION and DIRECTORY clauses, see "Identifying Specific Entities" earlier in this section.

Example

In the following example, an invocation with an SDF Plus record description is coded in the WORKING-STORAGE SECTION because record descriptions cannot be invoked in the FILE SECTION:

```
WORKING-STORAGE SECTION.
Ø1 MY-REC-LIST INVOKE
   ADDS-REC-LIST FROM DICTIONARY.
       VERSION IS 2
       DIRECTORY IS "*".
```

ADDS Programming Examples

Examples 2–1 and 2–2 are sample programs that illustrate the use of extensions in your COBOL74 program. The first program shows how to invoke entities; the second program shows how to run ADDS with the SDF interface.

Invoking Entities and Program Tracking

Example 2–1 shows how to invoke ADDS entities into a COBOL74 program.

The clauses that are used to invoke dictionary entities are accurately coded. However, this is only a program fragment and not a complete COBOL74 program. Note that a numbered line with an asterisk (*) in the first column of the excerpt code indicates file attributes that are generated from the ADDS definition.

```
000900 $$SET LIST MAP
001000 $SET DICTIONARY = TEST
001100 IDENTIFICATION DIVISION.

001200 PROGRAM-ID.     EXAMPLE-PROGRAM.
001300 ENVIRONMENT DIVISION.
001400 CONFIGURATION SECTION.
001500 SPECIAL-NAMES.

      ************************************************
      *                                              *
      *   This clause identifies the dictionary that *
      *   contains the data descriptions this program *
      *   invokes.  It also specifies in the         *
      *   dictionary the program to be tracked.      *
      *                                              *
      ************************************************

001600     DICTIONARY IS "DATADICTIONARY"
001700         PROGRAM-NAME IS "EXAMPLE-PROGRAM"
001800         PROGRAM-VERSION IS 1
001900         PROGRAM-DIRECTORY IS "EXAMPLE".
002000 INPUT-OUTPUT SECTION.
002100 FILE-CONTROL.
      ************************************************
      *                                              *
      *   Files can be invoked from the dictionary by *
      *   using the SELECT statement.                *
      *                                              *
      ************************************************

002200     SELECT ADDS-FILE FROM DICTIONARY
002300         VERSION IS 1
002400         DIRECTORY  IS "*".
```

Example 2–1. ADDS Sample Program to Invoke Entities

```
000100*--DICTIONARY DIRECTORY : *.
000110*--      ASSIGN TO DISK
000120*--      RESERVE 2 AREAS
000130*--      ORGANIZATION IS SEQUENTIAL
000140*--      .
002500     SELECT REMOTE-FILE FROM DICTIONARY.
000100*--DICTIONARY DIRECTORY : *.
000110*--      ASSIGN TO REMOTE
000120*--      .
002600 DATA DIVISION.
002700 FILE SECTION.
002800 FD ADDS-FILE    INVOKE ALL.
000100*--DICTIONARY DIRECTORY : *.
000110*--    BLOCK CONTAINS 0 TO 50 RECORDS
000120*--    LABEL RECORDS ARE STANDARD
000130*--    VALUE OF PROTECTION IS SAVE
000140*--    VALUE OF MYUSE IS IO
000150*--    VALUE OF NEWFILE IS TRUE
000160*--    VALUE OF TITLE IS
000170*--     "ADDS/FILE/061484 ON DISK"
000180*--      .
000110 01 ADDS-A.
000130    02 ADDS-B.
000140      03 ADDS-D PIC X(9).
000150      03 ADDS-E PIC X(7).
000170    02 ADDS-C.
000190      03 ADDS-F.
000210        04 ADDS-G.
000230          05 ADDS-H.
000240            06 ADDS-I PIC 9(4).
000250            06 ADDS-J PIC 9(4).
000260            06 ADDS-K PIC 9(1)V99 COMP.
000270            06 ADDS-L PIC S9(1)V99999 COMP.
000280            06 ADDS-M PIC X(6).
000290        04 ADDS-F1 PIC X(8).
000390      03 ADDS-N.
000400        04 ADDS-N1 PIC 9(10) BINARY.
000420      03 ADDS-O.
000430        04 ADDS-O1 PIC 9(4) BINARY.
000450      03 ADDS-P.
000460        04 ADDS-P1 PIC 9(3) BINARY.
000480 01 ADDS-REC-2.
000500    02 ADDS-A.
000520      03 ADDS-B.
000530        04 ADDS-D PIC X(9).
000540        04 ADDS-E PIC X(7).
000560      03 ADDS-C.
000580        04 ADDS-F.
000600          05 ADDS-G.
000620            06 ADDS-H.
000630              07 ADDS-I PIC 9(4).
```

Example 2–1. ADDS Sample Program to Invoke Entities (cont.)

```
000640            07 ADDS-J PIC 9(4).
000650            07 ADDS-K PIC 9(1)V99 COMP.
000660            07 ADDS-L PIC S9(1)V99999 COMP.
000670            07 ADDS-M PIC X(6).
000680          05 ADDS-F1 PIC X(8).
000780        04 ADDS-N.
000790          05 ADDS-N1 PIC 9(10) BINARY.
000810        04 ADDS-O.
000820          05 ADDS-O1 PIC 9(4) BINARY.
000840        04 ADDS-P.
000850          05 ADDS-P1 PIC 9(3) BINARY.
002900 FD REMOTE-FILE.
000100*--DICTIONARY DIRECTORY : *.
000110*--   BLOCK CONTAINS 2500 CHARACTERS
000120*--   RECORD CONTAINS 2500 CHARACTERS
000130*--   VALUE OF FILETYPE IS 3
000140*--   VALUE OF MYUSE IS IO
000150*--   CODE-SET IS EBCDIC
000160*--
003000 01 ADDS-FL   FROM DICTIONARY.
000110*--DICTIONARY FORMLIST<ADDS-FL>
000130 01 ADDS-FORM-1.
000140   02 ADDS-FF-1 PIC X(60).
000150   02 ADDS-FF-2 PIC 9(12) COMP.
003100 DATA-BASE SECTION.
003200 DB ADDSSAMPLEDB ALL.
       * ADDSSAMPLEDB ON DISK
       * ADDSSAMPLEDB TIMESTAMP = 02/26/85 @ 17:55:00
       *    02 ADDS-GBL-DATA-1                                  BOOL
       *    02 ADDS-GBL-DATA-2              PIC S9(4)V99        COMP
       * 01 ADDS-DS-1 STANDARD DATA SET(;3).
       *      ADDS-DS1-SET1 SET(;5,AUTO)
       *        OF ADDS-DS-1 KEY IS ADDS-
       *      ADDS-DS1-SET2 SET(;6,AUTO)
       *        OF ADDS-DS-1 KEY IS ADDS-
       *    02 ADDS-DS1-GRP1.
       *       03 ADDS-GRP1-A1             PIC X(10)           DISP
       *       03 ADDS-GRP1-R1                                 REAL
       *    02 ADDS-DS1-GRP2.
       *       03 ADDS-GRP2-A1             PIC X(10)           DISP
       *       03 ADDS-GRP2-R1                                 REAL
       *    02 ADDS-DS1-R1                                     REAL
       *    02 ADDS-DS1-A1                 PIC X(10)           DISP
       * 01 ADDS-DS-2 STANDARD DATA SET(;4).
       *      ADDS-DS2-SET1 SET(;7,AUTO)
       *        OF ADDS-DS-2 KEY IS ADDS-
       *    02 ADDS-DS2-R1                                     REAL
       *    02 ADDS-DS2-GRP1.
       *       03 ADDS-GRP1-A1             PIC X(10)           DISP
       *       03 ADDS-GRP1-R1                                 REAL
       *    02 ADDS-DS2-GRP2.
```

Example 2–1. ADDS Sample Program to Invoke Entities (cont.)

```
      *       03 ADDS-GRP2-A1              PIC X(10)       DISP
      *       03 ADDS-GRP2-R1                              REAL
      *    02 ADDS-DS2-A1                  PIC X(10)       DISP
003300 WORKING-STORAGE SECTION.
003400 01 MY-REC-LIST INVOKE
003450    ADDS-REC-LIST FROM DICTIONARY.
000100*--DICTIONARY DIRECTORY : *.
000110 77 ADDS-RL-1 PIC 9(11) BINARY.
000120 77 ADDS-RL-2 PIC 9(11) BINARY.
000140 01 ADDS-RL-GRP-1.
000150   02 ADDS-RLG-1 PIC S9(10)V99.
000160   02 ADDS-RLG-2 PIC S9(6)V99
000170      SIGN IS TRAILING SEPARATE CHARACTER.
000180   02 ADDS-RLG1-3 PIC $$$.99CR.
000200 01 ADDS-RL-GRP-2.
000210   02 ADDS-RLG2-1 PIC 9(12) COMP.
000220   02 ADDS-RLG2-2 PIC 9(12).
000240 01 ADDS-RL-GRP-3
000250    REDEFINES ADDS-RL-GRP-2.
000260   02 ADDS-RLG3-1 PIC 9(12).
000270   02 ADDS-RLG3-2 PIC 9(12) COMP.
000280 01 ADDS-ACTION-CODE PIC X(2).
000290    88 ADDS-VALID-CODES
000300       VALUE "I" ,   "C" ,   "M" ,   "D".
000310 01 ADDS-ALPHA-CHECK PIC X(1).
000320    88 ADDS-ALPHA-CHARS
000330       VALUE "A" THRU "Z".
000340 01 ADDS-NUMBER-CHECK PIC 9(2) COMP.
000350    88 ADDS-NUMBERS
000360       VALUE 1 THRU 9.
000380 01 ADDS-RL-GRP-4.
000390   02 ADDS-RLG4-2 PIC X(20).
000400   02 FILLER PIC X(60).
000410*--DICTIONARY FILLER NAME : ADDS-FILLER
000430   02 ADDS-RL-GRP-5
000440      OCCURS 1 TO 5 TIMES
000450      DEPENDING ON ADDS-RL-1.
000460     03 ADDS-RLG5-1 PIC 9(6) COMP.
003500 PROCEDURE DIVISION.
003600 MAIN-BLOCK.
003700      PERFORM INITIALIZE.
003800      PERFORM MAIN-ROUTINE.
003900      PERFORM EOJ.
004000      STOP RUN.
004100 INITIALIZE.
004200      OPEN OUTPUT ADDS-FILE.
004300      MOVE SPACES TO ADDS-REC-2.
004500      WRITE ADDS-REC-2.
004600 MAIN-ROUTINE.
004650      MOVE "ENTER YOUR DATA" TO ADDS-FF-1.
004700      WRITE FORM ADDS-FORM-1
```

Example 2–1. ADDS Sample Program to Invoke Entities (cont.)

```
004725          ON ERROR STOP RUN.
004800       READ FORM REMOTE-FILE USING ADDS-FORM-1
004825          ON ERROR STOP RUN.
004850       IF ADDS-FF-1 = "YOUR DATA"
004900          WRITE ADDS-REC-2.
005000 EOJ.
005100       CLOSE ADDS-FILE.
 ************************************************************************
 *                                                                      *
 * The information shown below is printed at the end of each listing *
 * and documents various statistics.                                    *
 * Note the DATADICTIONARY Program Tracking Summary.  This is only   *
 * printed if tracking is established for your program in the         *
 * dictionary.  The summary displays the program name, directory,     *
 * and version under which the program is stored in the dictionary.   *
 *                                                                      *
 ************************************************************************

COMPILE O.K.
TOTAL CARD COUNT: 157
D[01] STACK SIZE: 0023(017) WORDS
D[02] STACK SIZE: 0053(035) WORDS
CORE ESTIMATE:    2702 WORDS
STACK ESTIMATE:   310 WORDS
CODE FILE SIZE:   21 RECORDS
PROGRAM SIZE:     3 CODE SEGMENTS, 225 TOTAL WORDS
SUBROUTINE NAME:  CODE3780, LEVEL 02
COMPILED ON THE MICRO A  FOR THE A SERIES
COMPILER COMPILED WITH THE FOLLOWING OPTIONS:
    BDMS.
COMPILE TIMES:    ELAPSED    CPU      I-O    RPM
                 0042.645 0002.074 0002.112 04541
DATADICTIONARY PROGRAM TRACKING SUMMARY :
      PROGRAM NAME : EXAMPLE-PROGRAM , DIRECTORY : EXAMPLE ,
      VERSION : 000001 ,
      STATUS : TEST
```

Example 2–1. ADDS Sample Program to Invoke Entities (cont.)

Using ADDS with SDF

Example 2–2 shows an SDF program run with ADDS. The program uses a remote file and specific forms.

Refer to the *SDF Operations and Programming Guide* for more information on using SDF with ADDS. Refer to Section 6, "Using the SDF Program Interface," for a discussion of using SDF extensions in your COBOL74 program.

```
ØØ1ØØØ IDENTIFICATION DIVISION.
ØØ7ØØØ
ØØ8ØØØ ENVIRONMENT DIVISION.
ØØ9ØØØ CONFIGURATION SECTION.

       ********************************************************
       * The following lines specify the dictionary that     *
       * stores the formlibrary.                              *
       * This example program also indicates the version     *
       * and directory of the program entity.                 *
       ********************************************************
Ø11ØØØ
Ø11200 SPECIAL-NAMES.
Ø113ØØ   DICTIONARY IS "DATADICTIONARY"
Ø114ØØ   PROGRAM-NAME IS "SAMPLE2"
Ø11425   PROGRAM-VERSION IS 1
Ø1145Ø   PROGRAM-DIRECTORY IS "USER1".

       ********************************************************
       *   The following lines declare the remote file.      *
       *                                                      *
       ********************************************************
Ø115ØØ
Ø12ØØØ INPUT-OUTPUT SECTION.
Ø13ØØØ FILE-CONTROL.
Ø14ØØØ     SELECT REMFILE
Ø145ØØ       ASSIGN TO REMOTE.

       ********************************************************
       *  The lines following the FD statement illustrate    *
       *  file attributes that ensure a correct record size  *
       *  for full screen writes when using remote files.    *
       *                                                      *
       ********************************************************

Ø15ØØØ
Ø16ØØØ DATA DIVISION.
Ø17ØØØ FILE SECTION.
Ø18ØØØ FD REMFILE.
ØØØ1ØØ*--DICTIONARY DIRECTORY : *.
ØØØ2ØØ*--   BLOCK CONTAINS 25ØØ CHARACTERS
ØØØ3ØØ*--   RECORD CONTAINS 25ØØ CHARACTERS
```

Example 2–2. ADDS Sample Program with SDF

```
000400*--   VALUE OF FILETYPE IS 3
000500*--   VALUE OF MYUSE IS IO
000600*--   CODE-SET IS EBCDIC
000700*--
      ********************************************************
      *  The following lines invoke the formlibrary          *
      *  in the FILE SECTION.                                *
      *  This example program also indicates the version     *
      *  and directory.                                      *
      *                                                      *
      ********************************************************

021200 01 SAMPLELIB  FROM DICTIONARY
021300   VERSION IS 2
021350   DIRECTORY  IS "USER1".
021400
      ********************************************************
      *                                                      *
      *  The following lines indicate where                  *
      *  the system automatically invokes and                *
      *  copies these program record descriptions            *
      *  for all forms in the formlibrary                    *
      *  into the program during compilation.                *
      *                                                      *
      ********************************************************

000100*--DICTIONARY                                          D
000110*--DICTIONARY FORMLIST <SAMPLELIB>.                    D
000130 01 SAMPLEFORM1.                                       D
000140   02 ACTION PIC X(10).                                D
000150   02 ACCOUNT-NO PIC 9(9).                             D
000160   02 NAME PIC X(15).                                  D
000170   02 STREET PIC X(25).                                D
000180   02 CITY PIC X(15).                                  D
000190   02 STATE PIC X(2).                                  D
000200   02 ZIP PIC 9(9).                                    D
024000 PROCEDURE DIVISION.
024500 MAIN-PARA.
025000
      ********************************************************
      *                                                      *
      * The following line opens the remote file.           *
      *                                                      *
      ********************************************************

026000  OPEN I-O REMFILE.
026100
```

Example 2–2. ADDS Sample Program with SDF (cont.)

```
      ******************************************************
      *    The following lines  write a form from the      *
      *    formlibrary to the terminal with default        *
      *    values; the program then waits for a            *
      *    transmission  to do a read operation.  WRITE    *
      *    statements  are not necessary when you are      *
      *    sending a form with default values to the       *
      *    terminal.                                       *
      ******************************************************

032000  READ FORM REMFILE USING SAMPLEFORM1 FROM DEFAULT FORM
032100     ON ERROR STOP RUN.
038000  STOP RUN.
```

Example 2–2. ADDS Sample Program with SDF (cont.)

Section 3
Using the COMS Program Interface

The Communications Management System (COMS) is a general message control system (MCS) that controls online environments on A Series systems. COMS supports a network of users and handles a high volume of transactions from programs, stations, and remote files.

The program interface for COMS enables you to create online, interactive, and batch programs that take advantage of the features which COMS offers for transaction processing through direct windows. The program interface enables your program to communicate with COMS, using the following COMS functions:

- Message routing by transaction codes (trancodes) and agendas
- Processing of message data through processing items
- Security checking of messages
- Service functions
- Dynamic opening of direct windows to terminals not attached to COMS, and dynamic communication by modem
- Synchronized recovery

COMS can be used with the Advanced Data Dictionary System (ADDS), Data Management System II (DMSII), Screen Design Facility (SDF), Screen Design Facility Plus (SDF Plus), and Semantic Information Manager (SIM). For more information on the extensions developed for these products, see the section that provides an explanation of each product interface.

This section explains how to use the COMS extensions to communicate with COMS through direct windows. For an alphabetized list of the extensions developed for COMS, refer to "COMS Extensions" in Section 1, "Introduction to COBOL74 Program Interfaces."

The tasks presented in this section include

- Preparing a communication structure for routing or for describing information about the message.
- Declaring and using COMS headers when receiving and sending messages.
- Attaching to and detaching from stations dynamically.
- Preparing to receive and send messages. This preparation includes linking to COMS and initializing a program.
- Using communication statements to receive and send messages.
- Using service functions.

Note that processing items can be written using COBOL74 with an ALGOL shell. Instructions on using the shell, as well as general COMS programming concepts, are provided in the *A Series Communications Management System (COMS) Programming Guide*. For information about using DMSII and SIM databases together, see the *A Series InfoExec Semantic Information Manager (SIM) Programming Guide*.

When DMSII is run with COMS, the following DMSII statements are used to enable a program interfacing with COMS to support synchronized transactions and recovery:

- ABORT-TRANSACTION
- BEGIN-TRANSACTION
- CLOSE
- DMTERMINATE
- END-TRANSACTION

COMS can also use the DMSII TPS MID-TRANSACTION statement.

When COMS is run with SIM, the following statements are used to support transactions and recovery:

- ABORT-TRANSACTION
- BEGIN-TRANSACTION
- CLOSE
- END-TRANSACTION

See Also

- Information about the syntax and use of DMSII statements is provided in Section 4, "Using the DMSII Program Interface."
- Information about the use of the DMSII TPS *MID-TRANSACTION* statement is provided in Section 5, "Using the DMSII TPS Program Interface."
- Information about the syntax and use of SIM statements is provided in Section 8, "Using the SIM Program Interface."
- Information on syntax used with SDF is provided in Section 6, "Using the SDF Program Interface."

Preparing the Communication Structure

The program must provide a communication structure for routing or for describing information about the message. To provide a communication structure, you must do the following:

- Declare an area for the message
- Declare a COMS interface that directs input and output and that provides an optional conversation area for user-defined information

Declaring a Message Area

To receive and send messages, you must declare a message area in the DATA DIVISION of the program. Always declare the message area as an 01-level record.

Declare the message area with a format and size that are appropriate to the data your program is to receive. If the message area is too small to contain all the incoming text, COMS truncates the message.

After a message is received, the Text Length field in the header contains the number of characters in the entire message text.

Example

The following example shows the declaration for a COMS message area. The declaration occurs in the DATA DIVISION.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. COMSMSG.
DATA DIVISION.
WORKING-STORAGE SECTION.
Ø1 COMS-MESSAGE-AREA.
    Ø2 COMS-MESSAGE          PIC X(25ØØ).
COMMUNICATION SECTION.
```

When a program performs updates with SDF forms, the message area can receive the SDF form. For information on how to write an application that uses SDF forms with COMS, see Section 6, "Using the SDF Program Interface," and the sample programs in the *COMS Programming Guide*.

Declaring a COMS Interface

You declare a COMS interface by using COMS headers for input or output. COMS headers are dynamically built at compilation. The compiler requests the header structure from COMS and constructs the headers.

COMS headers offer the following advantages:

- You do not need to know the memory structure because COMS handles the location of fields in the headers.
- COMS identifiers access all fields within the header; therefore, there is no need to rename the fields in the queue structure.
- You need not change your program when new releases of COMS occur.
- COMS headers can be referenced by bound-in procedures.

Using COMS Headers

There are two types of COMS headers: input headers used to receive messages and output headers used to send messages. The fields in each of the input and output headers can be used to receive or send values that provide information or instruction for various activities.

This discussion of using COMS headers begins with a description of the declaration for COMS headers and an explanation of how the COMS data types map into COBOL74. The rest of the discussion covers the use of the header fields.

For detailed information on the use of the headers and fields, refer to the *COMS Programming Guide*.

Declaring COMS Headers

Input and output headers are declared in the COMMUNICATION SECTION of the COBOL74 program. The following explanation contains the COBOL74 syntax, rules, and steps for declaring COMS headers in your program. Examples are also provided.

General Format

The general format for declaring input and output COMS headers is as follows:

```
{INPUT  }
{OUTPUT } HEADER data-name-1
[data-name-2 IS data-name-3] ...
[CONVERSATION AREA [IS data-name-4 SIZE IS integer-1]] .
[record-description-entry] .
```

Explanation of Format Elements

INPUT or OUTPUT HEADER

This phrase specifies COMS header declarations.

data-name-1

This format element names the input or output header. The header description is invoked and provides access to all the fields of the input header.

data-name-2

This format element is any identifier retrieved from COMS. If data-name-2 is not unique to the input header, you can access it with an OF qualification that mentions data-name-1.

IS

This option renames the identifier.

data-name-3

This option replaces the field name supplied by COMS; it is not equated. However, a field cannot be renamed with a name that already exists in the header. If data-name-2 is renamed, it must be referred to by data-name-3.

CONVERSATION AREA

This option provides a user-defined area associated with the input or output header. This option must include either data-name-4 and a SIZE phrase, or be mapped with the record-description-entry.

data-name-4

This format element names the conversation area. The area is specified as *02 data-name-4 PIC X(integer-1).*

SIZE

This phrase defines the conversation area as a PIC X representation with the length indicated by integer-1.

record-description-entry

This format element is added to the end of the header. The record-description-entry must start at level 02.

See Also

- For more information on using the OF qualification, see the discussion of using set and data set names in Section 4, "Using the DMSII Program Interface."
- For more information on the CONVERSATION AREA option and the fields of the input and output headers see "Using COMS Input Header Fields" and "Using COMS Output Header Fields" later in this section.
- For more information on a specific requirement for SDF, see Section 6, "Using the SDF Program Interface."

Example

The following example shows the declarations for COMS input and output headers. An example of declarations for headers within the context of a complete program is provided in Example 3–1, "COMS Sample Program with a DMSII Database."

```
COMMUNICATION SECTION.
INPUT HEADER COMS-IN
   PROGRAMDESG IS COMS-IN-PROGRAM.
   CONVERSATION AREA.
     Ø2 CA.
         Ø5 CA-1 PIC X(2Ø).
         Ø5 CA-2 PIC X(3Ø).
OUTPUT HEADER COMS-OUT.
```

Mapping COMS Data Types to COBOL74

Table 3–1 shows the COMS data types and the valid COBOL74 usage. For information on COMS types and COBOL74 usage for the fields of the COMS headers, see "Using COMS Input Header Fields" and "Using COMS Output Header Fields" later in this section.

Table 3–1. COMS Data Types and COBOL74 Usage

COMS Type	COBOL74 Usage
Boolean	DMS Boolean
Designator	Real
Display	Display
Mnemonic	Binary
Record	Display
TIME(6)	Real

COMS data types include a COMS designator data type that is used only for specific fields and with service functions. COMS determines the kind of designator and required name from the value that is passed. The designator type is compatible with COBOL74 data items of type real. More information about COMS designators used with service functions is provided in "Using COMS Designators" later in this section.

Boolean items are similar to DMS Boolean items; they can be tested with an IF condition and set with a COMPUTE statement.

Note that when using logical tests against the COMS type TIME(6), you must redefine the type as a PIC X(6) DISPLAY item that you test against.

Using COMS Input Header Fields

The fields of the input header are COBOL74 attributes of the header. The conversation area is not part of the header provided by COMS; it is an optional user-defined field that is associated with the input header. All other fields of the input header are defined by either COMS or COBOL74. The structure of COMS input headers is obtained from COMS at compilation.

COMS places values (designators and integers) in the input header fields when a RECEIVE, ACCEPT, DISABLE, or ENABLE statement is executed.

You use a service function to translate a designator to a name representing a COMS entity. See "Using Service Functions" later in this section for more information.

COMS uses input headers for the following tasks:

- Confirming message status
- Passing data in the Conversation Area field
- Detecting queued messages
- Determining message origin
- Processing transaction codes (trancodes) for routing
- Obtaining direct-window notifications

See Also

- For more information about input headers and fields, refer to the *COMS Programming Guide*.
- For information about data types, see "Mapping COMS Data Types into COBOL74" and "Using COMS Designators" in this section.

Table 3–2 lists the fields of the input header, showing the COMS and COBOL74 field names, the COMS data types, and the COBOL74 usages. The fields are listed alphabetically. An example of coding the input header fields is provided in Example 3–1, "COMS Sample Program with a DMSII Database," (lines 017002 through 017062) at the end of this section.

Table 3–2. Input Header Fields

COMS Field Name	COBOL74 Field Name	COMS Data Type	COBOL74 Usage
Agenda Designator	AGENDA	Designator	Real
Function Index	FUNCTIONINDEX	Mnemonic	Binary
Function Status	FUNCTIONSTATUS	Mnemonic	Binary

continued

Table 3–2. Input Header Fields (cont.)

COMS Field Name	COBOL74 Field Name	COMS Data Type	COBOL74 Usage
Message Count	MESSAGECOUNT	Integer	Binary
Program Designator	PROGRAMDESG	Designator	Real
Restart	RESTART	Designator	Real
Security Designator	SECURITYDESG	Designator	Real
Station Designator	STATION	Designator	Real
Status Value	STATUSVALUE	Mnemonic	Binary
Text Length	TEXTLENGTH	Integer	Binary
Timestamp	TIMESTAMP	TIME(6)	Real
Transparent	TRANSPARENT	Boolean	DMS Boolean
Usercode Designator	USERCODE	Designator	Real
User-Defined Conversation Area	User defined	User defined	User defined
VT Flag	VTFLAG	Boolean	DMS Boolean

Using COMS Output Header Fields

The fields of the output header are COBOL74 attributes of the header. The conversation area is not a field in the header provided by COMS; it is an optional user-defined field that is associated with the output header. All other fields in the header are defined by either COMS or COBOL74. The structure of COMS output headers is obtained from COMS during compilation.

COMS uses the output header when sending messages. You place designators into the output header fields to route outgoing messages and describe their characteristics. You can obtain designators by calling service functions to translate names representing COMS entities to designators.

COMS uses the output header fields for the following tasks:

- Specifying a destination
- Routing by transaction code (trancode)
- Sending messages using direct windows
- Confirming message delivery
- Checking the status of output messages

See Also

- For general information on output headers and fields, refer to the *COMS Programming Guide.*
- For information on data types, refer to "Mapping COMS Data Types to COBOL74" earlier in this section and to "Using COMS Designators" later in this section.
- See "Calling Service Functions" later in this section for more information on obtaining designators.
- For information on values returned to the output header to indicate errors in destination routing, refer to the *COMS Programming Guide.*
- For more information on a specific requirement for SDF, see Section 6, "Using the SDF Program Interface."

Table 3–3 lists the fields of the output header, showing COMS and COBOL74 field names, the COMS data types, and the COBOL74 usages. The fields are listed alphabetically.

Table 3–3. Output Header Fields

COMS Field Name	COBOL74 Field Name	COMS Data Type	COBOL74 Usage
Agenda Designator	AGENDA	Designator	Real
Casual Output	CASUALOUTPUT	Boolean	DMS Boolean
Destination Count	DESTCOUNT	Integer	Binary
Destination Designator	DESTINATIONDESG	Designator	Real
Delivery Confirmation Flag	CONFIRMFLAG	Boolean	DMS Boolean
Delivery Confirmation Key	CONFIRMKEY	Display	Display
VT Flag	VTFLAG	Boolean	DMS Boolean
Next Input Agenda Designator	NEXTINPUTAGENDA	Designator	Real
Retain Transaction Mode	RETAINTRANSACTIONMODE	Boolean	DMS Boolean
Set Next Input Agenda	SETNEXTINPUTAGENDA	Boolean	DMS Boolean
Status Value	STATUSVALUE	Mnemonic	Binary
Text Length	TEXTLENGTH	Integer	Binary

continued

Table 3–3. Output Header Fields (cont.)

COMS Field Name	COBOL74 Field Name	COMS Data Type	COBOL74 Usage
Transparent	TRANSPARENT	Boolean	DMS Boolean
User-Defined Conversation Area	User defined	User defined	User defined

Using the VT Flag of the Output Header

You can use the VT (virtual terminal) flag bit of the output header with a COMS direct window. The window can have a virtual terminal name when it is used within a CP 2000 environment. The virtual terminal name describes to BNA the way in which the direct window has formatted the output.

A direct-window program can set the VT flag before sending output messages by using the following syntax:

```
COMPUTE VTFLAG OF OUTHDR = TRUE
```

COMS returns the result in the VT Flag field of the input header. You can test the result directly by using the following IF statement:

```
IF VTFLAG OF INHDR...
```

Requesting Delivery Confirmation on Output

Delivery confirmation is available for network support processor (NSP) and CP 2000 stations. This COMS feature informs a direct-window program when a station receives a particular message the window has sent.

To request delivery confirmation, perform the following steps before executing the SEND statement:

- Use the COMPUTE statement to set the Delivery Confirmation Flag field. For example, enter the following:

  ```
  COMPUTE CONFIRMFLAG OF OUTHDR = TRUE
  ```

- Identify a message individually by placing a unique value of your choice into the Delivery Confirmation Key field. When confirming delivery, COMS uses the default input agenda to return the unique value in the first three characters of the message area.

See Also

- For more information on requesting delivery confirmation on output, see the *COMS Programming Guide*.
- Refer to "Using Communication Statements" later in this section for information on receiving and sending messages.

Preparing to Receive and Send Messages

Before you can receive or send messages, you must first link to COMS and initialize the program. The following information explains how to perform these tasks.

Linking an Application Program to COMS

To prepare for receiving or sending messages, you must link your application program to COMS. Linkage of the program to COMS is achieved through the data communications interface (DCI) library, which contains the programmatic interface with COMS. The DCI library is linked to COMS by specifying COMSSUPPORT as a function name and by designating library access by function. Because COMS must be installed as the system library COMSSUPPORT, library access by function is the access method most often used to link an application program to COMS. If you want to test versions of COMS with the currently installed version of COMS, you must specify a different library access method to enable the DCI library to link to the test version. A discussion of the DCI library is provided in "Using the DCI Library" later in this section.

Examples

Examples of library access by function, by initiator, and by title are explained in the following paragraphs.

Example of Linking by Function (Default Method)

The syntax illustrated in this example is the programming equivalent of the default method used by the DCI library. COMS is installed under a different object code file title, for example COMS/ENTRY or COMS/PRODUCTION. Note that a change in the title of the object code file requires appropriate data communications configuration changes. The LIBACCESS library attribute specifies that the function name—not the object code file title—is used to access the library. The FUNCTIONNAME library attribute designates the function name of the library. Additional information on library attributes is provided in the discussion of libraries in Volume 1. Refer to the *A Series System Software Utilities Operations Reference Manual* for a description of the library attributes.

```
CHANGE ATTRIBUTE LIBACCESS OF "DCILIBRARY"
    TO BYFUNCTION.
CHANGE ATTRIBUTE FUNCTIONNAME OF "DCILIBRARY"
    TO "COMSSUPPORT".
```

Example of Linking by Initiator

The following example shows how to link a COMS program by initiator. This method of linking to COMS is the preferred method for COMS programs. All other programs should be linked by function.

```
CHANGE ATTRIBUTE LIBACCESS OF "DCILIBRARY"
    TO BYINITIATOR.
```

Example of Linking by Title

A COMS application program can link to COMS by title. Unisys does not recommend this method. Linking by title might be used, for example, when linkage is needed to a test version of COMS that was not assigned a function name with the SL (support library) system command, and access by initiator is not available on the level of the operating system you have installed.

Note: Linking by title can cause COMS to be initiated at an inappropriate time and with inappropriate attributes (for example, the usercode of the requestor, the family statement, or limits). Linking by title can fail because of the family substitution action if the object code file was renamed since initiation or if a new version of the code file was installed that replaced the code file of the current installation of COMS.

The following example shows how to link your program to COMS by title:

```
     CHANGE ATTRIBUTE TITLE OF "DCILIBRARY" TO
        "SYSTEM/COMS/TEST ON TESTPACK".
* Alternatively, the following constructs can be used.  This
* code is a close equivalent of linking by initiator.
*    MOVE ATTRIBUTE NAME OF ATTRIBUTE EXCEPTIONTASK OF
*        ATTRIBUTE EXCEPTIONTASK OF MYSELF TO workarea.
*    CHANGE ATTRIBUTE TITLE OF "DCILIBRARY" TO workarea.
* Store the family name so it can be temporarily changed.
*    MOVE ATTRIBUTE FAMILY OF MYSELF TO workarea.
* Reset the family name to null so that family substitution
* does not interfere with linkage to COMS by title.
*    CHANGE ATTRIBUTE FAMILY OF MYSELF TO ".".
*    ENABLE INPUT input-header-name KEY "ONLINE".
* Restore the family name for accessing files, etc.
*    CHANGE ATTRIBUTE FAMILY OF MYSELF TO workarea.
```

Initializing an Interface Link

You can initialize the interface between COMS and your program by including the following statement in the PROCEDURE DIVISION:

```
ENABLE INPUT COMS-IN KEY "ONLINE".
```

Example

The following example shows the initialization statement used within the context of code that links an application program to COMS:

```
77 SYSTEM-COMS                  PIC X(50).
.
.
.
START-UP-SECTION                SECTION  50.
START-UP.
    CHANGE ATTRIBUTE LIBACCESS OF "DCILIBRARY"
       TO BYINITIATOR.
    ENABLE INPUT COMS-IN KEY "ONLINE".
```

Using the DCI Library

A DCI library is a library to which the compiler builds references whenever a program uses the ACCEPT, DISABLE, ENABLE, RECEIVE, or SEND statements; in fact, a DCI library must be present for a COBOL program to use these statements.

A DCI library enables programs to deal with symbolic sources and destinations instead of peripherals. Program recompilation when the peripherals are changed or rearranged is therefore avoided. The library reference is built with the title DCILIBRARY and the library object name DCIENTRYPOINT. The DCIENTRYPOINT library object is an untyped procedure.

The DCIENTRYPOINT library object has the following five parameters:

- An integer with a value that specifies one of the following statements:

Value	Statement
1	ACCEPT MESSAGE COUNT
2	DISABLE
3	ENABLE
4	RECEIVE
5	SEND
6	BEGIN-TRANSACTION WITH TEXT
7	BEGIN-TRANSACTION ABORT
9	END-TRANSACTION ABORT
10	END-TRANSACTION WITH NO TEXT
11	END-TRANSACTION WITH TEXT

  All statements except DISABLE, ENABLE, RECEIVE, and SEND are A Series extensions to ANSI-74 COBOL.

- An integer with a value indicating the length, in characters, of the EBCDIC array (unindexed descriptor) that contains the message or the password.

- An integer with a value indicating either the type of end-indicator option to be sent or received or the type of enable or disable operation to be performed. For the SEND or RECEIVE statements, the following values are associated with an indicator option:

Value	Indicator Option
1	End-of-segment indicator (ESI)
2	End-of-message indicator (EMI)
3	End-of-group indicator (EGI)

For the DISABLE or ENABLE statement, the value specifies the device type as follows:

Value	Device Type
11	Input terminal
12	Input terminal
13	Output terminal

- An integer with a value that indicates advancing control when the SEND statement is performed. The parameter also indicates whether or not to wait for messages when the statement is a RECEIVE statement.

For the SEND statement, the value of the parameter specifies advancing as follows:

Value	Advancing Control
0	No advancing
1	Advance after lines
2	Advance before lines
3	Advance after page
4	Advance before page

For the RECEIVE statement, the value of the integer can be set as follows in the program or by the DCI library:

Setting	Integer Value
Program setting	The value indicates to the DCI library whether or not to wait if no message or text is available. A value of 0 (zero) means do not wait, and a value greater than 0 means wait that number of seconds. If this parameter is set to 0 and a NO DATA clause is supplied, no waiting occurs and the action specified in the NO DATA clause is taken if no message is available.
DCI library setting	Before returning control to the program, the DCI library sets the value of the integer to 1 if no text is available; otherwise, a value of 0 (zero) is returned with the text to the program.

- An integer equal to the number of lines the device is to advance when using the SEND statement. If a SEND statement is not used, the value of this parameter is 0 (zero).

Example

You can write the DCI library in COBOL, ALGOL, or DCALGOL to allow access to disk files, remote files, or port files. The symbolic queues, selection algorithms, sources, and destinations—as set out in standard ANSI-74 COBOL—can be tailored to the particular application using the DCI library. The following example shows a typical DCI library object written in COBOL74:

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID. DCIENTRYPOINT.
 DATA DIVISION.
 LINKAGE SECTION.
*
* Integer value of the statement
*
 77 DCI-FUNCTION                COMP PIC 9.
*
* Length of the array that contains the message or password
*
 Ø1 THE-MESSAGE
    Ø3 THE-MESSAGE-SUB             PIC X
       OCCURS 1 TO 9999 DEPENDING ON THE-MESSAGE-LENGTH.
*
* Indicator option or type of enable or disable to be performed
*
 77 THE-MESSAGE-LENGTH          COMP PIC 999.
 *
 * Value of advancing control
 *
 77 IO-OR-END-INDICATOR         COMP PIC 99.
 *
 * Amount to advance in a SEND statement
 *
 77 NO-DATA-OR-ADVANCING-TYPE   COMP PIC 99.
 77 ADVANCING-VALUE             COMP PIC 99.
 PROCEDURE DIVISION
    USING DCI-FUNCTION
          THE-MESSAGE,
          THE-MESSAGE-LENGTH,
          IO-OR-END-INDICATOR,
          NO-DATA-OR-ADVANCING-TYPE,
          ADVANCING-VALUE.
 MAIN SECTION.
   P1.
         PERFORM FUNCTION-SPECIFIED-BY-DCI-FUNCTION.
         EXIT PROGRAM.
```

Using Communication Statements

Communication statements are coded in the PROCEDURE DIVISION of the COBOL74 program. The statements include

- ACCEPT MESSAGE COUNT
- DISABLE
- ENABLE
- RECEIVE
- SEND

General information on using communication statements in a program is provided in the *COMS Programming Guide.*

ACCEPT MESSAGE COUNT Statement

The ACCEPT MESSAGE COUNT statement makes available to you the number of messages in the application queue of the program.

When the ACCEPT MESSAGE COUNT statement is executed, the Status Value field of the input header and Message Count field are appropriately updated.

General Format

The general format of the ACCEPT MESSAGE COUNT statement is as follows:

```
ACCEPT COMS-header-name-1 MESSAGE COUNT
```

Explanation of Format Elements

ACCEPT MESSAGE COUNT

This statement updates the Message Count field to indicate the number of messages present in the queue that COMS maintains for the program.

COMS-header-name-1

This format element is the name of the COMS input header.

DISABLE Statement

The DISABLE statement closes a direct window to a station or disconnects a station reached through a modem or a CP 2000 communications processor.

Information on detaching a station dynamically is provided in the *COMS Programming Guide*.

If the logical disconnection specified by the DISABLE statement has already occurred or is denied by COMS, the data item in the Status Value field is updated. See "Using COMS Input Header Fields" earlier in this section for more information on the Status Value field.

General Format

The general format of the DISABLE statement is as follows:

```
DISABLE INPUT [TERMINAL]
COMS-header-name-1 [ WITH KEY { identifier-1 } ]
                              { literal-1    }
```

Explanation of Format Elements

DISABLE INPUT

This phrase logically disconnects COMS from the specified sources or destinations. If this logical disconnection has already occurred or is handled by means external to the program, the DISABLE statement is not required in the program. In this case, the DISABLE statement does not affect the logical path for the transfer of data between the COBOL74 programs and COMS.

TERMINAL

This optional word specifies that only the data item in the Station field is meaningful.

COMS-header-name-1

This format element specifies the name of the COMS input header.

WITH KEY literal-1 or identifier-1

This format element must be alphanumeric. The KEY options are literals defined for use with COMS. KEY options are described in the *COMS Programming Guide*.

Example

The following example shows how to use the DISABLE statement with the optional word TERMINAL:

```
DISABLE INPUT TERMINAL COMS-IN WITH KEY "RETAIN".
```

ENABLE Statement

The ENABLE statement dynamically opens a direct window to a station not currently attached to COMS. You can check the status of a station attachment by querying values in the fields of the COMS input header.

For information about the use of this statement to dynamically attach a station, refer to the *COMS Programming Guide*.

When the logical connection specified by the ENABLE statement already exists or is denied by COMS. the data item in the Status Value field is updated.

General Format

The general format of the ENABLE statement is as follows:

```
ENABLE INPUT [TERMINAL]
COMS-header-name-1 [ WITH KEY { identifier-1 } ]
                              { literal-1    }
```

Explanation of Format Elements

ENABLE INPUT

This phrase provides a logical connection between COMS and the specified sources or destinations. When this logical connection is already present or is handled by means external to the program, the ENABLE statement is not required in the program. In this case, the ENABLE statement does not affect the logical path for the transfer of data between the COBOL74 program and COMS.

TERMINAL

This option dynamically opens a direct window to a station that is not attached to COMS.

COMS-header-name-1

This format element specifies the name of a COMS output header.

literal-1 or identifier-1

This format element must be alphanumeric. The KEY values provided by COMS are described in the *COMS Programming Guide*.

Examples

The following examples illustrate the use of the KEY values with the ENABLE statement. The first example can be seen within the context of a complete program in line 019100 of Example 3–1, "COMS Sample Program with a DMSII Database," later in this section. For more information on the use of the ENABLE statement to establish communication with the COMS MCS, see "Initializing an Interface Link" earlier in this section.

```
ENABLE INPUT COMS-IN KEY "ONLINE".

ENABLE INPUT COMS-IN KEY "BATCH".

ENABLE INPUT TERMINAL HDR-IN KEY "NOWAIT".

MOVE "WAITNOTBUSY (HOSTNAME = AB10)" TO TEMP.
ENABLE INPUT TERMINAL HDR-IN KEY TEMP.
```

RECEIVE Statement

The RECEIVE statement makes a message and pertinent information about the data available to the COBOL74 program from a queue maintained by COMS. You can use the RECEIVE statement to execute a specific imperative statement when you use the NO DATA option, as shown in the following format diagram.

You can use the RECEIVE statement as many times as needed in the PROCEDURE DIVISION of your program. You can structure your program to receive messages from one or more stations or programs, but you cannot programmatically limit the reception of messages to selected stations on the network or to certain types of programs.

Before you can receive or send messages, however, you must link to COMS and initialize the program. For information, refer to "Preparing to Receive and Send Messages" earlier in this section.

General Format

The general format of the RECEIVE statement is as follows:

```
RECEIVE COMS-header-name-1 MESSAGE INTO identifier-1
[NO DATA imperative-statement-1 ]
[WITH DATA imperative-statement-2]
```

Explanation of Format Elements

COMS-header-name-1

This format element specifies the name of a COMS input header.

MESSAGE INTO identifier-1

This format element references an area to which a message is transferred. The message is aligned to the left without space fill.

NO DATA imperative-statement-1

This option enables COMS to transfer control to the next executable statement when you execute a RECEIVE statement and receive data.

If COMS does not make data available in the identifier-1 data item, one of the following actions takes place when you execute a RECEIVE statement:

- If you specify the NO DATA option, the RECEIVE statement is terminated, which indicates that action is complete. Imperative-statement-1 is executed.
- If you do not specify the NO DATA option, then the object program execution is suspended until data is made available in identifier-1 or until end-of-task (EOT) notification.

Imperative-statement-1 can consist of a NEXT SENTENCE option.

WITH DATA imperative-statement-2

This option enables COMS to transfer control to imperative-statement-2.

Considerations for Use

The data items identified by the COMS input header are appropriately updated by COMS each time the RECEIVE statement is executed.

The following rules apply to the data transfer:

- If a message is the same size as the area of identifier-1, the message is stored in the identifier-1 area.
- If a message is smaller than the area of identifier-1, the message is aligned to the leftmost character position of the identifier-1 area with no space fill.
- If a message is larger than the area of identifier-1, the message fills the identifier-1 area from left to right, starting with the leftmost character of the message. The rest of the message is truncated.

You cannot use a RECEIVE statement between BEGIN-TRANSACTION and END-TRANSACTION statements. Doing so violates the rules of synchronized recovery and you might lose some of the data in your database.

Example

The following example shows part of an application routine that is written to receive a message from COMS and place it in an area defined in the WORKING-STORAGE SECTION. An example of a RECEIVE statement within the context of a complete program begins at line 019700 in Example 3–1, "COMS Sample Program with a DMSII Database," later in this section.

```
Ø1 MSG-IN-TEXT                PIC X(192Ø).
   :
COMMUNICATION SECTION.
INPUT HEADER COMS-IN.
OUTPUT HEADER COMS-OUT.

PROCEDURE DIVISION.
   :
RECEIVE-MSG-FROM-COMS.
    RECEIVE COMS-IN MESSAGE INTO MSG-IN-TEXT.
     IF STATUSVALUE OF COMS-IN = 99
        GO TO EOJ-ROUTINE.
*
*      Process the message.
*
    :
     GO TO RECEIVE-MSG-FROM-COMS.
EOJ-ROUTINE.
     STOP RUN.
```

SEND Statement

The SEND statement releases a message, message segment, or portion of a message to one or more output queues maintained by COMS. Before you can send or receive messages, however, you must link to COMS and initialize the program. For information on these functions, refer to "Preparing to Receive and Send Messages" earlier in this section.

When a SEND statement is executed, the Status Value field is updated by COMS. See the *COMS Programming Guide* for information on the values in the Status Value field.

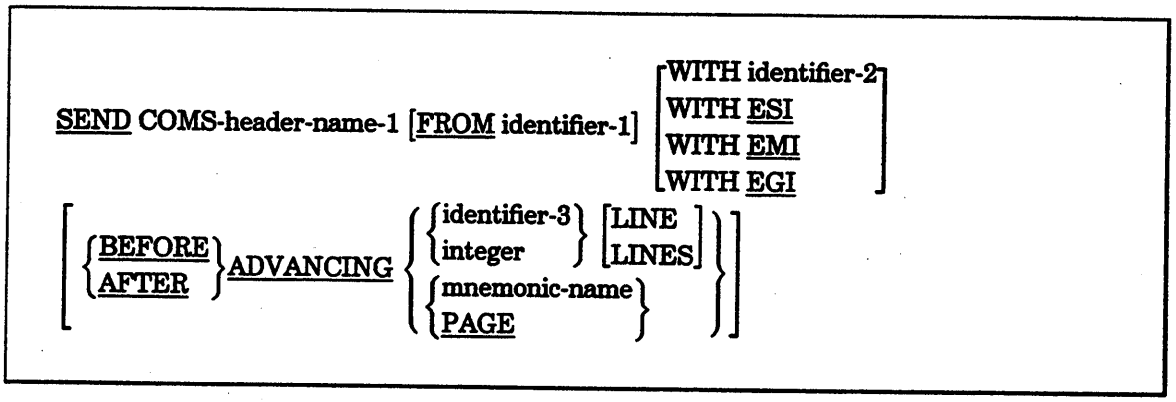
General Formats

The following are two general formats for the SEND statement. The second format is for segmented output.

General Format for the SEND Statement

```
SEND COMS-header-name-1 FROM identifier-1
```

General Format for the SEND Statement with Segmented Output



Explanation of Format Elements

COMS-header-name-1

This format element specifies the name of a COMS output header.

FROM identifier-1

This phrase moves the message or message segment to the sending character positions of the area of identifier-1 and is aligned to the left with space fill.

Identifier-1 specifies the data name of the area where the data to be sent is made available to COMS.

When you execute a SEND statement, COMS interprets the value in the Text Length field of the output header as the number of leftmost character positions of identifier-1 from which data is to be transferred.

If the value of Text Length is 0 (zero), no characters of identifier-1 are transferred. The value of Text Length cannot be outside the range 0 through the size of identifier-1. If the value is outside the range, the message is truncated to the size of identifier-1 and the Status Value field of the output header is set to 0.

The effect of special control characters within identifier-1 is undefined.

Explanation of Format Elements for Segmented Output

The following is an explanation for segmented output only:

WITH

This option designates the nonsegmented output in identifier-2 or a type of segmented output.

identifier-2

This format element must be the name of a 1-character integer without an operational sign, for example, PIC S9(11) USAGE BINARY.

ESI, EMI, and EGI

These options specify the segmenting options. See "Segmenting Options" later in this section for information on the three segmenting options.

BEFORE or AFTER ADVANCING

This option controls the vertical positioning of each message.

identifier-3

This format element is the name of an elementary nonnegative integer item.

integer

This format element specifies the number of lines to advance. The value can be 0 (zero).

mnemonic-name

This format element specifies a name that is identified with a particular feature specified in the SPECIAL-NAMES paragraph of the ENVIRONMENT DIVISION.

Segmenting Options

There are three segmenting options: the end-of-message indicator (EMI), the end-of-group indicator (EGI), and end-of-segment indicator (ESI). COMS recognizes these indicators and establishes the appropriate linkage to maintain control over groups, messages, and segments. For example, the following statement sends output immediately after values have been moved to the required fields of the output header:

```
SEND output-header-name FROM message-area WITH EMI.
```

The contents of identifier-2 indicate that the contents of identifier-1 are to have an associated ESI, EMI, or EGI according to the explanations in Table 3–4.

Table 3–4. Transmission Indicators for Identifier-1

Identifier-2	Identifier-1	Explanation
0	No indicator	No indicator
1	ESI	Message segment complete
2	EMI	Message complete
3	EGI	Group of messages complete

Any character other than 1, 2, or 3 is interpreted as 0 (zero). If identifier-2 is a number other than 1, 2, or 3 and if identifier-1 is not specified, the data is transferred and no error is indicated.

The hierarchy of ending indicators (major to minor) is EGI, EMI, and ESI. An EGI need not be preceded by an ESI or an EMI, and an EMI need not be preceded in the code by an ESI.

A single execution of the SEND statement with segmented output never releases to COMS more than the single message or message segment that is indicated by the contents of the data item referenced by identifier-2 and by the specified ESI, EGI, or EMI. However, the MCS does not transmit any portion of a message to a communications device until the entire message is placed in the output queue.

During the execution of the run unit, the disposition of a portion of a message not terminated by an EMI or EGI is undefined. Thus, the message does not logically exist for the MCS and cannot be sent to a destination.

After the execution of a STOP RUN statement, the system removes any portion of a message transferred from the run unit as a result of a SEND statement, but not terminated by an EMI or EGI. As a result, no portion of the message is sent.

Advancing Options

The ADVANCING option enables you to control the vertical positioning of each message or message segment on an output device where this control is possible. If vertical positioning is not possible on a device, COMS ignores the specified or implied vertical positioning.

On a device where vertical positioning is possible and the ADVANCING option is not specified, the COMS default setting of BEFORE ADVANCING one line is used.

You can use the BEFORE ADVANCING and AFTER ADVANCING options to specify whether the text of a message should be written to the output device before or after the device advances to the next page, or before or after the device advances a specified number of lines. If you specify neither of these options, it is assumed that you are

specifying lines. For example, the following code instructs the device to write a message after advancing one line:

```
SEND COMS-OUT FROM MSG-OUT-TEXT WITH ... AFTER ADVANCING 1 LINE.
```

Although COMS supplies a default setting for carriage control, a processing item can alter carriage control before an output message reaches its destination. For instructions on how to alter carriage control, refer to the *COMS Programming Guide*.

If you specify identifier-3 in the ADVANCING option and the value of identifier-3 is 0, the MCS ignores the ADVANCING option.

If you implicitly or explicitly specify the ADVANCING option, the following rules apply:

- If you use the BEFORE ADVANCING option, the output device writes the message or message segment before it repositions the device vertically according to the rules for identifier-3, integer, and mnemonic-name.

  In the following SEND statement, the BEFORE ADVANCING option instructs the output device to advance two lines after values have been moved into the appropriate fields of the output headers:

  ```
  SEND COMS-OUT FROM COMS-OUT-AREA WITH EMI
      BEFORE ADVANCING 2 LINES.
  ```

- If you use the AFTER ADVANCING option, the output device writes the message or message segment after it repositions the device vertically according to the rules for identifier-3, integer, and mnemonic-name.
- If you specify identifier-3 or integer, the output device repositions characters vertically downward a number of lines equal to the value of identifier-3 or integer.
- If you specify mnemonic-name, the output device positions characters according to the rules for that device.
- If you specify the PAGE advancing control, the output device writes characters either before or after the device is repositioned to the next page (depending on the option used). For example, in the following SEND statement, the AFTER ADVANCING option instructs the output device to advance one page after the message has been displayed:

  ```
  SEND COMS-OUT FROM COMS-OUT-AREA WITH EMI
      AFTER ADVANCING PAGE.
  ```

  If you specify the PAGE advancing control, but PAGE has no meaning in conjunction with a specific device, then the output device advances as if you had specified a BEFORE ADVANCING 1 LINE option or an AFTER ADVANCING 1 LINE option.

Considerations for Use

You cannot use a SEND statement between BEGIN-TRANSACTION and END-TRANSACTION statements. Doing so violates the rules of synchronized recovery and you might lose some of the data in your database.

Example

The following example shows SEND statements that specify segmented output using the WITH ESI and WITH EGI options. The options are specified to hold output until the message is complete.

```
 WORKING-STORAGE SECTION.
Ø1 MESSAGE-1  PIC X(1ØØ).
Ø1 MESSAGE-2  PIC X(1ØØ).
Ø1 MESSAGE-3  PIC X(1ØØ).

 COMMUNICATION SECTION.
 INPUT HEADER INHDR.
 OUTPUT HEADER OUTHDR.
 PROCEDURE DIVISION.
     MOVE OUTPUT-SIZE1 TO TEXTLENGTH OF OUTHDR.
     MOVE Ø TO STATUSVALUE OF OUTHDR.
     SEND OUTHDR FROM MESSAGE-1 WITH ESI.

     MOVE OUTPUT-SIZE2 TO TEXTLENGTH OF OUTHDR.
     MOVE Ø TO STATUSVALUE OF OUTHDR.
     SEND OUTHDR FROM MESSAGE-2 WITH ESI.

     MOVE OUTPUT-SIZE3 TO TEXTLENGTH OF OUTHDR.
     MOVE 1 TO DESTCOUNT.
     MOVE STATIONDESG OF INHDR TO DESTINATIONDESG OF OUTHDR.
     MOVE Ø TO STATUSVALUE OF OUTHDR.
     SEND OUTHDR FROM MESSAGE-3 WITH EGI.
```

Using Service Functions

This discussion of service functions includes explanations of the following:

- COMS designators used with service functions
- Mnemonics used to request information
- Statements used to call service functions
- Parameters used with service function calls

COMS uses the following service functions:

- CONVERT_TIMESTAMP
- GET_DESIGNATOR_ARRAY_USING_DESIGNATOR
- GET_DESIGNATOR_USING_DESIGNATOR
- GET_DESIGNATOR_USING_NAME
- GET_INTEGER_ARRAY_USING_DESIGNATOR
- GET_INTEGER_USING_DESIGNATOR
- GET_NAME_USING_DESIGNATOR

- GET_REAL_ARRAY
- GET_STRING_USING_DESIGNATOR
- STATION_TABLE_ADD
- STATION_TABLE_INITIALIZE
- STATION_TABLE_SEARCH
- TEST_DESIGNATORS

In COBOL74, you can use hyphens (-) rather than underscores (_) in the names of service functions. The compiler automatically translates hyphens to underscores for use with COMS.

For a complete discussion of the COMS service functions and their use, refer to the *COMS Programming Guide*.

Using COMS Designators

Service functions use numeric designators that are part of an internal code understood by COMS. You can obtain designators from COMS headers or from service functions that allow you to translate names to designators.

See Also

- Refer to "Mapping COMS Data Types to COBOL74" earlier in this section for information on the COMS designator data type and its use with COBOL74.
- See the *COMS Programming Guide* for information about COMS designators.

Identifying Information with Service Function Mnemonics

The COMS entities have designators that can be used in service function calls. Table 3–5 lists the service function mnemonics that you can use to identify particular data items. The data items indicate the kinds of information that can be requested in a program.

Table 3–5. Service Function Mnemonics

Data Item	Mnemonic
Agenda	AGENDA
Database	DATABASE
Device designator	DEVICE
Device list	DEVICE-LIST
Installation data	INSTALLATION-DATA
Installation data link by another COMS entity	INSTALLATION-DATA-LINK

continued

Table 3–5. Service Function Mnemonics (cont.)

Data Item	Mnemonic
Installation data string	INSTALLATION-STRING-1
Installation data string	INSTALLATION-STRING-2
Installation data string	INSTALLATION-STRING-3
Installation data string	INSTALLATION-STRING-4
Installation data	INSTALLATION-HEX-1
Installation data	INSTALLATION-HEX-2
Installation data integer	INSTALLATION-INTEGER-ALL
Installation data integer	INSTALLATION-INTEGER-1
Installation data integer	INSTALLATION-INTEGER-2
Installation data integer	INSTALLATION-INTEGER-3
Installation data integer	INSTALLATION-INTEGER-4
Library	LIBRARY
Message date in format MMDDYY	DATE
Message time in format HHMMSS	TIME
Processing item	PROCESSING-ITEM
Processing item list	PROCESSING-ITEM-LIST
Program	PROGRAM
Program: current input queue depth	QUEUE-DEPTH
Program: mix numbers for active copies	MIXNUMBERS
Program: response time for last transaction	LAST-RESPONSE
Program: aggregate response time	AGGREGATE-RESPONSE
Program: security designator	SECURITY
Program: total number of input messages handled	MESSAGE-COUNT
Security category	SECURITY-CATEGORY
Security category list	CATEGORY-LIST
Station	STATION
Station: language attribute	LANGUAGE
Station: convention attribute	CONVENTION

continued

Table 3–5. Service Function Mnemonics (cont.)

Data Item	Mnemonic
Station list	STATION-LIST
Station: logical station number (LSN)	LSN
Station: security designator	SECURITY
Statistics	STATISTICS
Transaction code	TRANCODE
Window	WINDOW
Window: maximum number of users	MAXIMUM-USER-COUNT
Window: current number of users	CURRENT-USER-COUNT
Window list	WINDOW-LIST
Usercode	USERCODE

Calling Service Functions

You can call the COMS service functions with application programs and processing items. When you call a service function, do the following:

- Use the CALL statement syntax for calling library procedures in COBOL74. The CALL statement is explained in detail in Volume 1.
- Pass the integer parameters (using unscaled integer values) by name rather than by value.

Do not perform arithmetic computations on the values returned from the procedure calls. You can move the values to other locations of a compatible type within your program, and you can pass them as parameters when calling other library objects. For information on compatible types, refer to "Mapping COMS Data Types to COBOL74" earlier in this section.

Note: If the library being called is DCILIBRARY, you must use hyphens (-) in the mnemonic names. The hyphens are automatically translated to underscores (_) by the compiler for use with COMS.

Example

The following example shows the use of the CALL statement to pass an agenda designator to obtain an agenda name. The agenda designator and name used in the example are also declared in the WORKING-STORAGE SECTION of the program.

The service function result value indicates the result of the service function call. The result value is returned to the parameter specified in the GIVING clause of the CALL statement.

```
CALL "GET-NAME-USING-DESIGNATOR OF DCILIBRARY"
        USING agenda-designator,
              agenda-name
        GIVING service-function-result-value
```

See Also

- For more information on the GET_NAME_USING_DESIGNATOR service function, and for an example of the CALL statement used for this service function, see "GET_NAME_USING_DESIGNATOR Service Function" later in this section.
- Refer to the *COMS Programming Guide* for information about the service function result values.
- Refer to "Passing Parameters to Service Functions" later in this section for information on valid parameters and examples of the CALL syntax.

General Format of the VALUE Option

You can use the VALUE option with the CALL statement for COMS service functions in which a mnemonic parameter is passed for a numeric result.

The format for the VALUE option is as follows:

```
[VALUE mnemonic-1]
```

Explanation of Format Elements

VALUE

This parameter indicates that a service function mnemonic is being passed by value to the service function.

mnemonic-1

This identifier designates the mnemonic parameter.

See Also

Refer to "Identifying Information with Service Function Mnemonics" earlier in this section for information on valid mnemonics.

Example

The following example uses the CALL statement with the VALUE parameter. The LSN-NUM variable contains the value of the LSN service function mnemonic. SF-RSLT receives the result of the procedure call. If the result of the procedure call is successful, SF-RSLT contains a 0 (zero); otherwise, SF-RSLT contains an error code.

Note that the GET_INTEGER_USING_DESIGNATOR service function has hyphens (-) between the words in its name because the DCILIBRARY library is the library called.

For information on valid parameters and an example of program code using the GET_INTEGER_USING_DESIGNATOR service function, see "GET_INTEGER_USING_DESIGNATOR Service Function" later in this section.

```
.
.
WORKING-STORAGE SECTION.
.
77 HDR-STATION    REAL.
77 LSN-NUM        PIC S9(11) USAGE BINARY.
77 SF-RSLT        PIC S9(11) USAGE BINARY.
.
.
PROCEDURE DIVISION.
.
.
   CALL "GET-INTEGER-USING-DESIGNATOR OF DCILIBRARY"
        USING HDR-STATION
              VALUE LSN
              LSN-NUM
        GIVING SF-RSLT.
```

Passing Parameters to Service Functions

The following parameters can be passed to service functions:

- Designators
- Mnemonics
- Arrays

Designators and arrays must be declared in the WORKING-STORAGE SECTION of your COBOL74 program.

You cannot pass header fields as parameters. To pass fields, you must first move them to a declared temporary parameter, and then pass the values. The temporary parameter must be declared as a real value.

The service function declarations and valid parameters for passing service functions in a COBOL74 program are explained on the following pages. Note the following general characteristics:

- An entity name is a data item with DISPLAY usage.
- A mnemonic is a COMS mnemonic representing an entity or a designator.
- An array is an EBCDIC or integer array.
- A designator is a data item of type real.

See Also

- For general information about passing service function parameters, refer to the *COMS Programming Guide*.
- For information on service function mnemonics used in COBOL74, refer to "Identifying Information with Service Function Mnemonics" earlier in this section.
- Refer to the coded examples of declarations of designator, integer, and real tables that are provided at the end of the discussion of each service function.

CONVERT_TIMESTAMP Service Function

The CONVERT_TIMESTAMP service function converts the COMS timestamp TIME(6) to the date or time in EBCDIC. For information on the COMS timestamp TIME(6) type and COBOL74 usage, refer to "Mapping COMS Data Types to COBOL74" earlier in this section.

The input parameters are the following:

- A real value that represents the timestamp.
- A mnemonic that represents the requested function. The allowable mnemonics include DATE, which returns MMDDYY, and TIME, which returns HHMMSS.

The output parameter is an EBCDIC array in which the time or date is returned.

Example

In the following example of the CONVERT_TIMESTAMP service function, SF-RSLT receives the result of the call to the service function. If the result of the call is successful, SF-RSLT contains a 0 (zero); otherwise, SF-RSLT contains an error code.

```
77 WS-TIMESTAMP                    REAL.
Ø1 WS-TIME                         PIC X(6).
77 SF-RSLT                         PIC S9(11) USAGE BINARY.
   .
   .
   .
   CALL "CONVERT-TIMESTAMP OF DCILIBRARY"
        USING WS-TIMESTAMP
              VALUE TIME
              WS-TIME
        GIVING SF-RSLT.
```

GET_DESIGNATOR_ARRAY_USING_DESIGNATOR Service Function

The GET_DESIGNATOR_ARRAY_USING_DESIGNATOR service function obtains a designator list representing the list of stations associated with the STATION LIST designator.

The input parameters are the following:

- The STATION LIST designator. Because the STATION LIST designator is not a field of the header, you must first obtain the designator representing the station list by using the GET_DESIGNATOR_USING_NAME service function. Then use the STATION LIST designator supplied by that function as the designator variable in the GET_DESIGNATOR_ARRAY_USING_DESIGNATOR function.
- The total number of designators in the station list by COMS.

The output parameter is a real array containing the designators.

Example

In the following example of the GET_DESIGNATOR_ARRAY_ USING_DESIGNATOR service function, SF-RSLT receives the result of the call to the service function. If the result of the call is successful, SF-RSLT contains a 0 (zero); otherwise, SF-RSLT contains an error code.

```
77 WS-TABLE-DESG                    REAL.
77 WS-DESG-TABLE-MAX-INDEX          PIC S9(11) USAGE BINARY.
Ø1 WS-DESG-TABLE                    REAL.
   Ø5 WS-D-TABLE                    REAL OCCURS 1Ø TIMES.
77 SF-RSLT                          PIC S9(11) USAGE BINARY.
   .
   .
   .
   CALL "GET-DESIGNATOR-ARRAY-USING-DESIGNATOR OF DCILIBRARY"
        USING WS-TABLE-DESG
              WS-DESG-TABLE-MAX-INDEX
              WS-DESG-TABLE
        GIVING SF-RSLT.
```

GET_DESIGNATOR_USING_DESIGNATOR Service Function

The GET_DESIGNATOR_USING_DESIGNATOR service function obtains a specific designator from the structure represented by the designator.

The input parameters are the following:

- A designator that represents the structure.
- A mnemonic that describes the requested integer array. The mnemonics allowed for the designators are as follows:

Designator	Mnemonic
All	INSTALLATION_DATA_LINK
Station, usercode, program	SECURITY
Station	DEVICE

The output parameter is a designator.

Example

In the following example of the GET_DESIGNATOR_USING_DESIGNATOR service function, SF-RSLT receives the result of the call to the service function. If the result of the call is successful, SF-RSLT contains a 0 (zero); otherwise, SF-RSLT contains an error code.

```
77 WS-DESG                  REAL.
77 WS-DESG-RSLT             REAL.
77 SF-RSLT                  PIC S9(11) USAGE BINARY.
  .
  .
  .
  CALL "GET-DESIGNATOR-USING-DESIGNATOR OF DCILIBRARY"
       USING WS-DESG
             VALUE INSTALLATION-DATA-LINK
             WS-DESG-RSLT
       GIVING SF-RSLT.
```

GET_DESIGNATOR_USING_NAME Service Function

The GET_DESIGNATOR_USING_NAME service function converts the COMS name variable into a designator.

The input parameters are the following:

- A mnemonic that is the entity type for the required name.
- An entity name. The string for the entity name of an agenda, a transaction code (trancode), and installation data includes the window name if the program calling the service function is running in another window or is outside COMS. For example, the following input passes the entity name:

  ```
  agenda-name of window-name
  ```

  For installation data, the data with a window value equal to the ALL entity (the default value) is used if no window is specified and if the window in which the program is running does not have an entity of the same name.

The output parameter is a designator.

Example

In the following example of the GET_DESIGNATOR_USING_NAME service function, SF-RSLT receives the result of the call to the service function. If the result of the call is successful, SF-RSLT contains a 0 (zero); otherwise, SF-RSLT contains an error code.

```
01 WS-NAME                    PIC X(30).
77 WS-DESG                    REAL.
77 SF-RSLT                    PIC S9(11) USAGE BINARY.
   .
   .
   .
   CALL "GET-DESIGNATOR-USING-NAME OF DCILIBRARY"
        USING WS-NAME
              VALUE STATION-LIST
              WS-DESG
        GIVING SF-RSLT.
```

GET_INTEGER_ARRAY_USING_DESIGNATOR Service Function

The GET_INTEGER_ARRAY_USING_DESIGNATOR service function obtains an array of integers from the structure represented by the designator.

The input parameters are the following:

- A designator that represents the structure.
- A mnemonic that describes the requested integer array. The mnemonics allowed for designators are as follows:

Designator	Mnemonic
All	INSTALLATION_INTEGER_ALL
Program	MIXNUMBERS

The output parameters are the following:

- An integer representing the number of integers returned in the array
- An integer array containing the returned information

Example

In the following example of the GET_INTEGER_ARRAY_ USING_DESIGNATOR service function, SF-RSLT receives the result of the call to the service function. If the result of the call is successful, SF-RSLT contains a 0 (zero); otherwise, SF-RSLT contains an error code.

```
    .
    .
    .
77 WS-DESG                    REAL.
77 WS-INT-TABLE-MAX-INDEX     PIC S9(11) USAGE BINARY.
77 SF-RSLT                    PIC S9(11) USAGE BINARY.
Ø1 WS-INT-TABLE               USAGE BINARY.
   Ø3 WS-INT-TABLE-DETAIL     PIC S9(11) OCCURS 1Ø TIMES.
   .
   .
   .
   CALL "GET-INTEGER-ARRAY-USING-DESIGNATOR OF DCILIBRARY"
        USING WS-DESG
              VALUE INSTALLATION-INTEGER-ALL
              WS-INT-TABLE-MAX-INDEX
              WS-INT-TABLE
        GIVING SF-RSLT.
```

GET_INTEGER_USING_DESIGNATOR Service Function

The GET_INTEGER_USING_DESIGNATOR service function obtains a specific integer from the structure represented by the designator.

The input parameters are the following:

- A designator representing the structure.
- A mnemonic that describes the requested integer. Allowable mnemonics and designators are as follows:

Designator	Mnemonic
All	INSTALLATION_INTEGER_1, 2, 3, 4
Station	LSN
Window	MAXIMUM_USER_COUNT and CURRENT_USER_COUNT
Program	QUEUE_DEPTH, MESSAGE_COUNT, LAST_RESPONSE, and AGGREGATE_RESPONSE

The result parameter is an integer.

Example

In the following example of the GET_INTEGER_USING_DESIGNATOR service function, SF-RSLT receives the result of the call to the service function. If the result of the call is successful, SF-RSLT contains a 0 (zero); otherwise, SF-RSLT contains an error code.

```
77 WS-DESG                    REAL.
77 WS-INT                     PIC S9(11) USAGE BINARY.
77 SF-RSLT                    PIC S9(11) USAGE BINARY.
   .
   .
   .
   CALL "GET-INTEGER-USING-DESIGNATOR OF DCILIBRARY"
        USING WS-DESG
              VALUE INSTALLATION-INTEGER-1
              WS-INT
        GIVING SF-RSLT.
```

GET_NAME_USING_DESIGNATOR Service Function

The GET_NAME_USING_DESIGNATOR service function converts a COMS designator to the COMS designator name.

The input parameter is a designator. All designators are allowed.

The result parameter is an entity name.

Example

In the following example of the GET_NAME_USING_DESIGNATOR service function, SF-RSLT receives the result of the call to the service function. If the result of the call is successful, SF-RSLT contains a 0 (zero); otherwise, SF-RSLT contains an error code.

```
77 WS-DESG                    REAL.
Ø1 WS-NAME                    PIC X(3Ø).
77 SF-RSLT                    PIC S9(11) USAGE BINARY.
   .
   .
   .
   CALL "GET-NAME-USING-DESIGNATOR OF DCILIBRARY"
        USING WS-DESG
              WS-NAME
        GIVING SF-RSLT.
```

GET_REAL_ARRAY Service Function

The GET_REAL_ARRAY service function obtains a data structure with no connection to any entity.

The input parameter is a mnemonic representing the requested function. The only allowable mnemonic is STATISTICS, which returns a table. Each table entry contains the following six items. (The input parameter items that show *DCI library* in parentheses are passed from DCI library programs only, and not from remote files.)

- Entity designator
- Type of entity, as follows:

Entry Code	Entity Represented
1	DCI library program
2	Remote file interface
3	MCS window

- Queue depth (DCI library)
- Number of transactions
- Last transaction response in milliseconds (DCI library)
- Aggregate response in milliseconds (DCI library)

The result parameters are the following:

- An integer representing the number of elements returned in the array
- An array containing the returned information

Example

In the following example of the GET_REAL_ARRAY service function, SF-RSLT receives the result of the call to the service function. If the result of the call is successful, SF-RSLT contains a 0 (zero); otherwise, SF-RSLT contains an error code.

```
77 WS-REAL-TABLE-MAX-INDEX          PIC S9(11) USAGE BINARY.
Ø1 WS-REAL-TABLE                    REAL.
   Ø5 WS-R-TABLE                    REAL OCCURS 5ØØ TIMES.
77 SF-RSLT                          PIC S9(11) USAGE BINARY.
    .
    .
    .
    CALL "GET-REAL-ARRAY OF DCILIBRARY"
         USING VALUE STATISTICS
               WS-REAL-TABLE-MAX-INDEX
               WS-REAL-TABLE
         GIVING SF-RSLT.
```

GET_STRING_USING_DESIGNATOR Service Function

The GET_STRING_USING_DESIGNATOR service function obtains an EBCDIC string from the structure represented by the designator.

The input parameters are the following:

- An entity designator that represents the structure.
- An entity mnemonic that describes the requested integer vector. The allowable mnemonics are INSTALLATION_STRING_1 through INSTALLATION_STRING_4; INSTALLATION_HEX_1 and INSTALLATION_HEX_2; LANGUAGE; and CONVENTION.

  INSTALLATION_STRING_1 through INSTALLATION_STRING_4, INSTALLATION_HEX_1, and INSTALLATION_HEX_2 can be used to return strings that you have set up as installation data in the COMS utility.

  The LANGUAGE and CONVENTION mnemonics provide a way of determining the language and convention attributes of the stations to which the program sends or receives messages.

The result parameters are the following:

- An integer that indicates the number of valid characters in the string
- An EBCDIC array indicating the returned string

Example

In the following example of the GET_STRING_USING_DESIGNATOR service function, SF-RSLT receives the result of the call to the service function. If the result of the call is successful, SF-RSLT contains a 0 (zero); otherwise, SF-RSLT contains an error code.

```
77  WS-DESG                  REAL.
77  WS-STRING-MAX-CHAR       PIC S9(11) USAGE BINARY.
Ø1  WS-STRING                PIC X(3Ø).
77  SF-RSLT                  PIC S9(11) USAGE BINARY.
      .
      .
      .
      MOVE SPACES TO WS-STRING.
      CALL "GET-STRING-USING-DESIGNATOR OF DCILIBRARY"
           USING WS-DESG
                 VALUE INSTALLATION-STRING-1
                 WS-STRING-MAX-CHAR
                 WS-STRING
           GIVING SF-RSLT.
```

STATION_TABLE_ADD Service Function

The STATION_TABLE_ADD service function adds a station designator to a table of station designators. The table is controlled by the transaction processor.

The input parameters are the following:

- A real array that contains the table of station designators
- A station designator that is to be added to the table

An example of the use of this service function is provided at the end of the explanation of the STATION_TABLE_SEARCH service function.

STATION_TABLE_INITIALIZE Service Function

The STATION_TABLE_INITIALIZE service function initializes the table into which the station index values are placed. Unisys recommends that the size of the table be based on the number of stations that exist.

The input parameters are the following:

- A real table of station designators. The table is implemented as a hash table.
- A table modulus. You use the modulus to determine the density and access time of the table. If, for example, you have a table with a sparse population and you desire very fast access time, select a modulus that is twice the maximum number of table entries. You should use a modulus with half the maximum number of entries if the table has a compact population and slower access is acceptable.

An example of the use of this service function is provided at the end of the explanation of the STATION_TABLE_SEARCH service function.

STATION_TABLE_SEARCH Service Function

The STATION_TABLE_SEARCH service function searches a table and locates a station designator.

The input parameters are the following:

- The name of the real table of designators to be searched
- The name of the stations designator to be found

If the designator is found, the value of the table index is returned; however, if the returned value is 0 (zero), the station designator was not found.

Example

The following example shows the declarations and statements for the station table service functions. After the execution of the code in the example, STATION-SEARCH-RESULT will contain the index of the station designator in the hash table.

```
.
.
.
WORKING STORAGE SECTION.
.
.
.
Ø1  STATION-HASH                    REAL.
    Ø2  STATION-DESIGNATOR          REAL OCCURS 1ØØ.
77  STATION-SEARCHRESULT           PIC S9(11) BINARY.
77  STATION-SEARCH-DESIGNATOR      REAL.
77  STATION-SEARCH-MODULUS         PIC S9(11) BINARY.
.
.
.
PROCEDURE DIVISION.
.
.
.

      CALL "STATION_TABLE_INITIALIZE OF DCILIBRARY"
        USING STATION-HASH, STATION-SEARCH-MODULUS.
.
.
      MOVE STATION OF COMS-IN TO STATION-SEARCH-DESIGNATOR.
      CALL "STATION_TABLE_SEARCH OF DCILIBRARY"
        USING STATION-HASH, STATION-SEARCH-DESIGNATOR
        GIVING STATION-SEARCH-RESULT.
      IF STATION-SEARCH-RESULT IS EQUAL TO Ø
        CALL "STATION_TABLE_ADD OF DCILIBRARY"
          USING STATION-HASH, STATION-SEARCH-DESIGNATOR
          GIVING STATION-SEARCH-RESULT.
```

TEST_DESIGNATORS Service Function

The TEST_DESIGNATORS service function determines whether a designator is part of a list included in the other designator. The input parameters are two designators representing structures. The allowable designator combinations must include an array of designators using either the mnemonics DEVICE and DEVICE_LIST or the mnemonics SECURITY and SECURITY_CATEGORY. The order in which the parameters for the designators appear in the code does not matter.

If one designator is contained within the list of the other designator, a value of 0 is returned. If not, an error occurs and a value of 2 is returned.

Example

In the following example of the TEST_DESIGNATORS service function, SF-RSLT receives the result of the call to the service function:

```
77 WS-DESG                          REAL.
77 WS-DESG-RSLT                     REAL.
77 SF-RSLT                          PIC S9(11) USAGE BINARY.
   .
   .
   .
   CALL "TEST-DESIGNATORS OF DCILIBRARY"
        USING WS-DESG
              WS-DESG-RSLT
        GIVING SF-RSLT.
```

COMS Sample Programs

The two sample programs track sailboat races. Each updates a database by using features of the COMS direct-window interface. Although both programs have the same purpose and both use the same COMS features, they illustrate the use of different databases. The first program, SAILOLPROG, uses a DMSII database; the second program, ONLINESAIL, uses a SIM database.

COMS Sample Program with a DMSII Database

The SAILOLPROG program maintains the SAILDB database. The program contains three transactions. Each transaction has a unique trancode and a unique module function index (MFI). CRERAC creates a race entry in the database. ADDENT adds a boat entry to a race. The race must exist for the ADDENT transaction to be completed. DELENT deletes a boat from a race.

The program exemplifies the programming techniques used in writing transaction processors that allow synchronized recovery. The program also shows the use of the Test and Debug System (TADS).

COMS Features Used in the Sample Program

The following features of the COMS direct-window interface are used in the program:

- Declared COMS input and output headers
- Trancodes
- Synchronized recovery

The SAILOLPROG program runs in a COMS environment that has been configured to include a DMSII database called SAILDB and the following COMS entities:

- A direct window called SAIL
- An agenda called SAILAGINOL
- The following three trancodes:
  - CRERAC (MFI = 1)
  - ADDENT (MFI = 2)
  - DELENT (MFI = 3)

All entities must be defined to COMS to allow the program to run.

See Also

- Information on synchronized recovery when COMS is used with DMSII is provided in the explanation of the CLOSE statement under "CLOSE Statement" in Section 4, "Using the DMSII Program Interface."
- Information on TADS is provided in the *A Series COBOL ANSI-74 Test and Debug System (TADS) Programming Guide*.

Data Sets in the Database

The database SAILDB contains three data sets. The data set RACE-CALENDAR contains one record for every race. The data set ENTRY contains one record for each boat entered in a particular race. A boat can have several records, depending on the number of races entered. The data set RDS is the restart data set with the appropriate fields requested by COMS. The DMSII option RDSSTORE is not set for the database.

Using the Sample Program

The SAILOLPROG program is shown in Example 3–1. This representation of the program contains comments to indicate the program actions at each step.

All transactions in the program are two-phase transactions. In phase 1, all records are locked. In phase 2, the data is stored in the database and only the END-TRANSACTION statement unlocks records. All transactions instruct COMS to audit the input message when the END-TRANSACTION statement takes the program out of transaction state.

```
001000$SET DICTIONARY = TEST.
010100$SET TADS(REMOTE RMT)
010200 IDENTIFICATION DIVISION.
010300 PROGRAM-ID. ONLINESAIL.
010400 ENVIRONMENT DIVISION.
010500 CONFIGURATION SECTION.
010600     SOURCE-COMPUTER. MICROA.
010700     OBJECT-COMPUTER. MICROA.
010800 INPUT-OUTPUT SECTION.
010900 FILE-CONTROL.
011000    SELECT RMT ASSIGN TO REMOTE ACCESS SEQUENTIAL.
011100 DATA DIVISION.
011200 FILE SECTION.
011300 FD RMT.
011400 01 REM-REC                      PIC X(72).
011500 DATA-BASE SECTION.
011600     DB SAILDB  ALL.
011700 WORKING-STORAGE SECTION.
011800 77  SCRATCH                     PIC X(256).
011900
012000*************************************************************************
012100*  MESSAGE AREA DECLARATIONS                                            *
012200*************************************************************************
012300
012400 01  MSG-TEXT.
012500     03 MSG-TCODE                PIC X(6).
012600     03 MSG-FILLER               PIC X.
012700     03 MSG-CREATE-RACE.
012800         05 MSG-CR-ID            PIC 9(6).
012900         05 MSG-CR-NAME          PIC X(20).
013000         05 MSG-CR-DATE          PIC X(6).
013100         05 MSG-CR-TIME          PIC X(4).
013200         05 MSG-CR-LOCATION      PIC X(20).
013300         05 MSG-CR-SPONSOR       PIC X(10).
013350         05 FILLER               PIC X(10).
013400     03 MSG-ADD-ENTRY REDEFINES MSG-CREATE-RACE.
013500         05 MSG-AE-RACE-ID       PIC 9(6).
013600         05 MSG-AE-ID            PIC X(6).
013700         05 MSG-AE-NAME          PIC X(20).
013800         05 MSG-AE-RATING        PIC 9(3).
013900         05 MSG-AE-OWNER         PIC X(20).
014000         05 MSG-AE-CLUB          PIC X(15).
014100         05 FILLER               PIC X(6).
014200     03 MSG-DELETE-ENTRY REDEFINES MSG-CREATE-RACE.
014300         05 MSG-DE-RACE-ID       PIC 9(6).
014400         05 MSG-DE-ID            PIC X(6).
014500         05 FILLER               PIC X(64).
014600     03 MSG-STATUS               PIC X(30).
014700
```

Example 3–1. COMS Sample Program with a DMSII Database

```
014710*****************************************************************
014720*  COMS INTERFACE DECLARATIONS                                  *
014730*****************************************************************
014800 COMMUNICATION SECTION.
017000
017002 INPUT HEADER COMS-IN;
017004   PROGRAMDESG            IS COMS-IN-PROGRAM;
017006   FUNCTIONSTATUS         IS COMS-IN-FUNCTION-STATUS;
017008   FUNCTIONINDEX          IS COMS-IN-FUNCTION-INDEX;
017010   USERCODE               IS COMS-IN-USERCODE;
017012   SECURITYDESG           IS COMS-IN-SECURITY-DESG;
017014   TRANSPARENT            IS COMS-IN-TRANSPARENT;
017016   VTFLAG                 IS CONS-IN-VT-FLAG;
017018   TIMESTAMP              IS COMS-IN-TIMESTAMP;
017020   STATION                IS COMS-IN-STATION;
017022   TEXTLENGTH             IS COMS-IN-TEXT-LENGTH;
017024   STATUSVALUE            IS COMS-IN-STATUS-KEY;
017026   MESSAGECOUNT           IS COMS-IN-MSG-COUNT;
017028   RESTART                IS COMS-IN-RST-LOC;
017030   AGENDA                 IS COMS-IN-AGENDA;
017032   CONVERSATION AREA.
017033     02 CA PIC X(60).
017034
017036 OUTPUT HEADER COMS-OUT;
017038   DESTCOUNT              IS COMS-OUT-COUNT;
017040   TEXTLENGTH             IS COMS-OUT-TEXT-LENGTH;
017042   STATUSVALUE            IS COMS-OUT-STATUS-KEY;
017044   TRANSPARENT            IS COMS-OUT-TRANSPARENT;
017046   VTFLAG                 IS COMS-OUT-VT-FLAG;
017048   CONFIRMFLAG            IS COMS-OUT-CONFIRM-FLAG;
017050   CONFIRMKEY             IS COMS-OUT-CONFIRM-KEY;
017052   DESTINATIONDESG        IS COMS-OUT-DESTINATION;
017054   NEXTINPUTAGENDA        IS COMS-OUT-NEXT-INPUT-AGENDA;
017055   CASUALOUTPUT           IS COMS-OUT-CASUAL-OUTPUT;
017056   SETNEXTINPUTAGENDA     IS COMS-OUT-SET-NEXT-INPUT-AGENDA;
017058   RETAINTRANSACTIONMODE  IS COMS-OUT-SAVE-TRANS-MODE;
017060   AGENDA                 IS COMS-OUT-AGENDA;
017100*****************************************************************
017200 PROCEDURE DIVISION.
017300*****************************************************************
017400 DECLARATIVES.
017500 DMERROR-SECTION SECTION.
017600     USE ON DMERROR.
017700 DMERROR-PARA.
017800
017900 END DECLARATIVES.
018000
018100 005-MAIN SECTION.
018110*    LINK APPLICATION PROGRAM TO COMS.
```

Example 3–1. COMS Sample Program with a DMSII Database (cont.)

```
018200 005-MAIN-SN.
018500     CHANGE ATTRIBUTE LIBACCESS OF
018600          "DCILIBRARY" TO BYINITIATOR.
018900     OPEN UPDATE SAILDB.
019000     IF DMSTATUS(DMERROR) CALL SYSTEM DMTERMINATE.
019010*    INITIALIZE INTERFACE TO COMS.
019100     ENABLE INPUT COMS-IN KEY "ONLINE".
019200     CREATE RDS.
019300*    MOVE COMS-IN-PROGRAM     TO RDS-PROG.
019400*    MOVE COMS-IN-RST-LOC     TO RDS-LOCATOR.
019420     PERFORM 007-PROCESS-COMS-INPUT
019440       UNTIL COMS-IN-STATUS-KEY = 99.
019500 005-MAIN-EXIT.
019510     PERFORM 910-CLOSEDOWN.
019520     STOP RUN.
019530
019540*****************************************************************
019550 007-PROCESS-COMS-INPUT SECTION.
019560*****************************************************************
019562*    Get the next message from COMS.  If the value is a 99,
019564*    go to the end of task (EOT); otherwise, make sure that the
019566*    message is valid before processing it.
019568*
019570 007-PROCESS-CI-SN.
019600     MOVE SPACES TO MSG-TEXT.
019700     RECEIVE COMS-IN MESSAGE INTO MSG-TEXT.
019800     IF COMS-IN-STATUS-KEY NOT = 99
020000       PERFORM 920-CHECK-COMS-INPUT-ERRORS
020100       IF ( COMS-IN-STATUS-KEY =   0 OR 92)  AND
020200            COMS-IN-FUNCTION-STATUS NOT < 0 THEN
020300         PERFORM 100-PROCESS-TRANSACTION.
020400 007-PROCESS-CI-EXIT.
020450     EXIT.
020500
021000*****************************************************************
021100 100-PROCESS-TRANSACTION  SECTION.
021200*****************************************************************
021220*    Because the transaction type is programmatically based on
021240*    the MFI, make sure that the type is within range before
021260*    exiting to the appropriate subprogram.
021280*
021300 100-PROCESS-TRANS-SN.
021400     IF MSG-TCODE = "CRERAC"
021500       PERFORM 200-CREATE-RACE
021600     ELSE
021700     IF MSG-TCODE = "ADDENT"
021800       PERFORM 300-ADD-ENTRY
021900     ELSE
022000     IF MSG-TCODE = "DELENT"
022100       PERFORM 400-DELETE-ENTRY
022200     ELSE
```

Example 3–1. COMS Sample Program with a DMSII Database (cont.)

```
022300       MOVE "INVALID TRANS CODE" TO MSG-STATUS
022320       PERFORM 900-SEND-MSG.
022340
022400 100-PROCESS-TRANS-EXIT.
022500     EXIT.
022600
022700*****************************************************************
022800 200-CREATE-RACE  SECTION.
022900*****************************************************************
023000 200-CREATE-RACE-SN.
023020*    Enter a new race record in the database.
023030*    Because the transaction is done in online mode,
023040*    save the restart data set (RDS) in the conversation area
023050*    only.  If there is an abort on the BEGIN-TRANSACTION or
023060*    END-TRANSACTION statement, return to the RECEIVE statement.
023080
023100     CREATE RACE-CALENDAR.
023200     MOVE MSG-CR-NAME     TO RACE-NAME.
023300     MOVE MSG-CR-ID       TO RACE-ID.
023400     MOVE MSG-CR-DATE     TO RACE-DATE.
023500     MOVE MSG-CR-TIME     TO RACE-TIME.
023600     MOVE MSG-CR-LOCATION TO RACE-LOCATION.
023700     MOVE MSG-CR-SPONSOR  TO RACE-SPONSOR.
023800
023900*    BEGIN-TRANSACTION COMS-IN USING MSG-TEXT NO AUDIT RDS
023950     BEGIN-TRANSACTION COMS-IN                NO-AUDIT RDS
024000        ON EXCEPTION
024100        IF DMSTATUS(ABORT)
024200*         THEN MUST GET BACK TO RECEIVE STMT
024300          GO TO 200-CREATE-RACE-EXIT
024400        ELSE
024500          CALL SYSTEM DMTERMINATE.
024600*    MOVE COMS-IN-RST-LOC TO RDS-LOCATOR.
024700     STORE RACE-CALENDAR.
024800     IF DMSTATUS(DMERROR)
024900       MOVE "STORE ERROR" TO MSG-STATUS
025000     ELSE
025100       MOVE "RACE ADDED" TO MSG-STATUS.
025200     END-TRANSACTION COMS-OUT AUDIT RDS
025300       ON EXCEPTION
025400       IF DMSTATUS(ABORT)
025500         GO TO 200-CREATE-RACE-EXIT
025600       ELSE
025700         CALL SYSTEM DMTERMINATE.
025800     PERFORM 900-SEND-MSG.
025900 200-CREATE-RACE-EXIT.
026000     EXIT.
026100
026200*****************************************************************
026300 300-ADD-ENTRY   SECTION.
026400*****************************************************************
```

Example 3–1. COMS Sample Program with a DMSII Database (cont.)

```
026500 300-ADD-ENTRY-SN.
026520*    Enter a boat in a race. The restart requirements are the same
026540*     as for the previous transaction.
026560
026600     FIND RACE-SET AT RACE-ID = MSG-AE-RACE-ID
026700       ON EXCEPTION
026800         IF DMSTATUS(NOTFOUND)
026900           MOVE "RACE DOES NOT EXIST" TO MSG-STATUS
027000           PERFORM 900-SEND-MSG
027100           GO TO 300-ADD-ENTRY-EXIT
027200         ELSE
027300           CALL SYSTEM DMTERMINATE.
027400
027500     CREATE ENTRY.
027600     MOVE MSG-AE-NAME      TO ENTRY-BOAT-NAME.
027700     MOVE MSG-AE-ID        TO ENTRY-BOAT-ID.
027800     MOVE MSG-AE-RATING    TO ENTRY-BOAT-RATING.
027900     MOVE MSG-AE-OWNER     TO ENTRY-BOAT-OWNER.
028000     MOVE MSG-AE-CLUB      TO ENTRY-AFF-Y-CLUB.
028100     MOVE MSG-AE-RACE-ID   TO ENTRY-RACE-ID.
028200
028300*    BEGIN-TRANSACTION COMS-IN USING MSG-TEXT NO AUDIT RDS
028350     BEGIN-TRANSACTION COMS-IN               NO-AUDIT RDS
028400        ON EXCEPTION
028500        IF DMSTATUS(ABORT)
028600*         THEN MUST GET BACK TO RECEIVE STMT
028700          GO TO 300-ADD-ENTRY-EXIT
028800        ELSE
028900          CALL SYSTEM DMTERMINATE.
029000*    MOVE COMS-IN-RST-LOC TO RDS-LOCATOR.
029100     STORE ENTRY.
029200     IF DMSTATUS(DMERROR)
029300       MOVE "STORE ERROR" TO MSG-STATUS
029400     ELSE
029500       MOVE "BOAT ADDED" TO MSG-STATUS.
029600     END-TRANSACTION COMS-OUT AUDIT RDS
029700       ON EXCEPTION
029800       IF DMSTATUS(ABORT)
029900         GO TO 300-ADD-ENTRY-EXIT
030000       ELSE
030100         CALL SYSTEM DMTERMINATE.
030200     PERFORM 900-SEND-MSG.
030300 300-ADD-ENTRY-EXIT.
030400     EXIT.
030500
030600*****************************************************************
030700 400-DELETE-ENTRY  SECTION.
030800*****************************************************************
030900 400-DELETE-ENTRY-SN.
030920*    Delete a boat from a race.  The restart requirements are the
030940*    same as for the previous transaction.
```

Example 3–1. COMS Sample Program with a DMSII Database (cont.)

```
030960
031000     LOCK ENTRY-RACE-SET AT
031790       ENTRY-RACE-ID = MSG-DE-RACE-ID AND
031200       ENTRY-BOAT-ID = MSG-DE-ID
031300       ON EXCEPTION
031400       IF DMSTATUS(NOTFOUND) THEN
031500         MOVE "BOAT ENTRY NOT FOUND" TO MSG-STATUS
031600         GO TO DE-SEND-MSG
031700       ELSE
031800         CALL SYSTEM DMTERMINATE.
031900
032000*    BEGIN-TRANSACTION COMS-IN USING MSG-TEXT NO-AUDIT RDS
032050     BEGIN-TRANSACTION COMS-IN                NO-AUDIT RDS
032100        ON EXCEPTION
032200        IF DMSTATUS(ABORT)
032300*         THEN MUST GET BACK TO RECEIVE STMT
032400          GO TO 400-DELETE-ENTRY-EXIT
032500        ELSE
032600          CALL SYSTEM DMTERMINATE.
032700*    MOVE COMS-IN-RST-LOC TO RDS-LOCATOR.
032800     DELETE ENTRY.
032900     IF DMSTATUS(DMERROR)
033000       MOVE "FOUND BUT NOT DELETED" TO MSG-STATUS
033100     ELSE
033200       MOVE "BOAT DELETED" TO MSG-STATUS.
033300     END-TRANSACTION COMS-OUT AUDIT RDS
033400       ON EXCEPTION
033500       IF DMSTATUS(ABORT)
033600         GO TO 400-DELETE-ENTRY-EXIT
033700       ELSE
033800         CALL SYSTEM DMTERMINATE.
033900 DE-SEND-MSG.
034000     PERFORM 900-SEND-MSG.
034100 400-DELETE-ENTRY-EXIT.
034200     EXIT.
034300
034400*************************************************************************
034500 900-SEND-MSG  SECTION.
034600*************************************************************************
034700 900-SEND-MSG-SN.
034720*    Send the message back to the originating station.
034740*    Do not specify an output agenda.  Test the result
034760*    of the SEND statement.
034780*
034800     MOVE 1                TO COMS-OUT-COUNT.
034950     MOVE Ø                TO COMS-OUT-DESTINATION.
035000     MOVE Ø                TO COMS-OUT-STATUS-KEY.
035100     MOVE 106              TO COMS-OUT-TEXT-LENGTH.
035200     SEND COMS-OUT FROM MSG-TEXT.
035300     IF COMS-OUT-STATUS-KEY = Ø OR 92
035400       NEXT SENTENCE
```

Example 3–1. COMS Sample Program with a DMSII Database (cont.)

```
035500     ELSE
035600       DISPLAY "ONLINE PROGRAM SEND ERROR: " COMS-OUT-STATUS-KEY.
035700 900-SEND-MSG-EXIT.
035800     EXIT.
035900*****************************************************************
036000 910-CLOSEDOWN  SECTION.
036100*****************************************************************
036200 910-CLOSEDOWN-SN.
036220*    Close the database.
036240
036300     BEGIN-TRANSACTION NO-AUDIT RDS.
036400     RECREATE RDS AND STORE RDS.
036600     END-TRANSACTION NO-AUDIT RDS.
036700     CLOSE SAILDB.
036800 910-CLOSEDOWN-EXIT.
036900     EXIT.
037000*****************************************************************
037100 920-CHECK-COMS-INPUT-ERRORS  SECTION.
037200*****************************************************************
037300 920-CHECK-CIE-SN.
037320*    Check for COMS control messages.
037340
037400     IF ( COMS-IN-STATUS-KEY =   0 OR 92 OR 99 )
037450
037500*       These codes signify a successful  message, a recovery
037550*       message and an EOT notification, respectively.
037575*
037600        NEXT SENTENCE
037650
037700     ELSE
037800     IF COMS-IN-STATUS-KEY = 93
037900       MOVE "MSG CAUSES ABORT, PLS DONT RETRY" TO MSG-STATUS
038000       PERFORM 900-SEND-MSG
038100     ELSE
038150
038200*      The COMS control message is 20, 100, 101, or 102.  These
038300*      values mean the application is manipulating the dynamic
038400*      attachment or detachment of stations and has
038410*      received an error.
038450
038500       MOVE "ERROR IN STA ATTACH/DETACHMENT" TO MSG-STATUS
038600       PERFORM 900-SEND-MSG.
038700
038800     IF COMS-IN-FUNCTION-STATUS < 0 THEN
038850
038900*      This means that the application ID is tied to a default
039000*      input agenda.  MSG-TEXT probably does not contain a valid
039010*      transaction.
039020
039100       MOVE "NEGATIVE FUNCTION CODE " TO MSG-STATUS
039200       PERFORM 900-SEND-MSG THRU 900-SEND-MSG-EXIT.
```

Example 3–1. COMS Sample Program with a DMSII Database (cont.)

```
039300
039400 920-CHECK-CIE-EXIT.
039500     EXIT.
039600
```

Example 3–1. COMS Sample Program with a DMSII Database (cont.)

COMS Sample Program with a SIM Database

This sample program, called ONLINESAIL, tracks sailboat races and updates a SIM database by using features of the COMS direct-window interface.

COMS Features Used in the Sample Program

The following features of the COMS direct-window interface are used in the program:

- Declared COMS input and output headers
- Trancodes
- Recovery

The ONLINESAIL program runs in a COMS environment that has been configured to include a SIM database called SIMSAILDB and the following COMS entities:

- A direct window called SAIL
- An agenda called SAILAGINOL
- The following three trancodes:
  - CRERAC (MFI = 1)
  - ADDENT (MFI = 2)
  - DELENT (MFI = 3)

All entities must be defined to COMS to allow the program to run.

See Also

For more information on TADS, see the *COBOL ANSI-74 TADS Guide*.

Classes in the Database

The database SIMSAILDB contains two classes. The class RACE-CALENDAR contains one entity for every race. The class ENTRY contains one entity for each boat entered in a race. A boat can have several entities, depending on the number of races entered.

Using the Sample Program

The ONLINESAIL program is shown in Example 3–2. The program maintains the SIMSAILDB database. The RESERVE SEMANTIC clause is used in the SPECIAL NAMES paragraph, and the database is declared using the VALUE OF DBKIND IS SEMANTIC statement to identify SIMSAILDB as a semantic database.

The program contains three transactions. Each transaction has a unique trancode and a unique MFI. The trancode CRERAC creates a race entry in the database. ADDENT adds a boat entry to a race. The race must exist for the transaction to be completed. DELENT deletes a boat from a race.

All transactions instruct COMS to audit the input message at END-TRANSACTION.

The program also contains the coding required by TADS. The program must be compiled with the TADS option and the TADS diagnostic station specified. No changes need to be made to the program if TADS is not present.

The program sample contains comments to indicate the program actions at each step.

```
000100$SET TADS(REMOTE RMT)  WARNSUPR   LIST
000200 IDENTIFICATION DIVISION.
000400 PROGRAM-ID. ONLINESAIL.
000500 ENVIRONMENT DIVISION.
000600 CONFIGURATION SECTION.
000700     SOURCE-COMPUTER. MICROA.
000800     OBJECT-COMPUTER. MICROA.
000900     SPECIAL-NAMES.   RESERVE SEMANTIC.
001000 INPUT-OUTPUT SECTION.
001100 FILE-CONTROL.
001200    SELECT RMT ASSIGN TO REMOTE ACCESS SEQUENTIAL.
001300 DATA DIVISION.
001400 FILE SECTION.
001500 FD RMT.
001600 01 REM-REC                    PIC X(72).
001700 DATA-BASE SECTION.
001800     DB SIMSAILDB  VALUE OF DBKIND IS SEMANTIC.
001900*    This query description is from class RACE-CALENDAR.
002000 QD RACE.
002100    01 RACE-REC.
002200       02 RACE-NAME            PIC X(20).
002300       02 RACE-ID              PIC 9(6)  COMP.
002400       02 RACE-DATE            PIC X(6).
002500       02 RACE-TIME            PIC X(4).
002600       02 RACE-LOCATION        PIC X(20).
002700       02 RACE-SPONSOR         PIC X(10).
002800*    From class ENTRY
002900 QD ENT.
003000    01 ENTRY-REC.
003100       02 ENTRY-BOAT-NAME      PIC X(20).
003200       02 ENTRY-BOAT-ID        PIC X(6).
003300       02 ENTRY-BOAT-RATING    PIC 9(3)  COMP.
```

Example 3–2. COMS Sample Program with a SIM Database

```
003400       02 ENTRY-BOAT-HELMSMAN   PIC X(20).
003500       02 ENTRY-AFF-Y-CLUB       PIC X(15).
003600       02 ENTRY-RACE-ID          PIC 9(6)  COMP.
003700*
003800 WORKING-STORAGE SECTION.
003900 77  SCRATCH                      PIC X(256).
004000 77  NUM-KEY                      PIC 9(6)  COMP.
004100 77  E-RACE                       PIC 9(6).
004200 77  E-BOAT                       PIC 9(6).
004300*************************************************************************
004400*  MESSAGE AREA                                                         *
004500*************************************************************************
004600
004700 01  MSG-TEXT.
004800     03 MSG-TCODE                 PIC X(6).
004900     03 MSG-FILLER                PIC X.
005000     03 MSG-CREATE-RACE.
005100         05 MSG-CR-ID             PIC 9(6).
005200         05 MSG-CR-NAME           PIC X(20).
005300         05 MSG-CR-DATE           PIC X(6).
005400         05 MSG-CR-TIME           PIC X(4).
005500         05 MSG-CR-LOCATION       PIC X(20).
005600         05 MSG-CR-SPONSOR        PIC X(10).
005700         05 FILLER                PIC X(10).
005800     03 MSG-ADD-ENTRY REDEFINES MSG-CREATE-RACE.
005900         05 MSG-AE-RACE-ID        PIC 9(6).
006000         05 MSG-AE-ID             PIC 9(6).
006100         05 MSG-AE-NAME           PIC X(20).
006200         05 MSG-AE-RATING         PIC 9(3).
006300         05 MSG-AE-HELMSMAN       PIC X(20).
006400         05 MSG-AE-CLUB           PIC X(15).
006500         05 FILLER                PIC X(6).
006600     03 MSG-DELETE-ENTRY REDEFINES MSG-CREATE-RACE.
006700         05 MSG-DE-RACE-ID        PIC 9(6).
006800         05 MSG-DE-ID             PIC 9(6).
006900         05 FILLER                PIC X(64).
007000     03 MSG-STATUS                PIC X(40).
007100
007200  01 WS-FAMILY                    PIC X(40).
007300  01 WS-MSG.
007400     03  MSG-1                    PIC X(78).
007500     03  MSG-2                    PIC X(98).
007600*************************************************************************
007625*  COMS INTERFACE DECLARATIONS                                          *
007650*************************************************************************
007700 COMMUNICATION SECTION.
007800 INPUT HEADER COMS-IN;
007900   PROGRAMDESG            IS COMS-IN-PROGRAM;
008000   FUNCTIONSTATUS         IS COMS-IN-FUNCTION-STATUS;
008100   FUNCTIONINDEX          IS COMS-IN-FUNCTION-INDEX;
008200   USERCODE               IS COMS-IN-USERCODE;
```

Example 3–2. COMS Sample Program with a SIM Database (cont.)

```
008300   SECURITYDESG           IS COMS-IN-SECURITY-DESG;
008400   TRANSPARENT            IS COMS-IN-TRANSPARENT;
008500   VTFLAG                 IS CONS-IN-VT-FLAG;
008600   TIMESTAMP              IS COMS-IN-TIMESTAMP;
008700   STATION                IS COMS-IN-STATION;
008800   TEXTLENGTH             IS COMS-IN-TEXT-LENGTH;
008900   STATUSVALUE            IS COMS-IN-STATUS-KEY;
009000   MESSAGECOUNT           IS COMS-IN-MSG-COUNT;
009100   RESTART                IS COMS-IN-RST-LOC;
009200   AGENDA                 IS COMS-IN-AGENDA;
009400
009500 OUTPUT HEADER COMS-OUT;
009600   DESTCOUNT              IS COMS-OUT-COUNT;
009700   TEXTLENGTH             IS COMS-OUT-TEXT-LENGTH;
009800   STATUSVALUE            IS COMS-OUT-STATUS-KEY;
009900   TRANSPARENT            IS COMS-OUT-TRANSPARENT;
010000   VTFLAG                 IS COMS-OUT-VT-FLAG;
010100   CONFIRMFLAG            IS COMS-OUT-CONFIRM-FLAG;
010200   CONFIRMKEY             IS COMS-OUT-CONFIRM-KEY;
010300   DESTINATIONDESG        IS COMS-OUT-DESTINATION;
010400   NEXTINPUTAGENDA        IS COMS-OUT-NEXT-INPUT-AGENDA;
010500   SETNEXTINPUTAGENDA     IS COMS-OUT-SET-NEXT-INPUT-AGENDA;
010600   RETAINTRANSACTIONMODE  IS COMS-OUT-SAVE-TRANS-MODE;
010700   AGENDA                 IS COMS-OUT-AGENDA;
010900*****************************************************************
011000 PROCEDURE DIVISION.
011100*****************************************************************
011200
011300 005-MAIN SECTION.
011400 005-MAIN-SN.
011500     CHANGE ATTRIBUTE LIBACCESS OF "DCILIBRARY" TO BYINITIATOR.
011800
011900     OPEN UPDATE SIMSAILDB.
012000     IF DMSTATE THEN
012100        CALL SYSTEM DMEXCEPTIONMSG GIVING SCRATCH
012200        DISPLAY SCRATCH
012300        STOP RUN.
012400
013500     ENABLE INPUT COMS-IN KEY "ONLINE".
013700     PERFORM 007-PROCESS-COMS-INPUT
013800       UNTIL COMS-IN-STATUS-KEY = 99.
013900 005-MAIN-EXIT.
014000     PERFORM 910-CLOSEDOWN.
014100     STOP RUN.
014200
014300*****************************************************************
014400 007-PROCESS-COMS-INPUT SECTION.
014500*****************************************************************
014600*    Get the next message from COMS. If the message is 99, go to
```

Example 3–2. COMS Sample Program with a SIM Database (cont.)

```
014625*    end of task (EOT); otherwise, make sure that the message
014700*    is valid before processing it.
014800
014900 007-PROCESS-CI-SN.
015000     MOVE SPACES TO MSG-TEXT.
015100     RECEIVE COMS-IN MESSAGE INTO MSG-TEXT.
015200     IF COMS-IN-STATUS-KEY NOT = 99
015300       PERFORM 920-CHECK-COMS-INPUT-ERRORS
015400       IF ( COMS-IN-STATUS-KEY =   0 OR 92)  AND
015500            COMS-IN-FUNCTION-STATUS NOT < 0 THEN
015600         PERFORM 100-PROCESS-TRANSACTION.
015700 007-PROCESS-CI-EXIT.
015800     EXIT.
015900
016000*************************************************************************
016100 100-PROCESS-TRANSACTION  SECTION.
016200*************************************************************************
016300*    Because the transaction type is programmatically based
016400*    on MFI, make sure that it is within range before
016450*    exiting to a subprogram.
016500
016600 100-PROCESS-TRANS-SN.
016700     MOVE 0 TO E-RACE,  E-BOAT.
016800     IF MSG-TCODE = "CRERAC"
016900       PERFORM 200-CREATE-RACE
017000     ELSE
017100     IF MSG-TCODE = "ADDENT"
017200       PERFORM 300-ADD-ENTRY
017300     ELSE
017400     IF MSG-TCODE = "DELENT"
017500       PERFORM 400-DELETE-ENTRY
017600     ELSE
017700       MOVE "INVALID TRANS CODE" TO MSG-STATUS
017800       PERFORM 900-SEND-MSG.
017900
018000 100-PROCESS-TRANS-EXIT.
018100     EXIT.
018200
018300*************************************************************************
018400 200-CREATE-RACE  SECTION.
018500*************************************************************************
018600 200-CREATE-RACE-SN.
018700*    Enter a new race in the database.
018800
018900     MOVE MSG-CR-ID TO E-RACE.
019000     IF MSG-CR-ID NOT NUMERIC
019100       MOVE "Race # must be numeric" TO MSG-STATUS
019200       GO TO CRE-SEND-MSG.
019300     BEGIN-TRANSACTION
019400       ON EXCEPTION
019500          PERFORM 905-SIM-ERR-RTN
```

Example 3–2. COMS Sample Program with a SIM Database (cont.)

```
019600          GO TO CRE-SEND-MSG.
019700      INSERT RACE-CALENDAR
019800        ASSIGN MSG-CR-NAME      TO RACE-NAME
019900        ASSIGN MSG-CR-ID        TO RACE-ID
020000        ASSIGN MSG-CR-DATE      TO RACE-DATE
020100        ASSIGN MSG-CR-TIME      TO RACE-TIME
020200        ASSIGN MSG-CR-LOCATION  TO RACE-LOCATION
020300        ASSIGN MSG-CR-SPONSOR   TO RACE-SPONSOR
020400          ON EXCEPTION
020500            PERFORM 905-SIM-ERR-RTN.
020550
021100      IF DMSTATE
021200        MOVE "Store Error" TO MSG-STATUS
021300      ELSE
021400        MOVE "Race added " TO MSG-STATUS.
021500
021600      END-TRANSACTION COMS-OUT
021700        ON EXCEPTION
021800          PERFORM 905-SIM-ERR-RTN.
021900 CRE-SEND-MSG.
022000      PERFORM 900-SEND-MSG.
022100 200-CREATE-RACE-EXIT.
022200      EXIT.
022300
022400*****************************************************************
022500 300-ADD-ENTRY   SECTION.
022600*****************************************************************
022700 300-ADD-ENTRY-SN.
022800*    To enter a boat in a race, determine if the race exists.
022900*    If not, do not add a boat.  A DM error in this section of
023000*    code indicates a duplicate entry.  Note that the
023100*    key is numeric and must be defined and compared.
023200
023300      IF MSG-AE-RACE-ID NOT NUMERIC OR
023400         MSG-AE-ID      NOT NUMERIC
023500           MOVE "Keys must be numeric" TO MSG-STATUS
023600           GO TO ADD-SEND-MSG.
023700      MOVE MSG-AE-RACE-ID TO E-RACE.
023800      MOVE MSG-AE-ID      TO E-BOAT.
023900      MOVE MSG-AE-RACE-ID TO NUM-KEY.
024000      SELECT RACE FROM RACE-CALENDAR
024100        WHERE RACE-ID = NUM-KEY.
024200      RETRIEVE RACE
024300        ON EXCEPTION
024400          MOVE "Race not found" TO MSG-STATUS
024500          DISCARD RACE
024600          GO TO ADD-SEND-MSG.
024700      DISCARD RACE.
024800      BEGIN-TRANSACTION
024900        ON EXCEPTION
025000           PERFORM 905-SIM-ERR-RTN
```

Example 3–2. COMS Sample Program with a SIM Database (cont.)

```
025100          GO TO ADD-SEND-MSG.
025200      INSERT ENTRY
025300        ASSIGN MSG-AE-NAME       TO ENTRY-BOAT-NAME
025400        ASSIGN MSG-AE-ID         TO ENTRY-BOAT-ID
025500        ASSIGN MSG-AE-RATING     TO ENTRY-BOAT-RATING
025600        ASSIGN MSG-AE-HELMSMAN   TO ENTRY-BOAT-HELMSMAN
025700        ASSIGN MSG-AE-CLUB       TO ENTRY-AFF-Y-CLUB
025800        ASSIGN MSG-AE-RACE-ID    TO ENTRY-RACE-ID
025900          ON EXCEPTION
026000            PERFORM 905-SIM-ERR-RTN.
026100*  Do not leave the transaction state.  An END-TRANSACTION
026150*  statement must be processed.
026200      IF DMSTATE
026300        MOVE "Store Error" TO MSG-STATUS
026400      ELSE
026500        MOVE "Boat added" TO MSG-STATUS.
026600      END-TRANSACTION COMS-OUT
026700        ON EXCEPTION
026800          PERFORM 905-SIM-ERR-RTN.
026900 ADD-SEND-MSG.
027000      PERFORM 900-SEND-MSG.
027100 300-ADD-ENTRY-EXIT.
027200      EXIT.
027300
027400*****************************************************************
027500 400-DELETE-ENTRY  SECTION.
027600*****************************************************************
027700 400-DELETE-ENTRY-SN.
027800* Delete a boat from a race. SIM always returns a positive
027900* result; therefore, check to see if the boat exists.
027950* If it does, delete it.
028000
028100      IF MSG-DE-RACE-ID NOT NUMERIC OR
028200         MSG-DE-ID      NOT NUMERIC
028300           MOVE "Keys must be numeric" TO MSG-STATUS
028400           GO TO DE-SEND-MSG.
028500      MOVE MSG-DE-RACE-ID TO NUM-KEY.
028600      MOVE MSG-DE-RACE-ID TO E-RACE.
028700      MOVE MSG-DE-ID      TO E-BOAT.
028800      SELECT ENT  FROM ENTRY
028900        WHERE ENTRY-RACE-ID = NUM-KEY AND
029000              ENTRY-BOAT-ID = MSG-DE-ID.
029100      RETRIEVE ENT
029200        ON EXCEPTION
029300          MOVE "Boat not found" TO MSG-STATUS
029400          DISCARD ENT
029500          GO TO DE-SEND-MSG.
029600      DISCARD ENT.
029700      BEGIN-TRANSACTION
029800        ON EXCEPTION
029900           PERFORM 905-SIM-ERR-RTN
```

Example 3–2. COMS Sample Program with a SIM Database (cont.)

```
030000          GO TO DE-SEND-MSG.
030100     DELETE ENTRY WHERE ENTRY-RACE-ID = NUM-KEY AND
030200                        ENTRY-BOAT-ID = MSG-DE-ID
030300       ON EXCEPTION
030400            PERFORM 905-SIM-ERR-RTN.
030500     IF DMSTATE
030600       MOVE "Delete Error" TO MSG-STATUS
030700     ELSE
030800       MOVE "Boat deleted" TO MSG-STATUS.
030900     END-TRANSACTION COMS-OUT
031000       ON EXCEPTION
031100         PERFORM 905-SIM-ERR-RTN.
031200 DE-SEND-MSG.
031300     PERFORM 900-SEND-MSG.
031400 400-DELETE-ENTRY-EXIT.
031500     EXIT.
031600
031700*****************************************************************
031800 900-SEND-MSG  SECTION.
031900*****************************************************************
032000 900-SEND-MSG-SN.
032100*    Send the message back to the originating station.
032150*    Do not specify an output agenda.
032200*    Test the result of the SEND statement.
032300
032400     MOVE 1               TO COMS-OUT-COUNT.
032500     MOVE 0               TO COMS-OUT-DESTINATION.
032600     MOVE 0               TO COMS-OUT-STATUS-KEY.
032700     MOVE 116             TO COMS-OUT-TEXT-LENGTH.
032800     SEND COMS-OUT FROM MSG-TEXT.
032900     IF COMS-OUT-STATUS-KEY = 0 OR 92
033000       NEXT SENTENCE
033100     ELSE
033200       DISPLAY "On-line program SEND error: " COMS-OUT-STATUS-KEY.
033300 900-SEND-MSG-EXIT.
033400     EXIT.
033500*****************************************************************
033600 905-SIM-ERR-RTN  SECTION.
033700*****************************************************************
033800 905-SIM-ERR-RTN-SN.
033900*   Get the error message from SIM.  The message can be
033950*   176 bytes in length.
033955*
034000     DISPLAY "SIM Error: Race=", E-RACE, " Boat=", E-BOAT.
034100     CALL SYSTEM DMEXCEPTIONMSG GIVING WS-MSG.
034200     DISPLAY MSG-1.
034300     DISPLAY MSG-2.
034400 905-SIM-ERR-RTN-EXIT.
034500     EXIT.
```

Example 3–2. COMS Sample Program with a SIM Database (cont.)

```
034600*****************************************************************
034700 910-CLOSEDOWN  SECTION.
034800*****************************************************************
034900 910-CLOSEDOWN-SN.
035000*    Close the database.
035100     CLOSE SIMSAILDB.
035200 910-CLOSEDOWN-EXIT.
035300     EXIT.
035400*****************************************************************
035500 920-CHECK-COMS-INPUT-ERRORS  SECTION.
035600*****************************************************************
035700 920-CHECK-CIE-SN.
035800*    Check for COMS control messages.
035900
036200     IF ( COMS-IN-STATUS-KEY =   0 OR 92 OR 99 )
036250*       The codes signify a successful  message, a recovery
036300*       message, or an EOT notification, respectively.
036350*
036400         NEXT SENTENCE
036500     ELSE
036600     IF COMS-IN-STATUS-KEY = 93
036700       MOVE "93: MSG CAUSED ABORT, DONT RETRY" TO MSG-STATUS
036800       PERFORM 900-SEND-MSG
036900     ELSE
036950*      COMS-IN STATUS-KEY is 20, 100, 101, or 102. These
037000*      values mean the application is manipulating the
037100*      dynamic attachment or detachment of stations, and has
037200*      received an error.
037300       DISPLAY COMS-IN-STATUS-KEY
037400       MOVE "ERROR IN STA ATTACH/DETACHMENT" TO MSG-STATUS
037500       PERFORM 900-SEND-MSG.
037600
037700     IF COMS-IN-FUNCTION-STATUS < 0 THEN
037800*      This statement means that the application ID is tied
037900*      to a default input agenda.  MSG-TEXT probably does not
037950*      contain a valid transaction.
038000       MOVE "NEGATIVE FUNCTION CODE " TO MSG-STATUS
038100       DISPLAY COMS-IN-FUNCTION-STATUS
038200       PERFORM 900-SEND-MSG-SN THRU 900-SEND-MSG-EXIT.
038300
038400 920-CHECK-CIE-EXIT.
038500     EXIT.
038600
```

Example 3–2. COMS Sample Program with a SIM Database (cont.)

Section 4
Using the DMSII Program Interface

Data Management System II (DMSII) is used to invoke a database and maintain relationships between the various data elements in the database.

This section explains how to use the extensions developed for the DMSII application program interface. The DMSII extensions allow you to perform the following tasks:

- Identify, qualify, and reference database items.
- Declare and invoke a database.
- Invoke data sets.
- Use database equation operations to specify and manipulate database titles, and to override compiled titles.
- Use selection expressions to identify a particular record in a data set.
- Use data management attributes for read-only access to the count, record type, and population information in a record.
- Use the DATADICTINFO option to place database and transaction base usage data into the object code file.
- Manipulate data through data management statements.
- Process exceptions.

For an alphabetized list of the extensions developed for DMSII, refer to Section 1, "Introduction to COBOL74 Program Interfaces." Refer to the *A Series DMSII Application Program Interfaces Programming Guide* for information on general programming considerations and concepts.

The DMSII program interface can be used with the Advanced Data Dictionary System (ADDS), Communications Management System (COMS), DMSII transaction processing system (TPS), and the Semantic Information Manager (SIM). For more information on the extensions developed for one of these products, see the section that provides an explanation of that product.

See Also

- For additional information on using ADDS with DMSII, see the *InfoExec ADDS Operations Guide.*
- For additional information on using COMS with DMSII, see the *COMS Programming Guide.*
- For additional information on using SIM with DMSII, see the *A Series InfoExec Semantic Information Manager (SIM) Programming Guide.*

Using Database Items

This discussion describes the naming conventions for database items and explains how to reference the items. A data record from a database is accessed directly by a COBOL74 program.

Naming Database Components

Data and Structure Definition Language (DASDL) naming conventions for database components follow COBOL74 rules; that is, some item or structure names can require qualification and some can contain hyphens (-). Whenever syntax specifies the names of database components, these names can be fully qualified names and can contain hyphens.

Using Set and Data Set Names

You must qualify set and data set names that are used to find records if the names are not unique. You can declare a variable name with the same name as a database item if the item can be qualified.

If you invoke a data set more than once and that data set uses internal names to invoke two paths, you must qualify each path by the data set that it spans. If you use improper or insufficient qualification, you receive a syntax error.

You qualify an item name by the name of any structure that physically contains the item. You can use any number of qualifying names as long as the result is unique.

A set name can be qualified by the name of the data set it spans. A group name can qualify an item contained within the group.

Note that the compiler usually issues a syntax error when qualification is logically required but not provided. However, unexpected results can still occur, even when the compiler issues a warning. If qualification is insufficient, the compiler usually uses the last item invoked in the database that has the same name as the variable. To avoid unexpected results, do one or both of the following when invoking DMSII data sets more than once in the same program:

- Use full qualification on all references.
- Invoke all data sets with INVOKE clauses. In addition, use the USING and INVOKE clauses to explicitly invoke all sets associated with each data set.

General Format

The general format of the statement to qualify a name is as follows:

```
identifier-1[OF identifier-2]...
```

Explanation of Format Elements

identifier-1 and identifier-2

These format elements specify DASDL identifiers.

Examples

The following examples show code in which name qualification is needed or in which a successful or unsuccessful attempt has been made to provide qualification. The applicable DASDL descriptions precede examples.

Example of Qualifying Valid and Invalid Names

The following DASDL description is used by the COBOL74 code in the following example:

```
DASDL (compiled as DBASE):
    D1 DATA SET (
       XYZ NUMBER (5););
```

In the following example, XYZ declared at level 77 is invalid because it cannot be qualified. However, XYZ declared at level 03 is valid because it can be qualified.

```
*Invalid DATA DIVISION entry:
 77 XYZ  PIC . . .

*Valid DATA DIVISION entry:
 Ø1 Q.
  Ø3 XYZ . . .
```

Example of Using Names That Require Qualification

The following DASDL description is used by the COBOL74 code in the following example:

```
DASDL (compiled as DBASE):
D1 DATA SET (
   A NUMBER (5);
   B . . .
   .
   .
   .        );
S1 SET OF D1 KEY A;
```

In the following example, *S1* and *A* require qualification:

```
DB DBASE.
Ø1 D1.
Ø1 DA=D1.

FIND S1 OF D1 AT A=V.
MOVE A OF D1 TO LA.
```

Referencing Database Items

You can invoke all or part of a database in the DATA-BASE SECTION of your program. When the description is invoked, the compiler generates the interfaces needed to allocate the proper record areas when the database is opened.

The record area for a data set is established in two parts: one part contains control items, and the other contains data items. You code the data items just as an 01-level WORKING-STORAGE SECTION entry, with appropriate lower level entries. This setup enables you to use the data manipulation statements to move database items, including groups.

Group moves are always considered alphanumeric moves. The arrangement of the data item record area also enables you to use MOVE CORRESPONDING statements. For more information on MOVE statements, refer to Volume 1.

If you use variable-format records in your programs, a group move at the 01-level fills the receiving area without regard to the individual items contained within either the sending or receiving area. Using variable-format records can therefore cause unexpected values to be stored in the receiving area. For MOVE CORRESPONDING statements, only items in the fixed portion of the record are candidates for the move.

Examples

The following examples reference database items that contain compiler-produced listings.

Example of Group Naming

The following example illustrates a group move involving database items. The items T, CT, L, E, and S are control items and are not affected by moves to or from D.

The record area for D is the following:

```
Ø1  D
    Ø2  A
    Ø2  B
    Ø2  C
        Ø3 E1
        Ø3 E2.
```

E1 and E2 are items of the record area for E rather than D; therefore, these items are not affected by moves to or from D.

```
Ø1  D  DATA SET (#1).
    Ø2  T RECORDTYPE.
    Ø2  CT COUNT.
    Ø2  A PIC X(6) DISPLAY.
    Ø2  B PIC 9(6) COMP.
    Ø2  C PIC 9(6) COMP.
    Ø2  L REFERENCE TO E.
    Ø2  E DATA SET (#2).
        Ø3  E1 . . .
        Ø3  E2 . . .
    Ø2  S SET(#4,MANUAL) OF E.
```

Example of Receiving Fields of a MOVE CORRESPONDING Statement

The following example shows database items that will be used in a MOVE CORRESPONDING statement. The items contained in the record depend on the value of T as follows:

- If T equals 0, then the record area contains T, A, and B.
- If T equals 1, then the record area contains T, A, B, and X.
- If T equals 2, then the record area contains T, A, B, and Y.

In this example, because A and B are in the fixed portions of the record, these items are the only candidates for a MOVE CORRESPONDING statement on D.

The items X and Y are never moved as a result of a MOVE CORRESPONDING statement.

```
 Ø1  D
     Ø2  T RECORDTYPE.
     Ø2  A PIC X(6) DISPLAY.
     Ø2  B PIC 9(6) COMP.
* FORMAT TYPE 1.
     Ø2  X PIC 9(6) COMP.
* FORMAT TYPE 2.
     Ø2  Y PIC 9(11) COMP.
```

Example of Creating Valid and Invalid DMSII Indexes

The following DASDL description is used by the COBOL74 code in the following example:

```
DASDL:
 C  COMPACT DATASET
   (N    NUMBER (1);
    X    NUMBER (5) OCCURS 9 TIMES DEPENDING ON N);
```

The following example shows three lines of COBOL74 code that use MOVE statements. In the statements, the variable N in the DASDL description equals 5. The first and second lines execute successfully. The third line, however, creates an invalid index error because the program attempts to access an occurrence of an OCCURS DEPENDING ON item that is larger than the current value of the DEPENDING ON item.

```
MOVE 123 to X(3).

MOVE 5 TO N.

MOVE 456 to X(7).
```

Declaring a Database

The DATA-BASE SECTION of a COBOL74 program supplies the COBOL74 compiler with a description of all or selected portions of one or more databases. You place the DATA-BASE SECTION in the DATA DIVISION after the FILE SECTION and before the WORKING-STORAGE SECTION.

The database declaration supplies information that identifies a given database. The compiler lists all the invoked descriptions of record formats, items, sets, subsets, and keys.

General Format

The general format for a database declaration is as follows:

```
DB [data-name-1 {INVOKE} ] [data-name-2 OF] data-name-3
                {=     }
[GLOBAL]
[ALL]
[VALUE OF TITLE IS literal-1]
```

Explanation of Format Elements

data-name-1

This format element specifies the internal name of the database, data sets, sets, or subsets within the program.

You can invoke a database, data set, set, or subset more than once; however, you must use the external name to reference only one invocation of each structure. Data-name-1 must be used to provide unique names for all other invocations of the structures. The default internal name is the external name of the structure.

You can establish multiple record areas and set paths, or both, by specifying data-name-1 with a data set reference or set reference. In this way, several records of a single data set can be manipulated simultaneously.

INVOKE or the equal sign (=)

These format elements invoke a database, data set, set or subset. The INVOKE option and the equal sign (=) are synonyms.

When you use the INVOKE option in your program, you must use the internal names of renamed structures in all subsequent references to the structures. A structure using an alias identifier can be a string literal enclosed in quotation marks ("").

data-name-2

This format element enables the program to reference a logical database. A program can invoke structures selectively from a logical database, or it can invoke the entire logical database. You can specify selective invocations such as physical databases; however, you can select only those structures that are included in the logical database.

data-name-3

This format element designates the name of the database to be invoked. You can use data-name-3 as a qualifier of a data set, set, or subset. If you use the INVOKE clause, data-name-3 can be a string literal enclosed in quotation marks ("").

GLOBAL

This option declares a database to be global, enabling a separately compiled procedure or program to reference the database declared in your COBOL74 program. The database reference in the separately compiled procedure or program must match the database reference in your COBOL74 program.

ALL

This option specifies that all structures of the specified database are to be used.

VALUE OF

This option specifies the file attribute name. The only mnemonic allowed in the VALUE OF option is TITLE. For more information about the VALUE OF clause, refer to Volume 1.

literal-1

This format element designates the name of the file attribute. Literal-1 can be a string literal enclosed in quotation marks ("").

Considerations for Use

For a database, the operating system constructs the control file title from the title specified in the declaration. The default title is that title plus the control file usercode and family name, if any, from the description file. Refer to the *DMSII DASDL Reference Manual* for a discussion of control and description files.

Examples

The following examples show varied formats you can use to declare a database:

```
DB DATABASE-1.

DB MY-DB-2 = DATABASE-2.

DB DATABASE-3 GLOBAL.

DB DATABASE-4 VALUE OF TITLE IS "(XYZ)DATABASE/4".

DB DATABASE-5 ALL.

DB DATABASE-6.
Ø1 DATASET-1.
Ø1 DATASET-2.
```

Invoking Data Sets

The data set reference specifies the structures that are to be invoked from the declared database. If you do not specify a particular data set to be invoked, all structures are invoked implicitly. The data set reference must be written at the 01-level.

The maximum number of structures that can be invoked by a user program is 4000. Only 1000 of the invoked structures can be data sets or remaps.

General Format

The general format for the data set reference is as follows:

```
01 [ data-name-1 { INVOKE } ]
                 { =      }
{ data-name-2 [ USING { NONE                                          } ] }
{             [       { [ data-name-3 { INVOKE } ] data-name-4... } ]   }
{ data-name-5                         { =      }                        }
```

Explanation of Format Elements

data-name-1

This format element designates the internal name of the data set or set.

INVOKE or the equal sign (=)

These format elements invoke only the data sets that you specify if you use the data set reference. If you do not use the data set reference, you implicitly invoke all structures of the specified database.

You cannot invoke only the global data from a database; you must invoke at least one structure. You can invoke the structure explicitly in the data set reference, or implicitly by default or by using the ALL option in the database statement.

INVOKE and the equal sign (=) are synonyms.

data-name-2

This format element must be the name of the data set to be used. You can use data-name-2 as a qualifier for its embedded structures. If the INVOKE option is used, data-name-2 can be a string literal enclosed in quotation marks ("").

USING

This option invokes specific sets from the data sets declared in the data set reference. If you omit the USING option, you invoke all sets. If you specify the USING option, you invoke either no sets (NONE) or only the specified sets.

data-name-3

This format element contains the name of a set. You can use data-name-3 in your program to reference a synonym for the set specified by data-name-4. All subsequent references to the set previously specified as data-name-4 must use the name specified for data-name-3.

data-name-4

This format element contains the name of a set.

data-name-5

This format element is a set reference that is not implicitly associated with any particular record area. When you choose data-name-5, data-name-1 must be the name of a set. You must specify the data set name VIA option in the selection expression to load a record area by using data-name-5. Additional information on the VIA option is provided in "Using Selection Expressions" later in this section.

Considerations for Use

You can explicitly invoke only disjoint structures. Embedded data sets, sets, and subsets are always implicitly invoked if their master data sets are (implicitly or explicitly) invoked. You must reference all implicitly invoked structures by their external names.

To use a path, you must invoke the disjoint data set. You establish a path by invoking a data set containing either of the following:

- An embedded set associated with a disjoint data set
- A link to another disjoint data set

Multiple invocations of a structure provide multiple record areas, set paths, or both, so that several records of a single data set can be manipulated simultaneously. Select only needed structures for the UPDATE and INQUIRY options to make better use of system resources. An explanation of these two options is provided in "OPEN Statement" later in this section.

You invoke remaps declared in DASDL the same way as you invoke conventional data sets.

Examples of Invoking Data Sets

The following DASDL description applies to the following six examples that show COBOL74 code that invokes a data set:

```
DASDL (compiled as DB):
D   DATA SET (
      K   NUMBER (6);
      R   NUMBER (5);
      );
S1  SET OF D KEY K;
S2  SET OF D KEY R;
```

Example of Establishing One Record Area and Two Paths

The following example shows the establishment of one current record area for the data set D, one path for the set S1, and one path for the set S2. If you execute either of the following statements, the program automatically loads the data to the D record area: FIND S1 , MODIFY S1 , FIND S2 , or MODIFY S2. S1 and S2 are invoked implicitly.

```
Ø1 D.
```

Example of Establishing Two Record Areas and Two Paths

The following example shows the establishment of two current record areas (D and X) and two paths (S1 and S2). The sets S1 and S2 are implicitly associated with the D record area. The USING NONE option prevents a set from being associated with the X record area. Thus, the FIND S1 or FIND S2 statement loads the D record area. The FIND X VIA S1 or FIND X VIA S2 statement must be executed to load the X record area using a set. S1 and S2 are invoked implicitly.

```
Ø1 D.

Ø1 X=D USING NONE.
```

Example of Establishing Multiple Record Areas and Paths

The following example shows how multiple current record areas and multiple current paths can be established. The FIND S1 OF D statement loads the D record area without disturbing the path S1 OF X, and the FIND S1 OF X statement loads the X record area without disturbing the path S1 OF D. The S1 set must be qualified. S1 and S2 are invoked implicitly.

```
Ø1 D.

Ø1 X=D.
```

Example of Establishing More Record Areas Than Paths

The following example shows how to establish more current record areas than paths. The three record areas (D, X, and Y) are established with only two paths (S1 OF D and S1 OF X). The program must execute the following statements to load the Y record area: FIND Y VIA S1 OF D, FIND Y VIA S1 OF X, or FIND Y. S1 is invoked explicitly.

```
Ø1 D   USING S1.

Ø1 X=D USING S1.

Ø1 Y=D USING NONE.
```

Example of Associating a Set with a Record Area

The following example shows how the USING clause syntax can be used to explicitly associate a set with a given record area. The FIND S1 statement loads the X record area, and the FIND T statement loads the Y record area. The S1 and T sets both use the same key.

```
Ø1 X=D USING S1.

Ø1 Y=D USING T=S1.
```

Example of Establishing a Set That Is Not Associated with a Record Area

The following example shows how the set reference can be used to establish a set that is not implicitly associated with any particular record area. The FIND D VIA SY statement must be executed to load a record area using the set S1.

```
Ø1 D.

Ø1 SY=S1.
```

Examples of Invoking Disjoint Data Sets with a Data Set Reference

The following DASDL description applies to the following two examples that use data set references to invoke disjoint data sets:

```
DASDL (compiled as DBASE):
F DATA SET (
  FI NUMBER (4);
  )
E DATA SET (
  EK NUMBER (8);
  )
D DATA SET (
  A NUMBER (6);
  SE SET OF E KEY EK;
  LINK REFERENCE TO F;
  );
```

In the following example, the data set references are not specified to invoke the E and F data sets; however, the paths are established by invoking the embedded set SE and the link item LINK:

```
Ø1 D.
```

The paths established in the preceding example, however, cannot be used unless they are specified as data set references for the E and F data sets to establish record areas for these paths, as shown in the following lines of code:

```
Ø1 F.
Ø1 E.
Ø1 D.
```

Example of Designating Sets That Are Visible or Invisible to User Programs

The following DASDL description applies to the following example that shows how DASDL can be used to designate sets visible or invisible in user programs:

```
DASDL (compiled as EXAMPLEDB):
D1 DATA SET (
   A  REAL;
   B  NUMBER (5);
   C  ALPHA (10);
   );
S1A SET OF D1 KEY IS A;
S1B SET OF D1 KEY IS (A,B,C);
D2 DATA SET (
   X  FIELD (8);
   Y  NUMBER (2);
   Z  REAL;
   E  DATA SET (
      V1  REAL;
      V2  ALPHA (2);
      );
   SE SET OF E KEY IS V1;
   );
S2A SET OF D2 KEY IS X;
S2B SET OF D2 KEY IS (X,Y,Z);
LDB1 DATABASE (D1(NONE), D2(SET S=S2A));
LDB2 DATABASE (D1(SET S1=S1B), D2(SET S2=S2B));
LDB3 DATABASE (D=D2);
```

The following example shows the commented code as it appears in the listing. For logical database LDB2, the following sets are visible to the user program:

- Data set D1 and its set S1B (referenced as S1)
- Data set D2 and its set S2B (referenced as S2)

Sets S1A and S2A are invisible to the user program because logical databases LDB1 and LDB3 are not declared in the DATA-BASE SECTION of the COBOL74 program.

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID.  DBTEST.
 DATA DIVISION.
 DATA-BASE SECTION.
 DB LDB2 OF EXAMPLEDB ALL.
*  Ø1 D1 STANDARD DATA SET(#2).
*        S1 SET(#4,AUTO) OF D1 KEYS ARE A,B,C.
*     Ø2 A REAL.
*     Ø2 B PIC 9(5) COMP.
*     Ø2 C PIC X(1Ø) DISPLAY.
*  Ø1 D2 STANDARD DATA SET(#5).
*        S2 SET(#9,AUTO) OF D2 KEYS ARE X,Y,Z.
*     Ø2 X FIELD SIZE IS Ø8 BITS.
*     Ø2 Y PIC 99 COMP.
*     Ø2 Z REAL.
*     Ø2 E STANDARD DATA SET(#6).
*           SE SET(#7,AUTO) OF E KEY IS V1.
*        Ø3 V1 REAL.
*        Ø3 V2 PIC XX DISPLAY.
 PROCEDURE DIVISION.
 T.
    STOP RUN.
```

Examples of Using the GLOBAL Option to Reference a Database

The separately compiled procedure SEP/P uses the GLOBAL option to reference the database declared in a COBOL74 program. The database reference in the separately compiled procedure or program must match the corresponding database reference in the COBOL74 program; otherwise an error occurs at binding.

The following DASDL description is used by the COBOL74 code in the following two examples:

```
DASDL (compiled as TESTDB):
DS DATA SET (
   NAME GROUP (
        LAST   ALPHA (1Ø);
        FIRST  ALPHA (1Ø);
        );
   AGE   NUMBER (2);
   SEX   ALPHA (1);
   SSNO  ALPHA (9);
   );
 NAMESET SET OF DS KEY (LAST, FIRST);
```

Using a Separately Compiled Procedure to Reference a Database with the GLOBAL Option

The following example shows the SEP/P procedure:

```
 $ LEVEL=3
 .
 .
 .
 DATA-BASE SECTION.
 DB TESTDB GLOBAL ALL.
 PROCEDURE DIVISION.
 P1.
    SET NAMESET TO BEGINNING.
    SET NAMESET TO BEGINNING.
 PERFORM P2 UNTIL DMSTATUS (DMERROR) = 1.
 P2.
 FIND NEXT NAMESET AT LAST = "SMITH" AND FIRST = "JOHN".
*   OTHER STATEMENTS
```

The following Work Flow Language (WFL) program binds the SEP/HOST procedure with the COBOL74 program. Refer to the *A Series Work Flow Language (WFL) Programming Reference Manual* for more information on using WFL.

```
?BEGIN JOB BIND/GLOB;
 BIND GLOBDB WITH BINDER LIBRARY;
 BINDER DATA CARD
 HOST IS SEP/HOST;
 BIND P FROM SEP/P;
?END JOB.
```

Using a DMSII Host Program to Call a Separately Compiled Procedure That References the Same Database

The following example shows the COBOL74 program declarations for the DMSII host program and the corresponding database reference. The program is compiled as SEP/HOST.

```
.
.
DATA-BASE SECTION.
DB TESTDB ALL.
PROCEDURE DIVISION.
DECLARATIVES.
P SECTION.  USE EXTERNAL AS PROCEDURE.
END DECLARATIVES.
P1.
   OPEN UPDATE TESTDB.
   CALL P.
   CLOSE TESTDB.
   STOP RUN.
```

Using a Database Equation Operation

Database equation refers to three separate operations:

- Specification of database titles during compilation
- Use of a Work Flow Language (WFL) equation to override titles specified at compilation time
- Run-time manipulation of database titles

The specification of the title of a database at run time enables the Accessroutines to use the reentrance capability. A database equation enables the database to be specified at run time, and enables access to databases stored under other usercodes and on pack families not visible to a task.

A database equation is like a file equation. A WFL equation overrides the specification of a database title in a language declaration, and run-time modification of a database title overrides both WFL equations and user language specifications. However, a database equation differs from a file equation in that a run-time error results if a COBOL74 program attempts to set or examine the TITLE attribute of the database while it is open.

The MOVE statement and the CHANGE statement manipulate the database TITLE attribute during program execution. For information on the use of the TITLE attribute, see the discussion of file attributes in Volume 1. Also refer to Volume 1 for more detailed information on the MOVE and CHANGE statements.

General Format

The general format of the MOVE and CHANGE statements is as follows:

```
{MOVE  }
{CHANGE} ATTRIBUTE TITLE {OF} internal-name
                         {IN}
TO alphanumeric-data-item
```

Example

In the following example, the first OPEN statement opens the LIVEDB database. The data and control files of LIVEDB are stored under the disk directory of the user. The second OPEN statement opens TESTDB. The files for TESTDB are stored on TESTPACK under the usercode UC.

Later in this section, Example 4–1, "DMSII Sample Program," shows the MOVE statement within the context of a complete program.

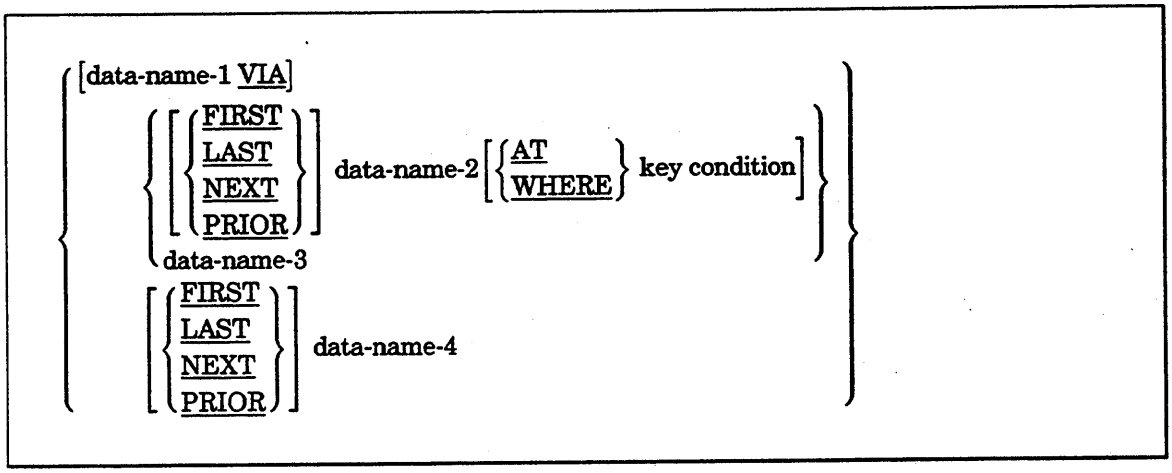
```
IDENTIFICATION DIVISION.
PROGRAM-ID. DBEQUATE.
.
.
.
DATA-BASE SECTION.

   DB MYDB ALL
   VALUE OF TITLE IS "LIVEDB".
.
.
.
OPEN UPDATE MYDB.
.
.
.
CLOSE MYDB.
CHANGE ATTRIBUTE TITLE OF MYDB
    TO "(UC)TESTDB ON TESTPACK".
OPEN UPDATE MYDB.
.
.
.
CLOSE MYDB.
STOP RUN.
```

Using Selection Expressions

A selection expression is used in FIND, LOCK, MODIFY, DELETE, and SECURE statements to identify a particular record in a data set. In the following explanation, the condition option is referred to as the key condition.

General Format

The general format for selection expressions is as follows:



Explanation of Format Elements

data-name-1

This format element identifies the record area and current path that are affected if the desired record is found. You can use this option for link items and for sets not implicitly associated with the data set.

VIA

This option indicates the path to be taken to identify the record in the data set.

FIRST

This option selects the first record in the specified data set, set, or subset. FIRST is specified by default. If you specify a key condition, DMSII selects the first record that satisfies the key condition.

LAST

This option selects the last record in the specified data set, set, or subset. If you specify a key condition, DMSII selects the last record that satisfies the key condition.

However, the record returned might not always be the absolute last record. Assume that program A performs a LOCK LAST operation when program B has the last record locked. Assume also that while program A is waiting, one or more records are added to the data set, set, or subset. The record program A receives is the last record that existed when program A performed the LOCK LAST and began its wait for the record, not the last existing record at the time program B released the record.

NEXT

This option selects the next record relative to either of the following:

- The set path if you specify a set name or subset name
- The data set path if you specify a data set name

If you specify a key condition, DMSII selects the next record (relative to the current path) that satisfies the key condition.

PRIOR

This option selects the prior record relative to either of the following:

- The set path if you specify a set name or subset name
- The data set path if you specify a data set name

If you specify a key condition, DMSII selects the prior record (relative to the current path) that satisfies the key condition.

data-name-2

This option identifies the record referred to by the set path. Data-name-2 must be a set or a subset. DMSII returns a NOTFOUND exception if the record has been deleted or if the path does not refer to a valid current record.

AT or WHERE

This option indicates that a key condition follows. AT and WHERE are synonyms.

key condition

This option specifies values used to locate specific records in a data set referenced by a particular set or subset.

data-name-3

This format element specifies a link item defined in DASDL. DMSII selects the record to which the link item refers and returns an exception if the link item is NULL.

data-name-4

This format element selects the record referred to by the data set path. DMSII returns a NOTFOUND exception if the record has been deleted or if the path does not refer to a valid current record. Data-name-4 must be a data set name.

FIRST, LAST NEXT and PRIOR

These options are described after the explanation of data-name-1.

Considerations for Use

If a specified data item is not unique, the compiler provides implicit qualification through the set name or subset name. You can qualify the item with the data set name containing the item; however, the compiler handles this qualification as documentation only.

Example

The following example locates the record S where A is equal to 50 and B is equal to 50, or locates the record S where A is equal to 50 and C is less than 90. The example also locates the record S where A is equal to the literal "MYNAME".

```
FIND S AT A = 5Ø AND (B = 5Ø OR C < 9Ø).

FIND S WHERE A = "MYNAME".
```

Using Data Management Attributes

Data management (DM) attributes are similar to file and task attributes in COBOL74. DM attributes allow read-only access to the following:

- Count field of a record
- Record Type field of a record
- Current population of a structure name

A description of the count, record type, and population attributes is provided in the following text.

Count Attribute

The value of the count attribute is the number of counted references pointing at the record in the Count field.

Because the ASSIGN statement updates a count item directly in the database, the value of the Count field can differ from the actual value in the database, even if the field is tested immediately after the record containing the Count field is made current.

DMSII returns an exception when you attempt to delete a record and the count item is not 0 (zero).

General Format

The general format for the count attribute is as follows:

```
data-name-1 (data-name-2)
```

Explanation of Format Elements

data-name-1

This format element specifies the name of the data set.

data-name-2

This format element designates a count name. The use of data-name-2 enables read-only access to the Count field of a record.

Example

The following example provides the DASDL description for code that exemplifies the use of the count attribute:

```
D DATA SET (
 A ALPHA (3);
 L IS IN E COUNTED;
 );

E DATA SET (
  C COUNT;
  N NUMBER (3);
  );
```

A COBOL74 statement that tests the count attribute follows:

```
IF E(C) = Ø DELETE D ON EXCEPTION PERFORM . . .
```

Record Type Attribute

The value in the record type attribute represents the type of record in the current record area.

General Format

The general format for the record type attribute is as follows:

```
data-name-1 (record-name-1)
```

Explanation of Format Elements

data-name-1

This format element specifies the name of the data set

record-name-1

This format element is the name of the record. Specifying the record enables read-only access to the Record Type field of a record.

Example

The following example provides the DASDL description for code that exemplifies the use of the record type attribute:

```
D DATA SET (
T RECORD TYPE (2);
A ALPHA (3);
),

2: (
   B BOOLEAN;
   R REAL;
   N NUMBER (3);
          ) ;
```

A COBOL74 statement that tests the record type attribute follows:

```
IF D(T) = 2 GO TO . . .
```

Population Attribute

The population attribute enables read-only access to the current population of the structure name. However, this value is often inaccurate, even if it is tested immediately after the record that contains it is made current. This inaccuracy occurs because other programs running concurrently on a multiprocessing system can cause the value of the population item in the database to change.

General Format

The general format for the population attribute is as follows:

```
data-name-1 (data-name-2)
```

Explanation of Format Elements

data-name-1

This format element specifies the name of the data set.

data-name-2

This format element specifies a population item.

Example

The following example provides the DASDL description for COBOL74 code that exemplifies the use of the population attribute:

```
D DATA SET ( . . .
EPOP POPULATION (100) OF E;
.
.
.
E DATA SET ( ...);
.
.
.
        );
```

The operation in this example accesses not the population of D, but the population of the structure embedded in D to which EPOP refers.

A COBOL74 statement that uses the population attribute follows:

```
MOVE D (EPOP) TO X.
```

Using the DATADICTINFO Option

The DATADICTINFO option causes the compiler to place database and transaction base usage data into the object code file. This compiler control option is used with the DMSII data dictionary. It is not used with ADDS.

To save usage data in the object code file, use the SET statement to set the DATADICTINFO option before the first source-language statement. You cannot use the SET, RESET, or POP option action indicators within the program.

General Format

The format of the DATADICTINFO option is as follows:

```
[DATADICTINFO]
```

Explanation of Format Element

DATADICTINFO

This option specifies the use of the DATADICTINFO option. The option is of type Boolean; the default value is FALSE.

See Also

- For information on compiler control options and the option action indicators, refer to Volume 1.
- For more information on the SET statement, see "SET Statement" later in this section.

Example

The following statement is an example of the DATADICTINFO option:

```
$SET DATADICTINFO.
```

Manipulating Data in a Database

You can use the following DM statements to manipulate data in a database.

ABORT-TRANSACTION Statement

The ABORT-TRANSACTION statement discards any updates made in a transaction after a BEGIN-TRANSACTION statement, and removes a program from transaction state.

General Format

The format of the ABORT-TRANSACTION statement is as follows:

```
ABORT-TRANSACTION [COMS-header-name-1] data-name-1
[ON EXCEPTION { imperative-statement-1   }]
              { conditional-statement-1  }
              { NEXT SENTENCE            }
```

Explanation of Format Elements

COMS-header-name-1

This option is used only with COMS and specifies a COMS header. COMS-header-name-1 calls the DCIENTRYPOINT of a data communications interface (DCI) library when a program detects an exception condition. This feature enables a program interfacing with COMS to support synchronized transactions and recovery.

data-name-1

This format element specifies the name of a restart data set (RDS).

ON EXCEPTION

This clause handles exceptions. Refer to "DMSII Exceptions" later in this section for information on the clause.

See Also

Refer to Section 3, "Using the COMS Program Interface," for more information on COMS.

Examples

The following lines of code provide examples of the ABORT-TRANSACTION statement:

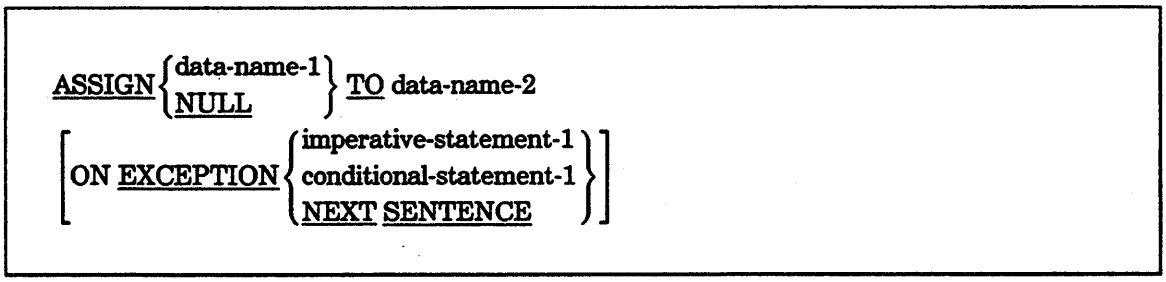
```
ABORT-TRANSACTION VENDOR-RESTART.

ABORT-TRANSACTION OUTHDR MY-RESTART.
```

ASSIGN Statement

The ASSIGN statement establishes the relationship between a record in a data set and a record in the same or another data set. The ASSIGN statement is effective immediately; therefore, the second record does not need to be stored unless data items of this record have been modified.

General Format

The general format of the ASSIGN statement is as follows:



Explanation of Format Elements

data-name-1

This format element specifies a data set or a link item.

If data-name-1 is a data set, you must declare it in DASDL as the object data set of the link item data-name-2. Data-name-2 is a value that points to the current record in data-name-1.

The current path of the data set specified by data-name-1 must be valid, but the record need not be locked. Your program returns an exception if the data set path is not valid.

If data-name-1 is a link item, it is assigned to data-name-2. Data-name-1 must be declared in DASDL according to the following requirements:

- Data-name-1 must have the same object data set as data-name-2.
- Data-name-1 must be the same type of link as data-name-2, for example, a counted link, a self-correcting link, a symbolic link, an unprotected link, or a verified link.

NULL

This clause severs the relationship between records by assigning a NULL value to data-name-2. If data-name-2 is already NULL, then DMSII ignores this clause. Executing a FIND, MODIFY, or LOCK statement on a NULL link item results in an exception.

data-name-2

This format element points either to the current record in the data set specified by data-name-1 (a data set) or to the record pointed to by data-name-1 (a link item).

The current path of the data set containing data-name-2 must be valid and the record locked; otherwise, the program returns an exception.

If data-name-1 refers to a disjoint data set, data-name-2 can point to any record in the data set. If data-name-2 refers to an embedded data set, it can reference only certain records in the data set. In this case, the record being referenced must be owned by the record containing data-name-2 or by an ancestor of the record containing data-name-2. (An ancestor is the owner of the record, the owner of the owner, and so forth.)

ON EXCEPTION

This clause handles exceptions. If the program finds an exception, it terminates the ASSIGN statement and assigns a NULL value to data-name-2. Refer to "DMSII Exceptions" later in this section for information on the ON EXCEPTION option.

Considerations for Use

Link items can easily join unrelated records. However, link items can also complicate the database as follows:

- Links must be maintained by the program. Other DMSII structures, such as automatic subsets, can perform the tasks that links perform but are maintained by the system.
- Links must be removed. If you delete a record pointed to by several links, you might forget to remove all the links pointing to that record. As a result, the links would point to nothing.
- Links are one-way pointers to a record. While you can find the record a link is pointing to, you cannot easily determine which record is pointing to the linked record.

If the link item is a counted link, DMSII automatically updates the count item, even if the referenced record is locked by another program.

Example

The following DASDL description is used by the COBOL74 code in the following example. The description is compiled with the name DBASE.

```
D DATA SET (
  A ALPHA (3);
  B BOOLEAN;
  L IS IN E VERIFY ON N;
  );
S SET OF D KEY A;

E DATA SET (
  N NUMBER (3);
  R REAL;
  );
T SET OF E KEY N;
```

The following example uses the ASSIGN statement:

```
IDENTIFICATION DIVISION.
PROGRAM-ID. EXAMPLE.
DATA DIVISION.
FILE SECTION.
FD  TAPE-FILE.
Ø1  TAPE-REC.
    Ø2  X PIC XXX.
    Ø2  Y PIC 999.
DATA-BASE SECTION.
DB  DBASE ALL.

PROCEDURE DIVISION.
OPEN-INPUT-FILE.
    OPEN INPUT TAPE-FILE.
OPEN-DB.
    OPEN UPDATE DBASE.
        START-PRG.
            READ TAPE-FILE AT END
                CLOSE TAPE-FILE
                CLOSE DBASE
                STOP RUN.
            FIND S AT A = X.
            FIND T AT N = Y.
            ASSIGN E TO L.
            FREE D.
            GO TO START-PRG.
```

BEGIN-TRANSACTION Statement

The BEGIN-TRANSACTION statement places a program in transaction state. This statement can be used only with audited databases.

The BEGIN-TRANSACTION statement performs the following steps:

1. Capture the RDS if the AUDIT clause is specified.
2. Place a program in transaction state.

General Format

The general format of the BEGIN-TRANSACTION statement is as follows:

```
BEGIN-TRANSACTION [COMS-header-name-1 option USING identifier-1]
{AUDIT    } data-name-1
{NO-AUDIT }
[ON EXCEPTION {imperative-statement-1  }]
              {conditional-statement-1 }
              {NEXT SENTENCE           }
```

Explanation of Format Elements

COMS-header-name-1

This option is used only with COMS and specifies the input header. You can use COMS-header-name-1 to call the DCIENTRYPOINT of a DCI library when your program detects an exception condition. This feature enables a program interfacing with COMS to support synchronized transactions and recovery.

Your program calls the DCI library before it performs the exception-handling procedure.

USING

If you have employed the USING option and your program does not detect an exception, your program calls the DCI library and passes the message area indicated by identifier-1 to the DCIENTRYPOINT.

identifier-1

This format element specifies the message area.

AUDIT

This clause captures the restart area. The path of the RDS named is not altered when the restart record is stored.

NO-AUDIT

This clause prevents the restart area from being captured.

data-name-1

This format element is the name of the RDS you want to update.

ON EXCEPTION

This clause handles exceptions. Your program returns an exception if you execute a BEGIN-TRANSACTION statement while the program is already in transaction state.

If an exception occurs, the program is not placed in transaction state. If the program returns an ABORT exception, all records that the program had locked are freed. Refer to "DMSII Exceptions" later in this section for information on the ON EXCEPTION option.

Considerations for Use

Any attempt to modify an audited database when the program is not in transaction state results in an audit error. The following DM statements modify databases:

- ASSIGN
- DELETE
- GENERATE
- INSERT
- REMOVE
- STORE

Example

The following DASDL description is used by the COBOL74 code in the following example. The description is compiled with the name DBASE.

```
OPTIONS (AUDIT);
R RESTART DATA SET (
  P ALPHA (10);
  Q ALPHA (100);
  );
D DATA SET (
  A ALPHA (3);
  N NUMBER (3);
  );
S SET OF D KEY N;
```

The following example uses the BEGIN-TRANSACTION statement:

```
IDENTIFICATION DIVISION.
PROGRAM-ID. EXAMPLE.
DATA DIVISION.
FILE SECTION.
FD  TAPE-FILE.
01  TAPE-REC.
    02 X PIC 999.
    02 Y PIC XXX.
DATA-BASE SECTION.
DB  DBASE ALL.
WORKING-STORAGE SECTION.
01 CNT PIC 999.
PROCEDURE DIVISION.
OPEN-INPUT-FILE.
    OPEN INPUT TAPE-FILE.
OPEN-DB.
    OPEN UPDATE DBASE.
CREATE-D.
    CREATE D.
    ADD 1 TO CNT.
    MOVE CNT TO N.
    BEGIN-TRANSACTION  AUDIT R.
        STORE D.
    END-TRANSACTION NO-AUDIT R.
    IF CNT  < 100
        GO TO CREATE-D.
START-PRG.
    READ TAPE-FILE AT END
        CLOSE TAPE-FILE
        CLOSE DBASE
        STOP RUN.
    LOCK S AT N = X.
    BEGIN-TRANSACTION AUDIT R
        MOVE Y TO A.
        STORE D.
    END-TRANSACTION NO-AUDIT R.
    GO TO START-PRG.
```

Later in this section, Example 4–1, "DMSII Sample Program," shows the BEGIN-TRANSACTION statement within the context of a complete program.

CANCEL TRANSACTION POINT Statement

The CANCEL TRANSACTION POINT statement discards all updates in a transaction or all updates after an intermediate transaction point or to the beginning of the transaction. The execution of the program continues with the statement following the CANCEL TRANSACTION POINT statement.

General Format

The general format for the CANCEL TRANSACTION POINT statement is as follows:

```
CANCEL TRANSACTION POINT data-name-1 [arithmetic-expression-1]
[ ON EXCEPTION { imperative-statement-1   } ]
               { conditional-statement-1  }
               { NEXT SENTENCE            }
```

Explanation of Format Elements

data-name-1

This format element is the name of an RDS.

arithmetic-expression-1

The CANCEL TRANSACTION POINT statement discards all database changes made between the current point in the transaction and the point specified by arithmetic-expression-1.

If you do not specify arithmetic-expression-1, DMSII discards all data updated since the BEGIN-TRANSACTION statement placed the program in transaction state. For details on arithmetic expressions, see Volume 1.

ON EXCEPTION

This clause handles exceptions. Refer to "DMSII Exceptions" later in this section for information on the ON EXCEPTION option.

Examples

The following lines of code provide examples of the CANCEL TRANSACTION POINT statement:

```
CANCEL TRANSACTION POINT MY-RESTART MAIN-SAVE-POINT.

CANCEL TRANSACTION POINT MY-RESTART.
```

CLOSE Statement

The CLOSE statement closes a database when your program requires no further access to it. The CLOSE statement is optional because the system closes any open database when the program terminates. A successfully closed database causes a syncpoint in the audit trail.

The CLOSE statement performs the following steps:

1. Closes the database
2. Frees all locked records

General Format

The general format of the CLOSE statement is as follows:

```
CLOSE data-name-1
[ ON EXCEPTION { imperative-statement-1 }
               { conditional-statement-1 } ]
               { NEXT SENTENCE           }
```

Explanation of Format Elements

data-name-1

This format element specifies the database you want to close.

ON EXCEPTION

This clause handles exceptions. Your program returns an exception if the specified database is not open. Refer to "DMSII Exceptions" later in this section for information on the clause.

Considerations for Use

The CLOSE statement is the only statement in which the status word has meaning when no exception is indicated. Your program should therefore examine the status word after it closes a database and should take appropriate action, whether or not it received an exception. If the ON EXCEPTION option is not used and the CLOSE statement is invalid, the program aborts.

The CLOSE statement closes the database unconditionally, regardless of exceptions. If you use just the CLOSE syntax, the program is discontinued on all exceptions that raise the exception flag.

Your program does not return some exceptions when the CLOSE statement is used. To be sure your program detects all exceptions that occur during the execution of the CLOSE statement, do both of the following:

- Use the ON EXCEPTION option to prevent the program from being discontinued if an exception flag is raised.
- Use an IF statement to check for exceptions that do not raise an exception flag.

If you are running COMS for synchronized recovery, Unisys recommends that you do not use the ON EXCEPTION option. If DMSII detects a database error during the closing of a database, the system allows your program to terminate abnormally; otherwise, the database might abort recursively. If you do use the ON EXCEPTION option, you should ensure that your program calls the DMTERMINATE statement for those exceptions that your program does not handle. Use the following syntax, therefore, to close a database when you are using COMS with DMSII for synchronized recovery:

```
CLOSE DBASE.
```

Example

The following example shows the recommended syntax for the CLOSE statement when the ON EXCEPTION option and the IF statement are used:

```
CLOSE MYDB
ON EXCEPTION
   DISPLAY "EXCEPTION WHILE CLOSING MYDB"
   CALL SYSTEM DMTERMINATE
IF DMSTATUS(DMERROR)
   OPEN MYDB
   GO TO ABORTED.
```

Later in this section, Example 4–1, "DMSII Sample Program," shows the CLOSE statement within the context of a complete program.

COMPUTE Statement

The COMPUTE statement assigns a value to a Boolean item in the current record of a data set. The COMPUTE statement affects only the record area. The database is not affected until a subsequent STORE statement is executed.

No exceptions are associated with this statement.

General Format

The general format of the COMPUTE statement is as follows:

```
COMPUTE data-name-1 = { condition }
                      { TRUE      }
                      { FALSE     }
```

Explanation of Format Elements

data-name-1

This format element specifies the current record of a data set.

condition

If you specify a condition, DMSII assigns the value of the condition to the specified Boolean item. The format rules for the condition are the same as the standard COBOL74 rules for the relation conditions.

TRUE

This clause assigns a TRUE value to the specified Boolean item.

FALSE

This clause assigns a FALSE value to the specified Boolean item.

Examples

The following lines of code provide two examples of the COMPUTE statement:

```
COMPUTE CLOSEFLAG = TRUE.

COMPUTE CHECKBALANCE = OLD-BALANCE + DEPOSIT EQUAL CURR-BALANCE
```

Later in this section, Example 4–1, "DMSII Sample Program," shows the COMPUTE statement within the context of a complete program.

CREATE Statement

The CREATE statement initializes the user work area of a data set record.

The CREATE statement performs the following steps:

1. Frees the current record of the specified data set. Note that if the INDEPENDENTTRANS option in DASDL is set and the CREATE statement is issued during transaction state, the locked record is not freed until an END-TRANSACTION statement is executed.

   For more information on the INDEPENDENTTRANS option, refer to the *DMSII DASDL Reference Manual*.
2. Reads the specified expression to determine the format of the record to be created.
3. Initializes data items to one of the following values:
   - INITIALVALUE value declared in DASDL, if present
   - NULL value declared in DASDL, if present
   - NULL value as a default value

General Format

The general format of the CREATE statement is as follows:

```
CREATE data-name-1 [(expression)]
[ON EXCEPTION {imperative-statement-1
               conditional-statement-1
               NEXT SENTENCE}]
```

Explanation of Format Elements

data-name-1

This format element specifies the data set you want to initialize. The current path of the data set is unchanged until you execute a STORE statement.

(expression)

The expression option specifies a variable-format record. You must use an expression only to create a variable-format record; otherwise, the expression must not appear.

ON EXCEPTION

This clause handles exceptions. Your program returns an exception if the expression does not represent a valid record type. See "DMSII Exceptions" later in this section for information.

Considerations for Use

You normally use a CREATE statement with a STORE statement to place the newly created record into the data set. A BEGIN-TRANSACTION statement must precede the STORE statement and an END-TRANSACTION statement is coded after the STORE statement. If you do not want to store the record, you can nullify the CREATE statement by executing a subsequent FREE statement or by using a FIND, LOCK, DELETE, CREATE, or RECREATE statement.

The CREATE statement only sets up a record area. If the record contains embedded structures, you must store the master record before you can create entries in the embedded structures. If you create only entries in the embedded structure (that is, if you do not alter items in the master), you need not store the master record a second time.

Example

The following DASDL description is used by the COBOL74 code in the following example. The description is compiled with the name DBASE.

```
D DATA SET (
  A ALPHA (10);
  B BOOLEAN;
  N NUMBER (3);
  );
S SET OF D KEY N;
```

The following example uses the CREATE statement:

```
IDENTIFICATION DIVISION.
PROGRAM-ID. EXAMPLE.
DATA DIVISION.
FILE SECTION.
FD  TAPE-FILE.
Ø1  TAPE-REC.
    Ø2  X PIC X(1Ø).
    Ø2  Y PIC 9.
    Ø2  Z PIC 999.
DATA-BASE SECTION.
DB  DBASE ALL.
PROCEDURE DIVISION.
OPEN-INPUT-FILE.
    OPEN INPUT TAPE-FILE.
OPEN-DB.
    OPEN UPDATE DBASE.
START-PRG.
    READ TAPE-FILE AT END
       CLOSE TAPE-FILE
       CLOSE DBASE
       STOP RUN.
    CREATE D.
    MOVE X TO A.
    IF Y = 1
    COMPUTE B = TRUE.
    MOVE Z TO N.
    BEGIN-TRANSACTION.
    STORE D.
    END-TRANSACTION.
    GO TO START-PRG.
```

Later in this section, Example 4–1, "DMSII Sample Program," shows the CREATE statement within the context of a complete program.

DELETE Statement

The DELETE statement finds a record by a method identical to that of the FIND statement. However, whereas the FIND statement transfers the record to the user work area associated with a data set or global data, the DELETE statement performs the following steps:

1. Frees the current record, unless the selection expression is the name of the data set and the current record is locked. In this case, the locked status is not altered.
2. Alters the current path to point to the record specified by the selection expression, and locks this record.
3. Transfers that record to the user work area.
4. Removes the record from all sets and automatic subsets, but not from manual subsets.
5. Removes the record from the data set.

If your program finds a record that cannot be deleted, the program returns an exception and terminates the DELETE statement, leaving the current path pointing to the record specified by the selection expression.

If you use a set selection expression and your program cannot find the record, the program returns an exception and changes and invalidates the set path. The selection expression refers to a location between the last key that is less than the condition and the first key that is greater than the condition. You can execute a set selection expression with the NEXT or PRIOR options from this location, provided keys that are greater than or less than the condition exist. The current path of the data set, the current record, and the current paths of any other sets for that data set remain unchanged.

When the DELETE statement is completed, the current paths still refer to the deleted record.

General Format

The general format of the DELETE statement is as follows:

```
DELETE selection-expression-1
[ ON EXCEPTION { imperative-statement-1   } ]
               { conditional-statement-1  }
               { NEXT SENTENCE            }
```

Explanation of Format Elements

selection-expression-1

This format element identifies the record you want to delete. Selection expressions are explained in "Using Selection Expressions" earlier in this section.

ON EXCEPTION

This clause handles exceptions. For information on the clause, refer to "DMSII Exceptions" later in this section.

Your program returns an exception and does not delete the record if one of the following is true:

- Counted links are pointing to the record.
- The record contains a link other than a NULL link, or an embedded structure that contains entries.

Your program returns an exception if the record belongs to a manual subset. Refer to "REMOVE Statement" later in this section.

Example

The following DASDL description is used by the COBOL74 code in the following example. The description is compiled with the name DBASE.

```
D DATA SET (
  A ALPHA (3);
  B BOOLEAN;
  N NUMBER (3);
  R REAL;
  );
S SET OF D KEY N;
```

The following example uses the DELETE statement:

```
IDENTIFICATION DIVISION.
PROGRAM-ID. EXAMPLE.
DATA DIVISION.
FILE SECTION.
FD  TAPE-FILE.
Ø1  TAPE-REC.
    Ø2  X PIC 999.
DATA-BASE SECTION.
DB  DBASE ALL.

PROCEDURE DIVISION.
OPEN-INPUT-FILE.
    OPEN INPUT TAPE-FILE.
OPEN-DB.
    OPEN UPDATE DBASE.
START-PRG.
    READ TAPE-FILE AT END
        CLOSE TAPE-FILE
        CLOSE DBASE
        STOP RUN.
    DELETE S AT N = X.
    GO TO START-PRG.
```

Later in this section, Example 4–1, "DMSII Sample Program," shows the DELETE statement within the context of a complete program.

DMTERMINATE Statement

The DMTERMINATE statement terminates programs. When an exception occurs that the program does not handle, DMTERMINATE terminates the program with a fault.

General Format

The general format of the DMTERMINATE statement is as follows:

```
CALL SYSTEM DMTERMINATE
```

Example

The following example uses the DMTERMINATE statement:

```
FIND FIRST D
ON EXCEPTION
  DISPLAY "D IS EMPTY DATA SET"
  CALL SYSTEM DMTERMINATE.
```

END-TRANSACTION Statement

The END-TRANSACTION statement takes a program out of transaction state. You can use this statement only with audited databases. The END-TRANSACTION statement performs the following steps:

1. Captures the restart area if the AUDIT clause is specified
2. Forces a syncpoint if the SYNC option is specified
3. Implicitly frees all records of the database that the program has locked

General Format

The general format of the END-TRANSACTION statement is as follows:

```
END-TRANSACTION [COMS-header-name-1[USING identifier-1]]
{AUDIT    } data-name-1 [SYNC]
{NO-AUDIT }
[ON EXCEPTION{imperative-statement-1  }]
             {conditional-statement-1 }
             {NEXT SENTENCE           }
```

Explanation of Format Elements

COMS-header-name-1

This option specifies the COMS output header. For information on COMS, refer to Section 3, "Using the COMS Program Interface."

The COMS-header-name-1 option is used only with COMS. You can specify COMS-header-name-1 to call the DCIENTRYPOINT of a DCI library whenever you execute the statement. This feature enables a program interfacing with COMS to support synchronized transactions and recovery. When your program detects an exception condition, it calls the DCI library before it performs any exception-handling procedures.

USING

This option causes your program to call the DCI library and pass the message area indicated by identifier-1 to the DCIENTRYPOINT.

identifier-1

This format element specifies the message area passed to the DCIENTRYPOINT.

AUDIT

This clause captures the restart area. Storing the restart record does not alter the path of the RDS.

NO-AUDIT

This clause prevents the restart area from being captured.

data-name-1

This format element is the name of the restart data area specified in the AUDIT or NO-AUDIT clause.

SYNC

This option forces a syncpoint.

ON EXCEPTION

This clause handles exceptions. Refer to "DMSII Exceptions" later in this section for information on the clause. The program returns an exception if you execute an END-TRANSACTION statement when the program is not in transaction state.

The END-TRANSACTION statement frees all records. However, if your program returns an exception, the transaction is not applied to the database.

Example

The following DASDL description is used by the COBOL74 code in the following example. The description is compiled with the name DBASE.

```
OPTIONS (AUDIT);
R RESTART DATA SET (
  P ALPHA (10);
  Q ALPHA (100);
  );
D DATA SET (
  A ALPHA (3);
  N NUMBER (3);
  );
S SET OF D KEY N;
```

The following example shows two sections of code, each of which begins with a BEGIN-TRANSACTION statement and ends with an END-TRANSACTION statement. Both sections of code define a transaction. The transaction becomes an indivisible, logical unit. During processing, the transactions are audited for recovery. The AUDIT or NO-AUDIT clause determines whether the restart record of the data set is captured.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. EXAMPLE.
DATA DIVISION.
FILE SECTION.
FD  TAPE-FILE.
Ø1  TAPE-REC.
    Ø2  X PIC 999.
    Ø2  Y PIC XXX.
DATA-BASE SECTION.
DB  DBASE ALL.
WORKING-STORAGE SECTION.
Ø1 CNT PIC 999.
PROCEDURE DIVISION.
OPEN-INPUT-FILE.
    OPEN INPUT TAPE-FILE.
OPEN-DB.
    OPEN UPDATE DBASE.
CREATE-D.
    CREATE D.
    ADD 1 TO CNT.
    MOVE CNT TO N.
    BEGIN-TRANSACTION AUDIT R.
        STORE D.
    END-TRANSACTION AUDIT R.
    IF CNT  < 1ØØ
        GO TO CREATE-D.

START-PRG.
    READ TAPE-FILE AT END
        CLOSE TAPE-FILE
        CLOSE DBASE
        STOP RUN.
LOCK S AT N = X.
BEGIN-TRANSACTION AUDIT R.
    MOVE Y TO A.
    STORE D.
END-TRANSACTION  NO-AUDIT R
GO TO START-PRG.
```

Later in this section, Example 4–1, "DMSII Sample Program," shows the END-TRANSACTION statement within the context of a complete program.

FIND Statement

The FIND statement transfers a record to the user work area associated with a data set or global data. Additional information on the use of the FIND statement with the REBLOCK and READAHEAD options is provided in the *DMSII Application Programming Guide*.

The FIND statement performs the following steps:

1. Frees a locked record in the data set if you specify a data set in the FIND statement. Specifying a set in the FIND statement frees a locked record in the associated data set.
2. Alters the current path to point to the record specified by the selection expression or the database name.
3. Transfers that record to the user work area.

Using the FIND statement does not prevent other transactions from reading the record before the current update transaction is completed.

General Format

The general format of the FIND statement is as follows:

```
{ FIND { selection-expression-1 }                                          }
{      { data-name-1            }                                          }
{ FIND KEY OF selection-expression-2                                       }
[ ON EXCEPTION { imperative-statement-1  }                                  ]
[              { conditional-statement-1 }                                 ]
[              { NEXT SENTENCE           }                                 ]
```

Explanation of Format Elements

selection-expression-1

This format element specifies the record you want to transfer to the user work area.

data-name-1

This format element specifies the global data record you want to transfer to the user work area associated with the global data. If no global data is described in DASDL, DMSII returns a syntax error.

FIND KEY OF

This clause moves the key and any associated data (as specified in the DASDL description) from the key entry to the user work area. Your program does not perform a physical read on the data set; consequently, the value and contents of all items in the record area that do not appear in the key entry retain whatever value they had before you executed the FIND KEY OF clause. The FIND statement does not affect the current path of the data set.

selection-expression-2

If you use selection-expression-2 and your program fails to find the record, the program returns an exception, and changes and invalidates the set path. The selection expression refers to a location between the last key that is less than the condition and the first key that is greater than the condition. You can process selection-expression-2 by using the NEXT or PRIOR option from this location, provided the keys that are greater than or less than the condition exist. The current path of the data set, the current record, and the current paths of any other sets for that data set remain unchanged.

ON EXCEPTION

This clause handles exceptions. Refer to "DMSII Exceptions" later in this section for information on the clause. The program returns an exception if no record satisfies the selection expression.

Examples

The following example shows the FIND statement used with a record selection expression:

```
FIND FIRST OVER-65 AT DEPT-NO = 1Ø19
ON EXCEPTION
   MOVE Ø TO POP-OVR-65 (1Ø19).
```

The following example shows the FIND statement used with a FIND KEY OF clause:

```
FIND KEY OF NAME-KEYS AT NAME = "FRED JONES".
```

Later in this section, Example 4–1, "DMSII Sample Program," shows the FIND statement within the context of a complete program.

FREE Statement

The FREE statement explicitly unlocks the current record or structure. A FREE statement executed on a record enables other programs to lock that record or structure.

Note that if you set the INDEPENDENTTRANS option in DASDL for the database, the program ignores a FREE statement during transaction state. For more information on the INDEPENDENTTRANS option, refer to the *DMSII DASDL Reference Manual* and the *DMSII Application Programming Guide*.

You can execute a FREE statement after any operation. If the current record or structure is already free, or if no current record or structure is present, DMSII ignores the FREE statement.

You can use the FREE statement to unlock a record or structure that you anticipate will not be implicitly freed for a long time.

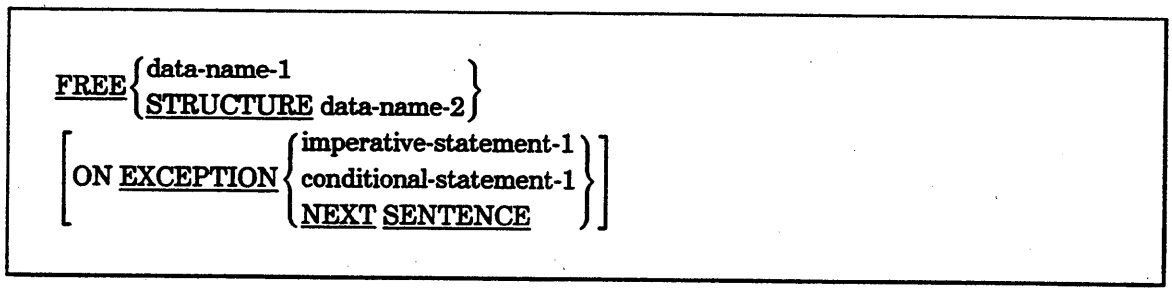
The FREE statement is optional in some situations because the FIND, LOCK, MODIFY, and DELETE statements can free a record before the program executes them. Generally, an implicit FREE operation is performed, if needed, during any operation that establishes a new data set path.

FIND, LOCK, and MODIFY statements using sets or subsets free the locked record only if a new record is retrieved. Other constructs that can free data set records are

- END-TRANSACTION
- BEGIN-TRANSACTION
- CREATE
- RECREATE
- SET TO BEGINNING
- SET TO ENDING

General Format

The general format for the FREE statement is as follows:



Explanation of Format Elements

data-name-1

This format element specifies either the data set whose current record is to be unlocked or the global data record to be unlocked. The data set path and current record area remain unchanged.

STRUCTURE

This phrase frees all records in the structure.

data-name-2

This format element specifies the structure to be freed.

ON EXCEPTION

This option handles exceptions. Refer to "DMSII Exceptions" later in this section for information on the clause. If the program returns an exception, the state of the database remains unchanged.

Example

The following example uses the FREE statement:

```
LOCK NEXT S
ON EXCEPTION
   GO TO NO-MORE.
IF ITEM-1 NOT = VALID-VALUE
   FREE DS
   GO ERR.
```

GENERATE Statement

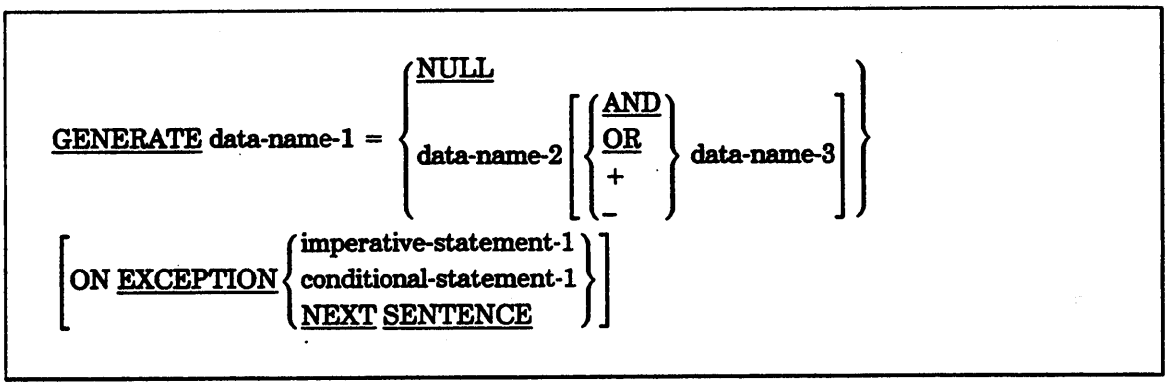
The GENERATE statement creates an entire subset in one operation. All subsets must be disjoint bit vectors. The GENERATE statement performs the following steps:

1. Deletes all records from the subset if the subset is not empty
2. Assigns to the generated subset the records in another subset, a combination of the records in two other subsets, or NULL values

Note: Unisys recommends that you coordinate any subset declaration with other users because subsets can be used concurrently and altered without your knowledge.

General Format

The general format of the GENERATE statement is as follows:



Explanation of Format Elements

data-name-1

This format element is the name of the subset you want to generate. Data-name-1 must be a manual subset and a disjoint bit vector.

NULL

This phrase assigns a NULL value to the generated subset so that the subset remains empty.

data-name-2

This format element is the name of the subset you want to assign to data-name-1. The data-name-2 subset must be of the same data set as the data-name-1 subset and must be a disjoint bit vector.

AND

This operator assigns the intersection of data-name-2 and data-name-3 to data-name-1. The intersection is defined to be all the records in data-name-2 that are also in data-name-3.

OR

This operator assigns the union of data-name-2 and data-name-3 to data-name-1. The union is defined to be all the records that are in either data-name-2 or data-name-3.

+ (plus)

This operator is the subset-exclusive OR. This operator assigns the records contained in either data-name-2 or data-name-3 (but not both) to data-name-1.

– (minus)

This operator is the subset difference. This operator assigns the records contained in data-name-2 that are not in data-name-3 to data-name-1.

data-name-3

This format element is the name of the subset you want to combine with data-name-2 and assign to data-name-1. The data-name-3 subset must be of the same data set as the data-name-2 subset and must be a disjoint bit vector.

ON EXCEPTION

This option handles exceptions. Refer to "DMSII Exceptions" later in this section for more information on the clause.

Example

The following DASDL description is used by the COBOL74 code in the following example. The description is compiled with the name DBASE.

```
DASDL (compiled as DBASE):
D DATA SET (
  A ALPHA (3);
  B BOOLEAN;
  N NUMBER (3);
  R REAL;
  );
X SUBSET OF D WHERE (N GEQ 21 AND NOT B) BIT VECTOR;
Y SUBSET OF D WHERE (R LSS 1000) BIT VECTOR;
Z SUBSET OF D BIT VECTOR;
```

The following example uses the GENERATE statement:

```
IDENTIFICATION DIVISION.
PROGRAM-ID. EXAMPLE.
DATA DIVISION.
FILE SECTION.
FD  TAPE-FILE.
Ø1  TAPE-REC.
    Ø2  S PIC XXX.
    Ø2  T PIC 9.
    Ø2  U PIC 999.
    Ø2  V PIC 9(4).
DATA-BASE SECTION.
DB  DBASE ALL.
PROCEDURE DIVISION.
OPEN-INPUT-FILE.
    OPEN INPUT TAPE-FILE.
OPEN-DB.
    OPEN UPDATE DBASE.
START-PRG.
    READ TAPE-FILE AT END
    CLOSE TAPE-FILE
    GO TO GENERATE-SUBSET.
    CREATE D.
    MOVE S TO A.
    IF T = 1
        COMPUTE B = TRUE.
    MOVE U TO N.
    MOVE V TO R.
    STORE D.
    GO TO START-PRG.
GENERATE-SUBSET.
    GENERATE Z = X AND Y.
    CLOSE DBASE.
    STOP RUN.
```

Later in this section, Example 4–1, "DMSII Sample Program," shows the GENERATE statement within the context of a complete program.

IF Statement

The IF statement tests an item to determine if it contains a NULL value.

Note: *Unisys recommends that you use the IF NULL test to test a database item for a NULL value. Attempts to compare a database item against some value assumed to be equivalent to NULL might not always produce correct results.*

General Format

The general format of the IF statement is as follows:

```
IF { { data-name-1 [NOT] } NULL                          } imperative-statement-1
   { { NOT data-name-2 IS }                              }
   { [ NOT { data-name-3 } ]                             }
   { [     { data-name-4 is NULL } ]                     }
[ELSE imperative-statement-2]
```

Explanation of Format Elements

data-name-1 and data-name-2

These format elements identify items you want to test.

imperative-statement-1

Your program executes imperative-statement-1 if the condition you are testing is satisfied.

NULL

This phrase is the null value defined in the DASDL description. The NULL phrase specifies a condition that can also appear in combined conditions. Refer to Volume 1 for information on combined conditions.

When the NULL phrase is applied to a group item, the NULL phrase defined in the DASDL description is used, regardless of the NULL specification for any elementary item contained within the group.

Data items declared in the DASDL description, besides being used in the NULL test, can also be used in standard COBOL74 relation conditions, exactly like data items declared in a COBOL74 program.

data-name-3

This format element specifies a Boolean item declared in the DASDL specification.

data-name-4

This format element specifies a link item declared in the DASDL specification. The specified link item contains a NULL value if one of the following is true:

- The link item does not point to a record.
- No current record is present for the data set containing the link item. This condition occurs following the OPEN, SET TO BEGINNING, or SET TO ENDING statements, or when the record containing the link item has been deleted.

The data-name-4 link item contains a value other than null if the link item points to a record, even if that record has been deleted.

imperative-statement-2

If the condition specified in imperative-statement-1 is not satisfied, imperative-statement-2 is executed.

Example

The following example illustrates the use of the NULL phrase with the IF statement:

```
IF THE-ITEM IS NULL
     PERFORM NEVER-USED.
```

Later in this section, Example 4–1, "DMSII Sample Program," shows the IF statement within the context of a complete program.

INSERT Statement

The INSERT statement places a record into a manual subset. To accomplish this task, the statement performs the following steps:

1. Inserts the current record of the specified data set into the specified subset
2. Alters the set path for the specified subset to point to the inserted record

General Format

The general format of the INSERT statement is as follows:

```
INSERT data-name-1 INTO data-name-2
[ ON EXCEPTION { imperative-statement-1   } ]
               { conditional-statement-1  }
               { NEXT SENTENCE            }
```

Explanation of Format Elements

data-name-1

This format element identifies the data set whose current record you want to insert into the subset specified by data-name-2. Data-name-1 must be the object data set of the specified subset. The path of data-name-1 must refer to a valid record; otherwise, the program returns an exception.

data-name-2

This format element must specify a manual subset of the data set specified by data-name-1.

ON EXCEPTION

This option handles exceptions. Refer to "DMSII Exceptions" later in this section for information on the clause. The program returns an exception if one of the following is true:

- The subset you specified does not permit duplicates, and the record you want to insert has a key identical to that of a record currently in the specified subset.
- The specified subset is embedded in a data set, and the data set does not have a valid current record.
- The LOCK TO MODIFY DETAILS option was specified in the DASDL description, and the current record is not locked.

Example

The following DASDL description is used by the COBOL74 code in the following example. The description is compiled with the name DBASE.

```
D DATA SET (
  A ALPHA (3);
  B BOOLEAN;
  N NUMBER (3);
  R REAL;
  );
X SUBSET OF D BIT VECTOR;
```

The following example uses the INSERT statement:

```
IDENTIFICATION DIVISION.
PROGRAM-ID. EXAMPLE.
DATA DIVISION.
DATA-BASE SECTION.
DB  DBASE ALL.
PROCEDURE DIVISION.
OPEN-DB.
    OPEN UPDATE DBASE.
    SET D TO BEGINNING.
START-PRG.
    FIND NEXT D ON EXCEPTION
        CLOSE DBASE
        STOP RUN.
    IF N > 10
        INSERT D INTO X.
    GO TO START-PRG.
```

Later in this section, Example 4–1, "DMSII Sample Program," shows the INSERT statement within the context of a complete program.

LOCK/MODIFY Statement

The LOCK statement finds a record in a manner identical to that of the FIND statement, and then locks a found record against a concurrent modification by another user. LOCK and MODIFY are synonyms. The LOCK statement also provides the STRUCTURE phrase, which simultaneously locks all records in a structure.

If the record to be locked has already been locked by another program, the system performs a contention analysis. The present program waits until the other program unlocks the record, unless the wait would result in a deadlock. If a deadlock would result, the system unlocks all records locked by the program having the lowest priority of the programs that would be involved in the deadlock, and terminates the lower-priority program with a DEADLOCK exception.

No other user can lock or secure the record once it is locked; therefore, the record must be freed when locking is no longer required. You can free a record explicitly by using a FREE statement or free a record implicitly by executing a subsequent LOCK, FIND, DELETE, CREATE, or RECREATE statement on the same data set.

The LOCK/MODIFY statement performs the following steps:

1. Implicitly frees a locked record. However, if you have set the INDEPENDENTTRANS option in DASDL, the LOCK/MODIFY statement does not free the locked record until you execute an END-TRANSACTION statement.
2. Alters the current path to point to the record specified by the selection expression or data name included in the statement.

3. Locks the specified record.
4. Transfers that record to the user work area.

General Format

The general format of the LOCK/MODIFY statement is as follows:

```
{LOCK  }  {selection-expression-1                }
{MODIFY}  {data-name-1                           }
          {STRUCTURE data-name-2                 }
[ON EXCEPTION {imperative-statement-1  }]
              {conditional-statement-1 }
              {NEXT SENTENCE           }]
```

Explanation of Format Elements

selection-expression-1

This format element specifies the record you want to lock.

data-name-1

This format element specifies the global data record you want to lock.

STRUCTURE

This phrase locks or secures all records in the structure simultaneously. If other users have locked or secured the structure or records in the structure, you must wait until those users free the records or the structure, or end their transactions to do so. A deadlock occurs when other users attempt to lock or secure more records while you are locking the structure. Once you have locked a structure, you must continue to lock individual records. Each new lock implicitly frees the previous record, even if you have set the INDEPENDENTTRANS option in DASDL. These freed records continue to be available only to the user securing the structure.

You cannot free structure locks with an END-TRANSACTION statement; instead, you must use a FREE statement. More information is provided in "FREE Statement" earlier is this section.

data-name-2

Data-name-2 must be a data set.

ON EXCEPTION

This option handles exceptions. Refer to "DMSII Exceptions" for more information on the clause. The program returns an exception if no record satisfies the selection expression. If the program returns an exception, the record is not freed. A DEADLOCK exception occurs if the program waits on a LOCK statement for a time longer than that specified in the MAXWAIT task attribute. For more information about the MAXWAIT attribute, refer to the *A Series Task Attributes Programming Reference Manual*.

If the LOCK statement uses a selection expression and an exception is returned, the program invalidates the current path of the specified set. However, the current path of the data set, the current record, and the current paths of any other sets for that data set remain unaltered.

Examples

The following line of code shows the LOCK statement used with the STRUCTURE phrase:

```
LOCK STRUCTURE VENDOR-DATA.
```

The following example shows the LOCK statement used with the ON ON EXCEPTION option:

```
LOCK FIRST EMP AT DEPT-NO = 1Ø19
ON EXCEPTION
   MOVE Ø TO POP-EMP (1Ø19).
```

The following example shows the MODIFY statement used with the ON EXCEPTION clause:

```
MODIFY EMP AT EMP-NO = IN-SSN
ON EXCEPTION
   MOVE INV-EMP-NO-ERR TO ERR-MSG
   PERFORM ERR-OUT.
```

Later in this section, Example 4–1, "DMSII Sample Program," shows the LOCK statement within the context of a complete program.

OPEN Statement

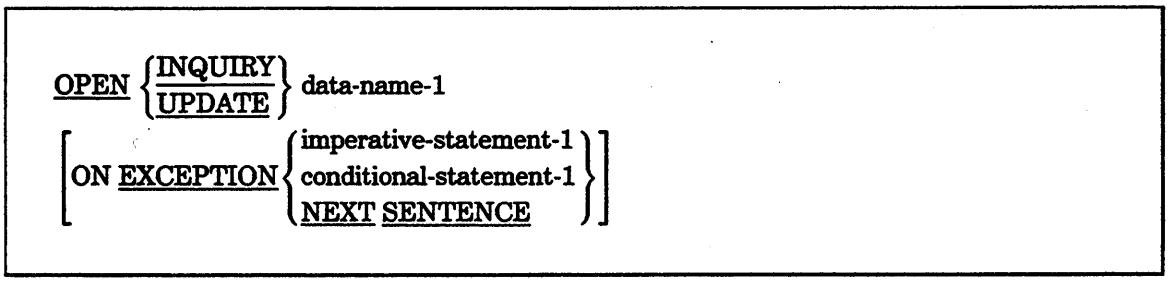
The OPEN statement opens a database for subsequent access and specifies the access mode. You must execute an OPEN statement before the database is first accessed; otherwise, the program terminates at run time with an invalid operator error.

The OPEN statement performs the following steps:

1. Opens an existing database. If files required for invoked structures are not in the system directory, DMSII displays an informative message.
2. Performs an implicit create operation on the RDS.

General Format

The general format of the OPEN statement is as follows:



Explanation of Format Elements

INQUIRY

This phrase enforces read-only access to the database specified by data-name-1. You use this option when you do not want to update the database. The program returns an exception to the following statements if you have opened the database with the INQUIRY phrase:

ASSIGN	GENERATE
BEGIN-TRANSACTION	INSERT
DELETE	REMOVE
END-TRANSACTION	STORE

DMSII does not open any audit files if the OPEN INQUIRY statement has been specified in all programs that are currently accessing the database.

UPDATE

This phrase enables you to modify the database specified by data-name-1. You must specify the UPDATE phrase to use the following statements:

ASSIGN	GENERATE
BEGIN-TRANSACTION	INSERT
DELETE	REMOVE
END-TRANSACTION	STORE

data-name-1

This format element specifies the database to be opened.

ON EXCEPTION

This option handles exceptions. More information on the clause is provided later in this section under "DMSII Exceptions." The program returns an exception if the database is already open or if access to the database is denied because of security. If the program returns an exception, the state of the database remains unchanged.

Examples

The following example illustrates the use of the OPEN statement with the INQUIRY phrase:

```
OPEN INQUIRY DBASE.
```

The following example uses the OPEN statement with the INQUIRY phrase and an ON EXCEPTION option:

```
OPEN INQUIRY MYDB
ON EXCEPTION
    DISPLAY "EXCEPTION OPENING MYDB"
    CALL SYSTEM DMTERMINATE.
```

Later in this section, Example 4–1, "DMSII Sample Program," shows the OPEN statement within the context of a complete program.

RECREATE Statement

The RECREATE statement partially initializes the user work area.

This statement performs the following steps:

1. Frees the current record of the specified data set
2. Reads any specified expression to determine the format of the record to be created
3. Unconditionally sets control items such as links, sets, counts, and data sets to NULL, although no data items are altered

To re-create variable-format records, you must supply the same record type as that supplied on the original CREATE statement. If you do not, the subsequent STORE statement results in a DATAERROR subcategory 4. Refer to the *DMSII Application Programming Guide* for more information.

General Format

The general format of the RECREATE statement is as follows:

```
RECREATE data-name-1 [(expression)]
[ON EXCEPTION {imperative-statement-1
               conditional-statement-1
               NEXT SENTENCE          }]
```

Explanation of Format Elements

data-name-1

This format element is the name of the data set you want to initialize.

(expression)

The expression specifies a variable-format-record. You must use an expression to create a variable-format record; otherwise, the expression must not appear. The program returns an exception if the expression does not represent a valid record type.

ON EXCEPTION

This option handles exceptions. More information on this clause is provided later in this section under "DMSII Exceptions."

Example

The following DASDL description is used by the COBOL74 code in the following example. The description is compiled with the name DBASE.

```
D DATA SET (
  A ALPHA (3);
  N NUMBER (3);
  );
S SET OF D KEY N;
```

The following example uses the RECREATE statement:

```
IDENTIFICATION DIVISION.
PROGRAM-ID. EXAMPLE.
DATA DIVISION.
DATA-BASE SECTION.
DB  DBASE ALL.
PROCEDURE DIVISION.
START-PRG.
    OPEN UPDATE DBASE.
    CREATE D.
    MOVE "ABC" TO A.
    MOVE 1 TO N.
    STORE D.
RECREATE-D.
    RECREATE D.
    ADD 1 TO N.
    STORE D.
    IF N < 500
        GO TO RECREATE-D
    ELSE
        CLOSE DBASE
        STOP RUN.
```

Later in this section, Example 4–1, "DMSII Sample Program," shows the RECREATE statement within the context of a complete program.

REMOVE Statement

The REMOVE statement is similar to the FIND statement except that a found record is locked and then removed from the specified subset.

The REMOVE statement performs the following steps:

1. Frees the current record
2. Alters the current path to point to the record specified by the CURRENT option or by the data set name
3. Locks the record found in step 2.
4. Removes the record from the specified subset

If the program returns an exception after step 2, the current path is invalid.

If the program returns an exception after step 3, the operation terminates, leaving the current path pointing to the record specified by the CURRENT phrase or by the data set name.

When the REMOVE statement is completed, the current paths still refer to the deleted record. As a result, a FIND statement on the current record results in a NOTFOUND exception, although the FIND NEXT and FIND PRIOR statements give valid results.

General Format

The general format of the REMOVE statement is as follows:

```
REMOVE { CURRENT     } FROM data-name-2
       { data-name-1 }
[ ON EXCEPTION { imperative-statement-1  } ]
               { conditional-statement-1 }
               { NEXT SENTENCE           }
```

Explanation of Format Elements

CURRENT

This phrase removes the current record from the subset specified by data-name-2. If you specify this option, the subset must have a valid current record. If the subset does not have a valid current record, the program returns an exception.

data-name-1

This format element identifies the name of record. If you specify data-name-1, the program finds the record located by the current path and removes it from the subset. The program returns an exception if the record is not in the subset.

data-name-2

This format element specifies the subset from which you want to remove a record. Data-name-2 must specify a manual subset of the data set specified by data-name-1.

If the subset is embedded in a data set, the data set must have a current record defined and that record must be locked. If the record is not locked, the program returns an exception.

ON EXCEPTION

This option handles exceptions. More information on this clause is provided later in this section under "DMSII Exceptions." Exceptions are returned if one of the following is true:

- You specify the CURRENT phrase, and the specified subset does not have a valid current record.
- You use the data set name (data-name-2) option, and the record is not in the subset.
- You specify a subset that is embedded in a data set, and the data set does not have a current record defined and locked.

Example

The following DASDL description is used by the COBOL74 code in the following example. The description is compiled with the name DBASE.

```
D DATA SET (
  A ALPHA (3);
  B BOOLEAN;
  N NUMBER (3);
  R REAL;
  );
X SUBSET OF D BIT VECTOR;
```

The following example uses the REMOVE statement:

```
IDENTIFICATION DIVISION.
PROGRAM-ID. EXAMPLE.
DATA DIVISION.
DATA-BASE SECTION.
DB  DBASE ALL.
PROCEDURE DIVISION.
OPEN-DB.
    OPEN UPDATE DBASE.
    SET X TO BEGINNING.
START-PRG.
    FIND NEXT X ON EXCEPTION
        CLOSE DBASE
        STOP RUN.
    IF N > 100
        REMOVE D FROM X.
    GO TO START-PRG.
```

Later in this section, Example 4–1, "DMSII Sample Program," shows the REMOVE statement within the context of a complete program.

SAVE TRANSACTION POINT Statement

The SAVE TRANSACTION POINT statement provides an intermediate transaction point record for audit. The transaction point applies only to the current transaction, and does not affect halt/load recovery. The system completes halt/load recovery at the end of the transaction, not when it encounters a transaction point.

General Format

The format for the SAVE TRANSACTION POINT statement is as follows:

```
SAVE TRANSACTION POINT data-name-1 [arithmetic-expression-1]
[ON EXCEPTION {imperative-statement-1
               conditional-statement-1
               NEXT SENTENCE}]
```

Explanation of Format Elements

data-name-1

This format element is the name of a RDS that identifies the database.

arithmetic-expression-1

This expression indicates a marker to be assigned to the present execution point in the transaction. Arithmetic expressions are discussed in Volume 1.

ON EXCEPTION

This option handles exceptions. An explanation of the ON EXCEPTION option is provided later in this section under "DMSII Exceptions."

Example

The following line of code shows an example of the SAVE TRANSACTION POINT statement:

```
SAVE TRANSACTION POINT MY-RESTART 3.
```

SECURE Statement

The SECURE statement prevents other programs from updating a record by applying a shared lock. It is called a shared lock because other users can read, find, or secure the record; however, they cannot update the record.

You can execute a LOCK statement to upgrade secured records to locked records. If two or more users try to upgrade the records at the same time, however, a deadlock can occur and cause an exception.

General Format

The format for the SECURE statement is as follows:

```
SECURE { selection-expression-1 }
       { data-name-1            }
       { STRUCTURE data-name-2  }
[ ON EXCEPTION { imperative-statement-1  } ]
               { conditional-statement-1 }
               { NEXT SENTENCE           }
```

Explanation of Format Elements

selection-expression-1

This format element specifies the record you want to secure. For more information, see "Using Selection Expressions" earlier in this section.

data-name-1

This format element specifies the global data record you want to secure. If the invoked database contains a remap of the global data, your program uses the name of the logical database, not the name of the global data remap, to lock the global data record.

STRUCTURE

The STRUCTURE phrase secures all records in the structure simultaneously. The structure must be a data set. If other users have locked records in the structure, you must wait until the users free the records or structures or end their transactions before you can secure the structure. A deadlock can occur if other users attempt to lock more records while you are securing the structure.

You cannot free a secured structure by ending the transaction; instead, you must use the FREE statement. More information on this statement is provided in "FREE Statement" earlier in this section.

data-name-2

This format element specifies the name of the data set used in the STRUCTURE phrase.

ON EXCEPTION

This option handles exceptions. See "DMSII Exceptions" for information on this clause.

Example

The following line of code shows an example of the SECURE statement used with the STRUCTURE phrase:

```
SECURE STRUCTURE VENDOR-DATA.
```

SET Statement

The SET statement alters the current path or changes the value of an item in the current record. The SET statement affects only the record area; it does not affect the data set until you execute a subsequent STORE statement.

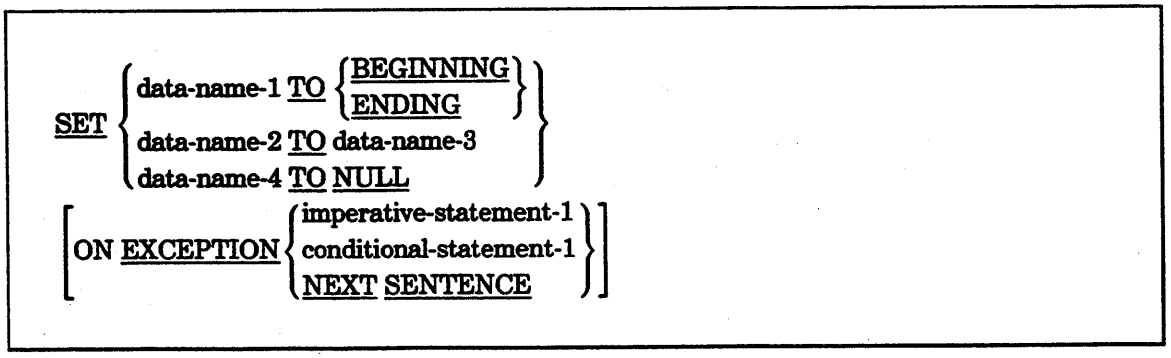
The SET statement performs the following steps:

1. Frees the current path of the data set, set, or subset
2. Performs one of the following:
   - Alters the current path of a data set, set, or subset to point to the beginning or the ending of the respective structure
   - Alters a set or subset path to point to the current path of a data set
   - Assigns a NULL value to a particular item

A FIND NEXT statement appearing after a SET TO BEGINNING statement is equivalent to a FIND FIRST statement. A FIND PRIOR statement appearing after a SET TO ENDING statement is equivalent to a FIND LAST statement.

General Format

The general format of the SET statement is as follows:



Explanation of Format Elements

data-name-1 and BEGINNING or ENDING

Data-name-1 specifies the data set, set, or subset whose current path you want to alter to point to the beginning or ending of the data set.

data-name-2 and data-name-3

These format elements identify the name of a set or subset. Data-name-2 specifies the set or subset whose current path you want to alter to point to the current record of data-name-3.

data-name-4

This format element specifies an item of the current record that is to be assigned a NULL value. Data-name-4 cannot be a link item.

NULL

If you declare a null value in DASDL, that value is used as the NULL value in this statement. Otherwise, the statement uses the system default NULL value.

When the NULL value is applied to a group item, the NULL value defined in DASDL is used, regardless of the NULL specification for any elementary item contained within the group.

ON EXCEPTION

This option handles exceptions. Information on this clause is provided later in this section under "DMSII Exceptions."

Example

The following DASDL description is used by the COBOL74 code in the following example. The description is compiled with the name DBASE.

```
DS DATA SET
    (A ALPHA (20);
     N NUMBER (2)
    );

S SET OF DS
    KEY (A);
SS SUBSET OF DS
    WHERE (N=3);
```

The following example uses the SET statement:

```
FIND S AT A = "ABC".
SET SS TO DS
ON EXCEPTION
   NEXT SENTENCE.
FIND NEXT SS.


SET S TO BEGINNING
ON EXCEPTION
   DISPLAY "NONE".


SET SS TO ENDING
ON EXCEPTION
   DISPLAY "NONE".
```

Later in this section, Example 4–1, "DMSII Sample Program," shows the SET statement within the context of a complete program.

STORE Statement

The STORE statement places a new or modified record into a data set. The statement inserts the data from the user work area for the data set or global record into the data set or global record area.

After you execute a CREATE or RECREATE statement, you can execute a STORE statement to do one of the following:

- Ensure the validity of the data in the user work area if you have specified a VERIFY condition in DASDL.
- Test the record for validity before the STORE statement inserts the record into each set in the data set. For example, the STORE statement can test the record to determine whether or not duplicate values for keys are allowed.
- Evaluate the WHERE condition for each automatic subset.
- Insert the record into all sets and automatic subsets if all conditions are satisfied.
- Lock the new record.
- Alter the data set path to point to the new record.

After you execute a LOCK or MODIFY statement, the STORE statement does the following:

- Checks the data in the user work area for validity if you have specified a VERIFY condition in the DASDL.
- Depending on the condition, performs the following steps:
  - If items involved in the insertion conditions have changed, reevaluates the conditions
  - If the condition yields a FALSE value, removes the record from each automatic subset that contains the record
  - If the condition yields a TRUE value, inserts the record into each automatic subset that does not contain the record
- Deletes and reinserts the record into the proper position if you have modified a key used in ordering a set or automatic subset so that the record must be moved within that set or automatic subset.
- Stores the record in a manual subset, but does not reorder that subset. Your program is responsible for maintaining manual subsets. A subsequent reference to the record using that subset produces undefined results.

General Format

The general format of the STORE statement is as follows:

```
STORE data-name-1
[ ON EXCEPTION { imperative-statement-1   }  ]
               { conditional-statement-1  }
               { NEXT SENTENCE            }
```

Explanation of Format Elements

data-name-1

This format element identifies the name of the data set you want to store. Data-name-1 causes the STORE statement to do one of the following:

- Return the data in the specified data set work area to the data set.
- Return the data in the global data work area to the global data record area.

You must lock the global data record before you execute a STORE statement; otherwise, the program terminates the STORE statement with an exception.

ON EXCEPTION

This option handles exceptions. The program returns an exception and does not store the record if the record does not meet any of the validation conditions. The program also returns an exception if either of the following is true:

- The data set path is valid and the current record is not locked.
- The global data record is not locked.

Information on the ON EXCEPTION option is provided later in this section under "DMSII Exceptions."

Example

The following DASDL description is used by the COBOL74 code in the following example. The description is compiled with the name DBASE.

```
D DATA SET (
  A ALPHA (3);
  N NUMBER (3);
  );
S SET OF D KEY N;
```

The following example uses the STORE statement:

```
IDENTIFICATION DIVISION.
PROGRAM-ID. EXAMPLE.
DATA DIVISION.
DATA-BASE SECTION.
DB  DBASE ALL.
PROCEDURE DIVISION.
START-PRG.
    OPEN UPDATE DBASE.
    CREATE D.
    MOVE "ABC" TO A.
    MOVE 1 TO N.
    STORE D.
RECREATE-D.
    RECREATE D.
    ADD 1 TO N.
    STORE D.
    IF N < 500
        GO TO RECREATE-D
    ELSE
        CLOSE DBASE
        STOP RUN.
```

Later in this section, Example 4–1, "DMSII Sample Program," shows the STORE statement within the context of a complete program.

Processing DMSII Exceptions

During the execution of data management (DM) statements, the program can encounter any one of several exception conditions. Exception conditions prevent an operation from being performed as specified. The conditions result if the program encounters a fault or does not perform the expected actions. For example, execution of the following statement results in an exception if no entry in S has a value of "JONES" for the key item:

```
FIND S AT NAME = "JONES"
```

If the operation terminates normally, the program returns no exception.

The database status word, DMSTATUS, is associated with each COBOL74 program that accesses a database. The value of DMSTATUS indicates whether an exception has occurred and specifies the nature of the exception. The data management structure number attribute, DMSTRUCTURE, can also be helpful in analyzing the results of exception conditions.

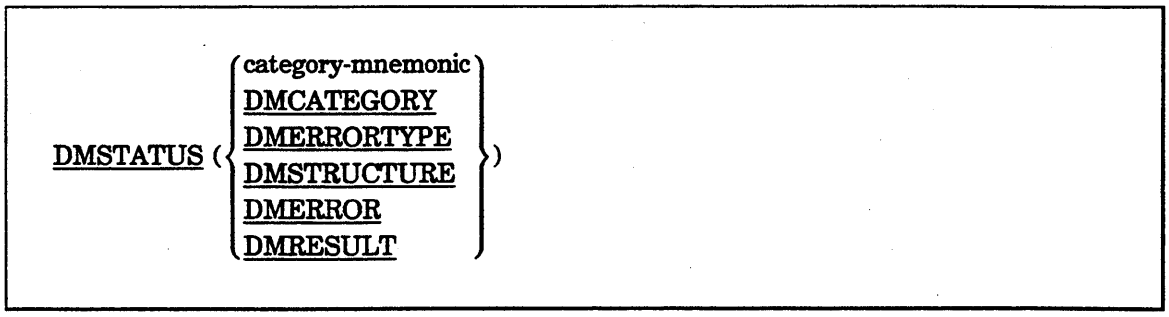
DMSTATUS Database Status Word

The system sets the value of DMSTATUS at the completion of each data management statement. You can use the DMSTATUS entry to discover information about an exception. When interrogating DMSTATUS, you must include an attribute name in parentheses after the DMSTATUS entry.

Information on the exception categories, subcategories, and mnemonics used in exception processing is provided in the *DMSII Application Programming Guide*.

General Format

The general format of the DMSTATUS database status word is as follows:



Explanation of Format Elements

category-mnemonic

This format element yields a TRUE value if the major category specified by category-mnemonic has occurred.

DMCATEGORY

This attribute yields a numeric value identifying the major category.

DMERRORTYPE

This attribute yields a numeric value identifying the subcategory of the major category.

DMSTRUCTURE

This attribute yields a numeric value identifying the structure number of the structure involved in the exception. See "DMSTRUCTURE Structure Number Function" in this section for more information.

DMERROR

This procedure yields a TRUE value if any error has occurred.

DMRESULT

This attribute yields the 48-bit contents of DMSTATUS as a PIC X(6) data item. If no exception has occurred, the program returns six EBCDIC nulls (that is, 48"000000000000").

DMSTRUCTURE Structure Number Function

The DMSTRUCTURE function enables you to programmatically determine the structure number of a data set, set, or subset. The structure numbers of all invoked structures are shown in the invocation information in the program listing. Your program can use the structure number to analyze the results of exception conditions.

The DMSTRUCTURE function is most useful when the previous operation against a data set that is spanned by several data sets yielded an exception. The program can determine from the structure number which structure caused the exception.

When you declare a partitioned structure in DASDL, the structure is assigned one or more structure numbers, depending on the value of the integer in the following option:

```
OPEN PARTITIONS = integer
```

For example, three structure numbers are assigned to the structure when you specify the following option:

```
OPEN PARTITIONS = 3
```

The DMSTRUCTURE function returns the lowest structure number assigned to the structure. However, the value in DMSTRUCTURE can be any of the values assigned by DMSII at run time; it is not necessarily the same value every time.

General Format

The general format for the DMSTRUCTURE function is as follows:

```
data-name-1 (DMSTRUCTURE)
```

Explanation of Format Element

data-name-1

This format element returns the structure number of the data set, set, or subset.

Example

The following example uses the DMSTRUCTURE structure number function:

```
IF D(DMSTRUCTURE) = DMSTATUS(DMSTRUCTURE) DISPLAY "D FAULT".
```

DMSII Exceptions

You can use any of the following methods in your program code to handle exceptions:

- Call the DMERROR Use procedure.
- Specify the ON EXCEPTION option with the data management statement.

If you neither call the DMERROR Use procedure nor specify the ON EXCEPTION option, the program returns an exception and terminates the program with an error. As a result, the values of the DMSTATUS category, subcategory, and structure number are displayed on the system console, placed in the system log, and printed with the job summary output.

An explanation of the DMERROR Use procedure and ON EXCEPTION option are included in this discussion of DMSII exceptions. See Volume 1 for information on the USE statement with Use procedures.

DMERROR Use Procedure

You can use the DMERROR Use procedure to extend the DECLARATIVES SECTION of the PROCEDURE DIVISION to enable you to specify a DMERROR Use procedure. The DMERROR Use procedure yields a TRUE value if any error has occurred.

You can call the DMERROR Use procedure each time the program returns an exception during the execution of a data management (DM) statement, unless the program contains an ON EXCEPTION option for that statement. The DMERROR Use procedure passes control of your program to the statement following the data management statement that encountered the exception.

The DMERROR Use procedure can appear by itself or in any order with other Use procedures in the DECLARATIVES SECTION. However, you can declare only one DMERROR Use procedure in a COBOL74 program. In addition, the DMERROR Use procedure cannot contain GO TO statements that reference labels outside the procedure.

If you use both a DMERROR Use procedure and an ON EXCEPTION option, the ON EXCEPTION option takes precedence, and the DMERROR Use procedure is not executed.

Example

The following example shows the declaration for the DMERROR Use procedure:

```
DECLARATIVES.
DMERR-SECT SECTION.
      USE ON DMERROR.
DMERR-PARA.
      IF DMSTATUS(NOTFOUND)...
END DECLARATIVES.
```

ON EXCEPTION Option

You can place the ON EXCEPTION option after the verb of the following DM statements to handle exceptions:

ABORT-TRANSACTION	INSERT
ASSIGN	LOCK
BEGIN-TRANSACTION	MODIFY
CANCEL TRANSACTION POINT	OPEN
CLOSE	RECREATE
CREATE	REMOVE
DELETE	SAVE TRANSACTION POINT
END-TRANSACTION	SECURE
FIND	SET
FREE	STORE
GENERATE	

The ON EXCEPTION option is also shown in the syntax of these data management statements.

If you use both a DMERROR Use procedure and an ON EXCEPTION option, the ON EXCEPTION option takes precedence, and the DMERROR Use procedure is not executed.

General Format

The general format of the ON EXCEPTION option is as follows:

```
[ ON EXCEPTION { imperative-statement-1     } ]
               { conditional-statement-1    }
               { NEXT SENTENCE              }
```

Explanation of Format Elements

imperative-statement-1, conditional-statement-1, or NEXT SENTENCE

One of these format elements is executed if the program returns an exception. Refer to Volume 1 for information about these statements.

Examples

In the following line of code, a branch to the LBL1 subprogram is executed if a STORE statement encounters an exception:

```
STORE D ON EXCEPTION GO TO LBL1.
```

The following example uses the ON EXCEPTION option and interrogates DMSTATUS:

```
MODIFY S AT X = 3 ON EXCEPTION
     IF DMSTATUS (NOTFOUND) GO NOT-FOUND-L ELSE
     IF DMSTATUS (DEADLOCK) GO DEAD-LOCK-L ELSE
     .
     .
     .
NOT-FOUND-L.
     IF DMSTATUS (DMERRORTYPE) = 1 statement ELSE
     IF DMSTATUS (DMERRORTYPE) = 2 statement ELSE
     .
     .
     .
DEAD-LOCK-L.
     IF DMSTATUS (DMERRORTYPE) = 1 statement ELSE
```

DMSII Sample Program

Example 4–1 shows methods of using DMSII programming interfaces. Examples of coding the DMSTRUCTURE and DMERRORTYPE attributes for the DMSTATUS database status word begin at line 035700. The example shows the use of many verbs discussed in this section; however, the sample program is not designed to demonstrate recommended programming practices. The following verbs are used with the ON EXCEPTION option:

CLOSE	INSERT	SET
DELETE	LOCK	STORE
FIND	REMOVE	

The program accesses the UNIVDB database to create and update a master student file called MSF. The master student file is sorted by the social security number of each student. Updates can be performed to the file such as changing the name or grade of a student. For example, the code for changing grades begins at line 032900. Two subsets are created for students beginning at line 030000. One subset is based on the sex of each student; the other subset contains information on students who have a grade point average greater than 3.5.

```
010000$ SET LINEINFO BDMS
010100 IDENTIFICATION DIVISION.
010200 PROGRAM-ID. UNIVERSITY.
010300 ENVIRONMENT DIVISION.
010400 INPUT-OUTPUT SECTION.
010500 FILE-CONTROL.
010600     SELECT CARD ASSIGN TO READER.
010700     SELECT MONITOR-DMS ASSIGN TO PRINTER.
010800 DATA DIVISION.
010900 FILE SECTION.
011000 FD CARD.
011100 01 CARD-REC.
011200    03 C-TYPE                          PIC 9.
011300    03 C-SSNO                          PIC 9(9).
011400    03 C-GRD-PT-AVG                    PIC 999V99.
011500    03 C-SEX                           PIC X.
011600    03 C-AGE                           PIC 99.
011700    03 C-QTR                           PIC X(4).
011800    03 C-TYPECOURSE                    PIC 9.
011900    03 C-GRADE                         PIC XX.
012000    03 C-TITLE-PAPER                   PIC X(30).
012100    03 C-NAME                          PIC X(24).
012200    03 FILLER                          PIC X.
012300 FD MONITOR-DMS.
012400 01 MONITOR-REC.
012500    03 MONITOR-EXCEPTION               PIC X(4).
012600    03 MONITOR-STATUS-B                PIC X(20).
012700    03 MONITOR-STATUS                  PIC ZZZ9BB.
012800    03 MONITOR-VERB                    PIC X(20).
012900    03 MONITOR-SET                     PIC X(17).
013000    03 MONITOR-STRUCTURE               PIC 999.
013100    03 FILLER                          PIC X(62).
013200
013300 DATA-BASE SECTION.
013400 DB UNIVDB.
013500 01 MSF.
013600 01 ADR.
013700 01 RSTDS.
013800
013900 WORKING-STORAGE SECTION.
014000 77   TOOMANYEXCEPTIONS                PIC 99 COMP.
014100 77   N                                 PIC 9(2).
014200 77   END-OF-DATA-IND                   PIC 9.
014300
014400 PROCEDURE DIVISION.
014500 MAIN-PROC.
014600     OPEN OUTPUT MONITOR-DMS.
014700     MOVE SPACES TO MONITOR-REC.
014800     OPEN UPDATE UNIVDB.
014900     BEGIN-TRANSACTION AUDIT RSTDS.
015000     PERFORM CREATE-ADR THRU CREATE-ADR-EXIT.
```

Example 4–1. DMSII Sample Program

```
015100      PERFORM RECREATE-ADR THRU RECREATE-ADR-EXIT
015200          VARYING N FROM 1 BY 1 UNTIL N = 11.
015300      END-TRANSACTION NO-AUDIT RSTDS.
015400      OPEN INPUT CARD.
015500      MOVE 0 TO END-OF-DATA-IND.
015600      PERFORM READ-CARD-LOOP THRU READ-CARD-LOOP-EXIT
015700          UNTIL END-OF-DATA-IND = 1.
015800      GO TO EOJ.
015900
016000 CREATE-ADR.
016100      CREATE ADR.
016200      MOVE 1 TO FACULTY-STUDENT.
016300      MOVE 1 TO ZIPC.
016400      MOVE 1 TO PHON.
016500      MOVE "FIRST AVE" TO ADLN(1).
016600      ADD 1 TO SNO.
016700      STORE ADR ON EXCEPTION
016800          PERFORM STATUS-BOOLEAN.
016900 CREATE-ADR-EXIT.
017000     EXIT.
017100
017200
017300 RECREATE-ADR.
017400      RECREATE ADR.
017500      ADD 1 TO SNO.
017600      STORE ADR ON EXCEPTION
017700          PERFORM STATUS-BOOLEAN.
017800 RECREATE-ADR-EXIT.
017900     EXIT.
018000*
018100*          Build MSF master student file.
018200*
018300 READ-CARD-LOOP.
018400      READ CARD AT END MOVE 1 TO END-OF-DATA-IND.
018500      WRITE MONITOR-REC FROM CARD-REC.
018600
018700      IF C-TYPE=1
018800         BEGIN-TRANSACTION AUDIT RSTDS
018900         PERFORM 100-CREATE-MSF THRU 100-CREATE-MSF-EXIT
019000         END-TRANSACTION NO-AUDIT RSTDS
019100
019200      ELSE IF C-TYPE=2
019300         BEGIN-TRANSACTION AUDIT RSTDS
019400         PERFORM 200-CREATE-QUARTER THRU 200-CREATE-QUARTER-EXIT
019500         END-TRANSACTION NO-AUDIT RSTDS
019600
019700      ELSE IF C-TYPE=3
019800         BEGIN-TRANSACTION AUDIT RSTDS
019900         PERFORM 300-CREATE-CORSES THRU 300-CREATE-CORSES-EXIT
020000         END-TRANSACTION NO-AUDIT RSTDS
020100
```

Example 4–1. DMSII Sample Program (cont.)

```
020200     ELSE IF C-TYPE=4
020300        PERFORM 400-LINK-MSF-TO-ADR
020400
020500     ELSE IF C-TYPE=5
020600        BEGIN-TRANSACTION AUDIT RSTDS
020700        PERFORM 500-DELETE-ADR THRU 500-DELETE-ADR-EXIT
020800        END-TRANSACTION NO-AUDIT RSTDS
020900
021000     ELSE IF C-TYPE=6
021100        PERFORM 600-GENERATE-SUBSET
021200
021300     ELSE IF C-TYPE=7
021400        BEGIN-TRANSACTION AUDIT RSTDS
021500        PERFORM 700-CHANGE-MSF-NAME THRU 700-CHANGE-MSF-NAME-EXIT
021600        END-TRANSACTION NO-AUDIT RSTDS
021700
021800     ELSE IF C-TYPE=8
021900        BEGIN-TRANSACTION AUDIT RSTDS
022000        PERFORM 800-CHANGE-GRADE THRU 800-CHANGE-GRADE-EXIT
022100        END-TRANSACTION NO-AUDIT RSTDS
022200
022300     ELSE DISPLAY C-TYPE "INVALID CARDTYPE"
022400          MOVE 1 TO END-OF-DATA-IND.
022500 READ-CARD-LOOP-EXIT.
022600     EXIT.
022700
022800 100-CREATE-MSF.
022900     IF C-SSNO < 1 OR > 10
023000         MOVE "C_SSNO COLS 2-10 MUST BE BETWEEN 0 AND 11"
023100            TO MONITOR-REC
023200         WRITE MONITOR-REC
023300         GO TO 100-CREATE-MSF-EXIT.
023400     CREATE MSF.
023500     MOVE C-SSNO TO SSNO.
023600     MOVE C-GRD-PT-AVG TO GRADE-POINT-AVG.
023700     IF C-SEX = "M"
023800         COMPUTE SSEX = TRUE.
023900     MOVE C-AGE TO SAGE.
024000     MOVE C-NAME TO LNAME.
024100     STORE MSF ON EXCEPTION
024200         PERFORM STATUS-BOOLEAN.
024300     IF GRADE-POINT-AVG > 3.50
024400         INSERT MSF INTO SMART ON EXCEPTION
024500             PERFORM STATUS-BOOLEAN.
024600 100-CREATE-MSF-EXIT.
024700     EXIT.
024800
024900 200-CREATE-QUARTER.
025000     LOCK MSFSET AT SSNO = C-SSNO ON EXCEPTION
025100         IF DMSTATUS (NOTFOUND)
025200             DISPLAY C-SSNO "NOT IN MSF"
```

Example 4–1. DMSII Sample Program (cont.)

```
025300              GO TO 200-CREATE-QUARTER-EXIT
025400          ELSE
025500              PERFORM STATUS-BOOLEAN.
025600
025700      CREATE QUARTER.
025800      MOVE C-QTR TO QTR.
025900      STORE QUARTER.
026000
026100 200-CREATE-QUARTER-EXIT.
026200     EXIT.
026300
026400 300-CREATE-CORSES.
026500      LOCK MSFSET AT SSNO = C-SSNO ON EXCEPTION
026600          IF DMSTATUS(NOTFOUND)
026700              DISPLAY C-SSNO "NOT IN MSF"
026800              GO TO 300-CREATE-CORSES-EXIT
026900          ELSE
027000              PERFORM STATUS-BOOLEAN.
027100      LOCK QSET AT QTR = C-QTR.
027200      IF C-TYPECOURSE = 1 CREATE CORSES(1).
027300      IF C-TYPECOURSE = 2 CREATE CORSES(2).
027400      MOVE C-TYPECOURSE TO TYPECOURSE.
027500      IF C-TYPECOURSE = 1
027600          MOVE C-GRADE TO GRADE
027700      ELSE
027800          MOVE C-GRADE TO GGD.
027900      STORE CORSES ON EXCEPTION
028000          PERFORM STATUS-BOOLEAN.
028100 300-CREATE-CORSES-EXIT.
028200     EXIT.
028300
028400 400-LINK-MSF-TO-ADR.
028500     LOCK MSFSET AT SSNO = C-SSNO.
028600     LOCK SSAD AT SNO OF ADR = C-SSNO.
028700     MOVE ADLN(1) TO HOME-ADDRESS.
028800 400-LINK-MSF-TO-ADR-EXIT.
028900     EXIT.
029000
029100 500-DELETE-ADR.
029200     MOVE "LOCK MSFSET" TO MONITOR-VERB.
029300     LOCK MSFSET AT SSNO = C-SSNO ON EXCEPTION
029400         PERFORM STATUS-BOOLEAN.
029500     DELETE ADR ON EXCEPTION
029600         PERFORM STATUS-BOOLEAN.
029700 500-DELETE-ADR-EXIT.
029800     EXIT.
029900
030000 600-GENERATE-SUBSET.
030100     MOVE "GENERATE AND" TO MONITOR-VERB.
030200     GENERATE UTILITY = SMART AND SEXSET.
030300     FIND FIRST UTILITY ON EXCEPTION
```

Example 4–1. DMSII Sample Program (cont.)

```
030400          IF DMSTATUS(NOTFOUND)
030500              DISPLAY "SUBSET EMPTY"
030600              GO TO 600-GENERATE-SUBSET
030700          ELSE
030800              PERFORM STATUS-BOOLEAN.
030900      MOVE LNAME TO MONITOR-REC.
031000      WRITE MONITOR-REC.
031100      MOVE "GENERATE NULL" TO MONITOR-VERB.
031200      GENERATE UTILITY = NULL.
031300 600-GENERATE-SUBSET-EXIT.
031400      EXIT.
031500
031600 700-CHANGE-MSF-NAME.
031700      MODIFY MSFSET AT SSNO = C-SSNO.
031800      MOVE LNAME TO MONITOR-REC.
031900      WRITE MONITOR-REC.
032000      MOVE "NAME IN MSF WAS CHANGED TO" TO MONITOR-REC.
032100      WRITE MONITOR-REC.
032200      MOVE C-NAME TO LNAME.
032300      STORE MSF.
032400      MOVE LNAME TO MONITOR-REC.
032500      WRITE MONITOR-REC.
032600 700-CHANGE-MSF-NAME-EXIT.
032700      EXIT.
032800
032900 800-CHANGE-GRADE.
033000      MOVE "LOCK MSFSET" TO MONITOR-VERB.
033100      LOCK MSFSET AT SSNO = C-SSNO ON EXCEPTION
033200           PERFORM STATUS-BOOLEAN.
033300      MOVE C-GRD-PT-AVG TO GRADE-POINT-AVG.
033400      MOVE "STORE MSF" TO MONITOR-VERB.
033500      STORE MSF ON EXCEPTION
033600           PERFORM STATUS-BOOLEAN.
033700      IF GRADE-POINT-AVG NOT < 3.50
033800          GO TO 800-CHANGE-GRADE-EXIT.
033900      SET SMART TO MSF ON EXCEPTION
034000          PERFORM STATUS-BOOLEAN.
034100      MOVE "REMOVE CURRENT" TO MONITOR-VERB.
034200      REMOVE CURRENT FROM SMART ON EXCEPTION
034300          PERFORM STATUS-BOOLEAN.
034400
034500*
034600*    Set SMART to MSF and remove the current record (MSF).
034700*
034800 800-CHANGE-GRADE-EXIT.
034900      EXIT.
035000
035100 STATUS-BOOLEAN.
035200      ADD 1 TO TOOMANYEXCEPTIONS.
035300      IF TOOMANYEXCEPTIONS > 10
035400          DISPLAY TOOMANYEXCEPTIONS  "IS TOO MANY EXCEPTIONS"
```

Example 4–1. DMSII Sample Program (cont.)

```
035500          GO TO XIT.
035600      MOVE ALL "*" TO MONITOR-EXCEPTION.
035700      MOVE DMSTATUS(DMSTRUCTURE) TO MONITOR-STRUCTURE.
035800      MOVE DMSTATUS(DMERRORTYPE) TO MONITOR-STATUS.
035900      IF DMSTATUS(NOTFOUND)
036000          MOVE "NOTFOUND" TO MONITOR-STATUS-B ELSE
036100      IF DMSTATUS(DUPLICATES)
036200          MOVE "DUPLICATES" TO MONITOR-STATUS-B ELSE
036300      IF DMSTATUS(DEADLOCK)
036400          MOVE "DEADLOCK" TO MONITOR-STATUS-B ELSE
036500      IF DMSTATUS(DATAERROR)
036600          MOVE "DATAERROR" TO MONITOR-STATUS-B ELSE
036700      IF DMSTATUS(NOTLOCKED)
036800          MOVE "NOTLOCKED" TO MONITOR-STATUS-B ELSE
036900      IF DMSTATUS(KEYCHANGED)
037000          MOVE "KEYCHANGED" TO MONITOR-STATUS-B ELSE
037100      IF DMSTATUS(SYSTEMERROR)
037200          MOVE "SYSTEMERROR" TO MONITOR-STATUS-B ELSE
037300      IF DMSTATUS(READONLY)
037400          MOVE "READONLY" TO MONITOR-STATUS-B ELSE
037500      IF DMSTATUS(IOERROR)
037600          MOVE "IOERROR" TO MONITOR-STATUS-B ELSE
037700      IF DMSTATUS(LIMITERROR)
037800          MOVE "LIMITERROR" TO MONITOR-STATUS-B ELSE
037900      IF DMSTATUS(OPENERROR)
038000          MOVE "OPENERROR" TO MONITOR-STATUS-B ELSE
038100      IF DMSTATUS(CLOSEERROR)
038200          MOVE "CLOSEERROR" TO MONITOR-STATUS-B ELSE
038300      IF DMSTATUS(NORECORD)
038400          MOVE "NORECORD" TO MONITOR-STATUS-B ELSE
038500      IF DMSTATUS(INUSE)
038600          MOVE "INUSE" TO MONITOR-STATUS-B ELSE
038700      IF DMSTATUS(AUDITERROR)
038800          MOVE "AUDITERROR" TO MONITOR-STATUS-B ELSE
038900      IF DMSTATUS(ABORT)
039000          MOVE "ABORT" TO MONITOR-STATUS-B ELSE
039100      IF DMSTATUS(SECURITYERROR)
039200          MOVE "SECURITYERROR" TO MONITOR-STATUS-B ELSE
039300      IF DMSTATUS(VERSIONERROR)
039400          MOVE "VERSIONERROR" TO MONITOR-STATUS-B.
039500      WRITE MONITOR-REC.
039600      MOVE SPACES TO MONITOR-REC.
039700 EOJ.
039800      CLOSE UNIVDB ON EXCEPTION
039900          STOP RUN.
040000 XIT.
040100      STOP RUN.
```

Example 4–1. DMSII Sample Program (cont.)

Section 5
Using the DMSII TPS Program Interface

The transaction processing system (TPS) provides users of the Data Management System II (DMSII) with improved processing for a high volume of transactions. To minimize program coding and maintenance, TPS separates the database processing functions into modules and provides a library of transaction do printed procedures.

Typically, there are two types of programs you can write for TPS:

- The application program
- The update library, which is a collection of processing routines that provide an interface between the transaction library and a DMSII database

The COBOL74 program interface developed for TPS includes extensions that enable you to invoke a transaction base, declare and manipulate transaction records, and pass records to procedures in the transaction library.

This section discusses the functions of the program interface and provides program samples that demonstrate a typical application using TPS. The following topics are included:

- Identifying and interpreting the Transaction Formatting Language (TFL) items in COBOL74
- Declaring the transaction base
- Creating transaction records
- Using transaction records to pass variables as parameters and to assign (or copy) the contents of a variable to another transaction record variable
- Accessing transaction record items
- Inquiring about transaction record control items
- Using transaction record compile-time functions to access certain properties of transaction record formats
- Using transaction library entry points to invoke library procedures
- Using the update library to perform data management against the database with statements for processing transactions

The section also provides information about the extensions developed for use with TPS and provides examples and sample programs. For an alphabetized list of the extensions for TPS, refer to "DMSII TPS Extensions" in Section 1, "Introduction to COBOL74 Program Interfaces."

See Also

Refer to the *A Series DMSII Transaction Processing System (TPS) Programming Guide* for information on general programming considerations and concepts, and for detailed information on the Transaction Formatting Language (TFL) used in TPS to describe transaction processing information.

Using the Transaction Formatting Language (TFL)

The basic constructs for the Transaction Formatting Language (TFL) are described in the *DMSII TPS Programming Guide*. However, it is important to know how COBOL74 processes some constructs, and how COBOL74 interprets TFL items.

In TFL, descriptive information about a transaction base is stored in the TRDESCRIPTION file as comments. If the COBOL74 LIST compiler control option is set and the program invokes a transaction base, the TFL comments are listed with each transaction record format and transaction item.

If you are using identifiers in a COBOL74 program, reference these identifiers as they are declared in the TFL source. Note that BDMSCOBOL74 compilers support the hyphenated identifier.

Table 5–1 shows how COBOL74 interprets TFL items in programs that access a transaction base.

Table 5–1. TFL Item Interpretations

TFL Item	COBOL74 Interpretation
<name> ALPHA(n)	<name> PIC X(n) DISPLAY
<name> NUMBER(n)	<name> PIC 9(n) COMP
<name> NUMBER(Sn)	<name> PIC S9(n) COMP
<name> NUMBER(n,m)	<name> PIC 9(n-m)V9(m) COMP
<name> NUMBER(Sn,m)	<name> PIC S9(n-m)V9(m) COMP
<name> REAL	<name> BINARY
<name> REAL(n)	<name> 9(n) BINARY
<name> REAL(Sn)	<name> PIC S9(n) BINARY
<name> REAL(n,m)	<name> PIC 9(n-m)V9(m) BINARY
<name> REAL(Sn,m)	<name> PIC S9(n-m)V9(m) BINARY
<name> BOOLEAN	<name> BOOLEAN
<name> FIELD(n)	<name> FIELD SIZE IS n BITS
<name> GROUP	<name>

Legend

<name>	Declared item name
n, m	Unsigned integers
S	Optional negative sign (–)

Declaring a Transaction Base

You must declare a transaction base before making any references to formats or items defined within that transaction base. The transaction base declaration specifies the transaction base to be invoked. In the declaration, you can do the following:

- Specify only a transaction base to invoke all structures in the transaction base by default.
- Optionally specify a list of transaction record formats and subformats, to invoke only those structures of the transaction base.

The program can designate alternative internal names for the transaction base and for any of the formats or subformats declared. If alternative internal names are used for the base name, subbase name, format name, or subformat name, the program must reference these internal identifiers rather than the TFL source identifiers.

A program can also declare a subbase that has been defined for the transaction base. If a subbase is declared, only the transaction record formats and subformats defined within

that subbase are accessible to the program. The program can specify a list of transaction record formats and subformats, using internal names if needed, that can be invoked from that subbase.

The TRANSACTION SECTION in the DATA DIVISION contains the declarations of all transaction bases that can be referenced in a COBOL74 program. The TRANSACTION SECTION must appear between the DATA-BASE SECTION and the WORKING-STORAGE SECTION.

General Format

The general format for invoking the transaction base is as follows:

```
TRANSACTION SECTION.
[TB[[internal-base-id-1 =] subbase-id] OF base-id [GLOBAL]
  [USING [internal-format-id-1 = ] format-id-1
    [[ ( { ALL
           NONE
           [internal-subformat-id-3 =] subformat-id-3
                [[ , internal-subformat-id-4 =] subformat-id-4 ...] } ) ]
    [ ,[internal-format-id-2 = ] format-id 2
      [[ ( { ALL
             NONE
             [internal-subformat-id-3 =] subformat-id-3
                  [[ , internal-subformat-id-4 =] subformat-id-4 ...] } ) ] ... ]]]]
```

Explanation of Format Elements

TRANSACTION SECTION

This section invokes all transaction bases that can be referenced in a program.

TB

This option specifies the name of the transaction base or transaction subbase to be invoked in the program. All transaction bases that can be referenced by the program must have transaction base declarations.

internal-base-id-1

If internal-base-id-1 is specified, the program must reference this internal-base-id rather than the base-id or subbase-id.

subbase-id

If subbase-id is specified, only the transaction record formats and subformats defined within that subbase are accessible.

base-id

This format element specifies the TFL base name of the transaction base to be invoked.

GLOBAL

This option enables a separately compiled bound procedure to reference the same transaction base declared in the host program.

The transaction base formats and subformats must be declared in the same order in the two programs being bound. The logical identifiers for transaction bases, formats, and subformats also must be the same in the two programs being bound.

Any transaction library entry point that is used in the bound procedure must be declared in the host program. Therefore, a COBOL74 host program must have CALL statements to all the transaction library entry points that are used in the bound procedure.

USING

This option specifies one or more transaction record formats and subformats to be invoked. If this option is used, only the indicated transaction record formats and subformats are invoked. If this option is not used, transaction record formats and subformats are invoked for that transaction base or subbase.

internal-format-id-1 and internal-format-id-2

If an optional internal-format-id is specified, the program must reference the internal ID rather than the format ID.

format-id-1 and format-id-2

These format-ids specify the names of the transaction record format being invoked.

ALL

If the ALL option is used in the subformat list, all the subformats of that format are invoked.

NONE

If the NONE option is used in the subformat list, no subformats of that format are invoked.

internal-subformat-id-1 through 4

If an optional internal-subformat-id is specified, the program must reference these internal subformat IDs rather than the subformat IDs.

subformat-id-1 through 4

If a list of subformat IDs is specified, only those subformats are invoked. If this list is not specified, all subformats of the transaction record format are invoked.

See Also

See "Using Transaction Library Entry Points" later in this section for more information on that topic.

Examples

The following example shows the syntax for declaring a transaction base:

```
TRANSACTION SECTION.
TB  TRB = BANKTR
        USING STATIS=STATUS, CREATEACCT, RESTARTDETANKER,
              DEPOSIT, WITHDRAWAL, PURGEACCT, SYSTEMTR.
```

In the next example, subbase-id SUBB is the transaction record format of the BANKTR base-id. The USING option includes ALL, which specifies that all the subformats are invoked.

```
TRANSACTION SECTION.
TB  TRB = SUBB OF BANKTR
    USING ALL.
```

Creating Transaction Records

A transaction record is an array row that can contain the transaction data of one of the transaction record formats declared in the TFL source. A transaction record variable names one of these array rows. A transaction record variable can contain the transaction data of one of the transaction record formats, in effect becoming a structured variable.

A transaction record variable can be associated with only one transaction base or transaction subbase and can contain only one of the formats or subformats that have been invoked for that particular transaction base or subbase. The size of the array row must be large enough to accommodate the largest of all the formats invoked for it.

The following information explains how transaction record variables are declared, and how transaction records are created.

Declaration of Transaction Record Variables

Each transaction record for a given transaction base is declared using a COBOL74 01-level indicator immediately after the appropriate transaction base entry in the TRANSACTION SECTION. This declaration associates the records with that particular transaction base entry.

Different formats are used to declare variables for the single array row and table array row.

General Formats

The general formats used to declare transaction record variables for a single array row and for a table array row are as follows:

Declaration for a Single Array Row

```
01 transaction-record-id-1.
```

Declaration for a Table Array Row

```
[01 transaction-record-id-2 [OCCURS integer-2 [TIMES]] [GLOBAL] ].
```

Explanation of Format Elements

transaction-record-id-1 and transaction-record-id-2

These identifiers specify a transaction record variable.

OCCURS integer-2

This option declares the number of occurrences of a transaction record. The transaction-record-id can be considered an array of transaction records. You can select a specific transaction record within the transaction-record-id by subscripting.

TIMES

The keyword TIMES is optional.

GLOBAL

A transaction record or transaction record array that is declared in the host program can be declared global in the bound procedure by using this option. The GLOBAL option cannot be specified in the host program; it is allowed only in COBOL74 procedures compiled with the LEVEL option set at 3 or higher. If a transaction base is declared global, the associated transaction records are not made global unless the record syntax also includes the GLOBAL option.

See Also

Refer to "Accessing Transaction Record Items" later in this section for more information about subscripts.

Example

The following example shows declarations for transaction record variables:

```
TRANSACTION SECTION.
TB  TRB = BANKTR
       USING STATIS=STATUS, CREATEACCT, RESTARTDETANKER,
             DEPOSIT, WITHDRAWAL, PURGEACCT, SYSTEMTR.
Ø1  TRIN.
Ø1  TROUT.
Ø1  LASTINPUT.
Ø1  RESTARTTRREC.
Ø1  LASTRESPONSE.
```

Creation of Transaction Record Formats

The contents of a transaction record variable are undefined until you initialize the variable to a particular format by using a CREATE statement. The CREATE statement does the following:

- Assigns to the record variable the initial values of all items in the transaction record format and subformat, if given. The items that are assigned are all declared in TFL.
- Initializes the transaction record control items.

When a format is created, only those items in the common part are assigned initial values. When a subformat is created, the common part items as well as the subformat part items are assigned initial values. The record variable contains the given format until you reinitialize it with a subsequent CREATE statement. It is never cleared by the system.

Once a transaction record variable has been created in a particular format and optional subformat, the items defined within the format and subformat can be accessed and manipulated. If a transaction record is created in a particular format, the record contains only the data items associated with that format. If a transaction record is created in a particular format and subformat, the record contains the data items associated with the format and the data items associated with the subformat.

General Format

The general format of the CREATE statement is as follows:

```
CREATE [subformat-id {OF|IN}] format-id {OF|IN}
transaction-record-id [(subscript)].
```

Explanation of Format Elements

subformat-id

This format element is the name of the subformat in which the initial values of the data items are assigned to the transaction record variable.

format-id

This format element is the name of the format in which the initial values of the data items are assigned to the transaction record variable.

transaction-record-id

This format element identifies the transaction record variable that is initialized to the particular format, and the subformat if specified.

subscript

A subscript is a COBOL74 arithmetic expression that identifies a particular element within a table. If transaction-record-id is declared with the OCCURS option, the subscript must appear.

See Also

- More information on subscripts is provided in "Subscripts" later in this section.
- More information on transaction record variables is provided in "Declaration of Transaction Record Variables" earlier in this section.

Examples

The following examples illustrate the use of the CREATE statement to initialize a variable to a particular format:

```
CREATE RESTARTDETANKER IN RESTARTTRREC.

CREATE CREATEACCT IN TRIN.
```

Using Transaction Records

The compiler enforces certain restrictions on the use of transaction record variables. To avoid syntax errors, you must use transaction record variables as follows:

- To create transaction records and to use compile-time and run-time functions
- To store data in transaction record fields
- To obtain data from transaction record fields
- To pass transaction records as parameters in procedures

Note that transaction record variables cannot be used as follows:

- In a list of multiple destinations
- In input/output statements
- In assignment statements, except as described in "Transaction Record Variable Assignment" later in this section
- As arithmetic, Boolean, or string primaries in expressions

The following information explains how to pass transaction record variables as parameters and how to assign (or copy) the contents of one transaction record variable to another.

Transaction Record Variables as Parameters

In transaction processing, most of the work is carried out by transaction library procedures. Transaction records are passed to these procedures as parameters.

The formal and actual parameters must refer to the same transaction base, but need not specify the same list of transaction formats.

A transaction record is passed as a parameter in the same way that any 01-level entry is passed.

See Also

- For more information on passing an 01-level entry as a parameter, refer to the discussion of the USE statement in Volume 1.
- For coded program examples of parameter passing, see "TPS Programming Examples" later in this section.
- For information on transaction library procedures see "Using Transaction Library Entry Points" later in this section.

Transaction Record Variable Assignment

The contents of a transaction record variable can be assigned, or copied, to another transaction record variable, provided that both variables represent the same transaction base. This assignment is performed by the MOVE statement. Both the control and data portions of the transaction record are transferred when an assignment is performed.

General Format

The general format for the MOVE statement is as follows:

```
MOVE transaction-record-1 TO transaction-record-2.
```

Explanation of Format Elements

transaction-record-1

This format element is a fully subscripted transaction record variable whose contents you are assigning or copying to another transaction record variable.

transaction-record-2

This format element is a fully subscripted transaction record variable that receives its contents from another transaction record variable.

See Also

For more information on the syntax and semantics of the MOVE statement, refer to Volume 1.

Example

The following example assigns a transaction record variable named TRIN to the transaction record variable named TROUT:

```
MOVE TRIN TO TROUT.
```

Accessing Transaction Record Items

A transaction record can contain a transaction of any format or subformat that is declared in the TFL source. Data items in the declared format and subformat of that transaction can be referenced.

Transaction record data items are considered normal data items and can be referenced at any place in the PROCEDURE DIVISION that is acceptable for a normal data item.

The following rules apply to the use of the CORRESPONDING phrase of a MOVE, ADD, or SUBTRACT statement in referencing transaction record items:

- A transaction record name alone is not considered a legitimate group name. Instead, a transaction record format name, a transaction subformat name, or a subordinate group data item must be referenced.
- When a transaction record format is referenced, only the data items in the common portion of the format are considered eligible for use in the CORRESPONDING phrase.

See Also

Refer to Volume 1 for more information on the CORRESPONDING phrase used in these statements.

General Format

The general format for accessing transaction record items is as follows:

```
transaction-record-item-id {IN|OF}
[subformat-id {IN|OF}] [format-id {IN|OF}] transaction-record-id
[(subscript-1 [, subscript-2]...)].
```

Explanation of Format Elements

transaction-record-item-id

This format element identifies a particular transaction format data item that you want to reference.

subformat-id

This format element identifies a particular transaction subformat. To access a data item within a particular transaction record, you can specify subformat-id to qualify the item.

format-id

This format element identifies a particular transaction format. You might need to use format-id as a qualifier to access a data item within a particular transaction record.

transaction-record-id

This format element identifies the transaction record variable that contains the data item to be referenced.

subscript-1 and subscript-2

Subscripts for the transaction-record-id are explained in "Subscripts" later in this section. Coded examples are also provided.

See Also

See "Data Item Qualification" later in this section for more information on data item qualification.

Examples

Following are examples of code used to access transaction record items:

```
MOVE ACCT TO ACCTNUM OF CREATEACCT OF TRIN.
MOVE USER-NAME TO NAME OF CREATEACCT OF TRIN.

MOVE ACCT TO ACCTNUM OF PURGEACCT OF TRIN.

MOVE TIME(1) TO TRANDATE OF DEPOSIT OF TRIN.
MOVE AMT TO AMOUNT OF DEPOSIT OF TRIN.
```

Subscripts

A subscript is a COBOL74 arithmetic expression. A subscript identifies an element declared with an OCCURS clause and within a table. A subscript is required if the transaction-record-id is declared with an OCCURS clause, and this subscript must be specified first if there are several subscripts.

The following must be subscripted:

- Data items of transaction record formats or subformats that are occurring items
- Items embedded within one or more occurring groups
- Occurring items embedded within occurring groups

If there is a subscript for the transaction record, subscripts within a list of transaction record items follow that subscript. The subscripts within a list of transaction record items are listed from left to right, and from the outermost occurring group to the innermost occurring group or occurring items.

A valid subscript for a transaction record item ranges from 1 to the value of the unsigned integer, which must not exceed 4095.

See Also

More information on transaction record variables declared with an OCCURS clause is provided in "Declaration of Transaction Record Variables" earlier in this section.

Example of Subscripting with a TFL Description

Subscripting is illustrated in the following example, which uses a TFL fragment of a transaction record format. The fourth item of C-2, in the second GROUP (B-2), in the seventh GROUP of A is referenced with the subscript triple C-2 (7,2,4). Referencing data item C-1 in the eighth GROUP of B-2, and the first GROUP of A requires the subscript pair C-1 (1,8).

```
 A    GROUP
(
   B-1  REAL;
   B-2  GROUP
   (
     C-1  REAL;
     C-2  NUMBER(6) OCCURS 5 TIMES;
   ) OCCURS 10 TIMES;
) OCCURS 20 TIMES;
```

The following example shows the TFL description in COBOL74 code:

```
04 A OCCURS 20.
   05 B-1 REAL.
   05 B-2 OCCURS 10.
      06 C-1 REAL.
      06 C-2 PIC9(6) COMP OCCURS 5.
```

Example of a Subscript for a Transaction Record

In this example, the third occurrence of the transaction record TR in the fourth item of C-2, in the second GROUP of B-2, in the seventh GROUP of A, can be referenced with the following syntax:

```
C-2  OF TR (3,7,2,4)
```

The following example shows the declaration for the subscripted reference:

```
01 TR OCCURS 10.
   04 A OCCURS 20.
      05 B-1 REAL.
      05 B-2 OCCURS 10.
          06 C-1 REAL.
          06 C-2 PIC(6) COMP OCCURS 5.
```

Data Item Qualification

You can qualify a data item to make it unique. The use of qualification is recommended to assure that the items referenced are the correct data items.

The amount of qualification required to access a data item of a particular transaction record format or subformat varies. In every case, however, the transaction record variable containing the desired data item must be referenced. The following table shows the requirements for qualification:

Item	Qualification Requirement
Data item	If the name of the desired data item is unique with respect to data items of other invoked formats, then only the data item name need be specified. For example: `DATAITEMNAME OF TRANREC`
Format name and data item name	If the name of the desired data item is not unique with respect to data items of other invoked formats, but is unique to the format that contains it, both the format and data item names are needed for qualification. For example: `DATAITEMNAME OF FORMATNAME IN TRANREC`
Subformat name and data item name	Both the subformat name and the data item name are needed for qualification when all the following conditions prevail: • The name of the desired data item is not unique with respect to the common portion of another invoked format. • The data item is contained within a subformat. • Another data item within a different subformat of the same format has the same name as the desired data item. Also, if the data item is contained within a subformat whose name is unique to all invoked formats and subformats, but is not unique with respect to a subformat of another format, both the subformat name and data item names are needed for qualification. For example: `DATAITEMNAME OF SUBFORMATNAME IN TRANREC`
Format name, subformat name, and data item name	The format name, subformat name, and data item name are needed for qualification when all the following conditions prevail: • The name of the desired item is not unique with respect to a subformat of another invoked format. • The name of the desired data item is not unique with respect to the format that contains it. • The name of the subformat that contains the desired data item is not unique with respect to all invoked formats and subformats. For example: `DATAITEMNAME OF SUBFORMATNAME OF FORMATNAME IN TRANREC`

Inquiring about Transaction Record Control Items

Control items are system-defined items contained in every transaction record and can be inquired about at run time. These items are maintained by TPS and are read-only in all BDMSCOBOL74 programs. The initial values of these control items are assigned when a transaction record is created. After a transaction record has been created with a particular format and optional subformat, the control items are defined.

Additional information and a description of the record control items are provided in the *DMSII TPS Programming Guide*.

General Format

The general format used to inquire about transaction control record items is as follows:

```
transaction-control-item (transaction-record-id [(subscript)])
```

Explanation of Format Elements

transaction-control-item

This format element is the name of the transaction control record about which you are inquiring.

transaction-record-id

This format element identifies a transaction record variable. It must be fully subscripted if a transaction record array element is used.

subscript

A subscript is a COBOL74 arithmetic expression that identifies a particular transaction record variable within an array of transaction record variables. If transaction-record-id is declared with the OCCURS clause, the subscript must appear.

Example

The following example shows code used to inquire about transaction control record items:

```
READP.
******
      .
      .
      .
     MOVE "F/B/O =" TO RMT-1.
     MOVE TRFILENUM(TRIN) TO RMT-2.
     MOVE TRBLOCKNUM(TRIN)  TO RMT-3.
     MOVE TROFFSET(TRIN)  TO RMT-4.
     MOVE TRFORMAT(TRIN)  TO RMT-5.
     MOVE TRSUBFORMAT(TRIN)  TO RMT-6.
```

Using Transaction Compile-Time Functions

Transaction compile-time functions provide access to certain properties of transaction record formats that are constant at compile time. These compile-time constructs are particularly useful for coding an update library. However, they are not used in application programs.

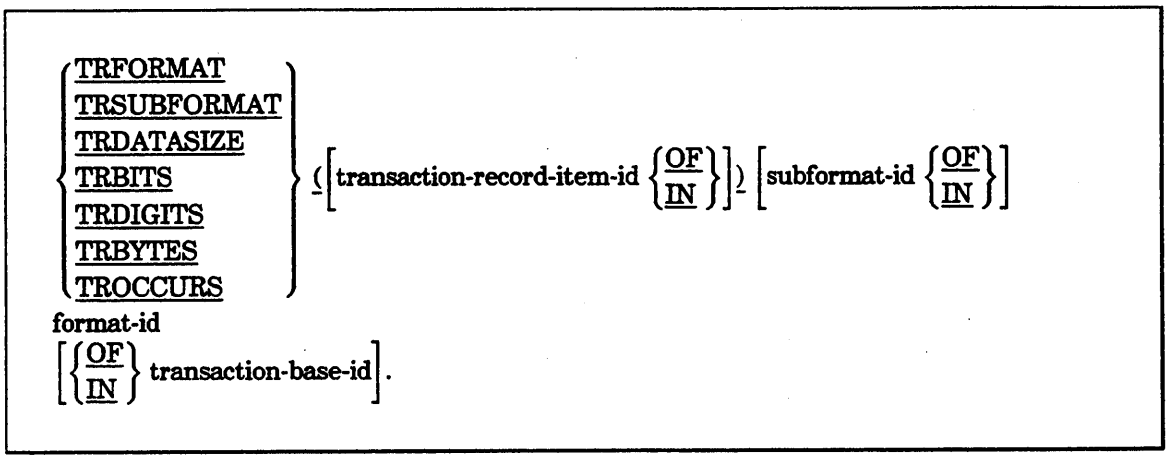
The compile-time functions are discussed in detail in the *DMSII TPS Programming Guide*. The format shown in the following text provides the syntax you use to access the functions.

See Also

For examples of using compile-time functions, refer to "Update Library Programming Conventions" later in this section.

General Format

The general format for using compile-time functions is as follows:



Explanation of Format Elements

transaction-record-item-id

This format element is the name of a data item contained within an invoked transaction format or subformat.

subformat-id

This format element is the name of a transaction subformat that has been invoked within the program.

format-id

This format element is the name of a transaction format that has been invoked within the program.

transaction-base-id

This format element is the name of a transaction base that has been invoked within the program.

Considerations for Use

Table 5–2 includes an explanation of the purpose of each compile-time function when it is used with a specific argument. You must reference transaction-base-id only when transaction base qualification is required. Note that OF and IN are synonyms; the syntax is shown with only the OF construct.

Table 5–2. Using Compile-Time Functions

Function	Explanation
TRFORMAT format-id	Returns the numeric value assigned to the format-id.
TRSUBFORMAT subformat-id OF format-id	Returns the numeric value assigned to the subformat-id.
TRDATASIZE format-id	Returns the size, in bytes, of the common portion of the transaction record.
TRDATASIZE subformat-id OF format-id	Returns the size, in bytes, of the subformat record.
TRBITS transaction-record-item-id OF format-id	Returns the size, in bits, of the referenced data item.
TRBITS transaction-record-item-id OF subformat-id OF format-id	Returns the size, in bits, of the referenced data items.
TRDIGITS transaction-record-item-id OF format-id	Returns the size, in digits, of the referenced data item.
TRDIGITS transaction-record-item-id OF subformat-id OF format-id	Returns the size, in digits, of the referenced data items.
TRBYTES transaction-record-item-id OF format-id	Returns the size, in bytes, of the referenced data item.
TRBYTES transaction-record-item-id OF subformat-id OF format-id	Returns the size, in bytes, of the referenced data items.
TROCCURS transaction-record-item-id OF format-id	Returns the maximum number of occurrences of the data item. This value is 0 (zero) if the item has no OCCURS clause in the TFL source.
TROCCURS transaction-record-item-id OF subformat-id OF format-id	Returns the maximum number of occurrences of the data items. This value is 0 (zero) if an item has no OCCURS clause in the TFL source.

Using Transaction Library Entry Points

The transaction library is a collection of procedures that are accessed by application programs to process or store transactions and to read the transactions back from transaction journal files. The transaction library is tailored for a particular transaction base during compilation. The library performs functions such as

- Calling the update library to process a transaction against a database
- Saving transaction records in transaction journal files
- Automatically reprocessing transactions backed out by DMSII recovery

The transaction library supplies a set of entry points that allow you to invoke the transaction library procedures. Each entry point returns a nonzero result as the value of the procedure if the library detects an exception condition. The value returned can be examined to determine the cause of the exception.

The following information explains how to pass a numeric item to an integer parameter of an entry point and a display item to a string parameter of an entry point. It also discusses the TPS entry points and the CALL statement syntax used to access the entry points.

See Also

For detailed information on transaction library entry points, refer to the *DMSII TPS Programming Guide*.

Parameter Passing with COBOL74 Constructs

The program interface provides the means by which you can use the entry points of the transaction library. The interface provides two methods for passing a numeric item to an integer parameter of an ALGOL library entry point; additionally, the interface provides one method for passing a display item to a string parameter of an entry point.

To pass a numeric item to an ALGOL integer parameter, you use one of the following two methods:

- Use the INTEGER function, which converts an item declared with USAGE COMPUTATIONAL to an ALGOL integer representation.
- Specify the BINARY option in the USAGE clause of the item declaration. When the BINARY option is specified, the item is represented in binary form, which is the same as an ALGOL integer representation.

In addition, you can pass an item declared with the USAGE DISPLAY clause to an ALGOL string parameter of a library entry point by using the STRING function to convert the item to an ALGOL string representation.

See Also

- For more information regarding the STRING and INTEGER functions, refer to Volume 1.
- For information on how integer parameters must be declared so that an update library can receive them from transaction library entry points, refer to "Parameter Passing to an Update Library" later in this section.

General Format

The general format for the INTEGER and STRING functions is as follows:

```
INTEGER ( COBOL74-identifier )
STRING ( COBOL74-identifier )
```

Explanation of Format Elements

INTEGER (COBOL74-identifier)

This function converts the COMP item specified by the identifier to an ALGOL integer representation. The identifier must be a COMP item.

STRING (COBOL74-identifier)

This function converts the DISPLAY item specified by the identifier to an ALGOL string representation so that the item can be passed to an ALGOL string parameter of a library entry point.

Examples of Using the INTEGER Function to Pass Parameters

The INTEGER function can be used in both the USING phrase and the GIVING phrase of the CALL statement. The function performs the data conversion on both the call to and the return from the library entry point. Refer to Volume 1 for the syntax of the CALL statement used with libraries.

```
CALL......USING INTEGER(VARNAME),....
GIVING INTEGER(RESULT).
```

If VARNAME and RESULT are COMP items declared with the BINARY option in the USAGE clause, the previous example is changed to

```
CALL......USING VARNAME,......
GIVING RESULT.
```

Example of Using the STRING Function to Pass Parameters

The STRING function can be used in the USING phrase of the CALL statement. This function performs the data conversion on both the call to and the return from the library entry point.

```
CALL......USING INTEGER(VARNAME-1),
STRING(DISPLAYVAR) GIVING INTEGER(VARNAME-2).
```

TPS Entry Points in COBOL74

To call TPS entry points, use the standard CALL statement for use with libraries. Refer to Volume 1 for the syntax of the CALL statement. For detailed information on the entry points and the related parameters used with them, refer to the *DMSII TPS Programming Guide*.

Example

The following example uses the CALL statement with a TPS entry point. Refer to the TPS sample programs at the end of this section for more examples of the syntax used to call the entry points.

```
PROCESSTRANSACTION.
    CALL "PROCESSTRANSACTION  "
       USING IDNUM, TRIN, TROUT,
        RESTARTTRREC
       GIVING TPS-RESULT.
```

Table 5–3 includes the TPS entry points and their parameters.

Table 5–3. TPS Entry Points

Entry Point	Parameters
CLOSETRBASE	(No parameters)
CREATETRUSER	ID-X, IDNUM
HANDLESTATISTICS	STATOPTION
LOGOFFTRUSER	IDNUM
LOGONTRUSER	ID-X, IDNUM
OPENTRBASE	OPT, TIMEOUTV
PROCESSTRANSACTION	IDNUM, TRIN, TROUT, RESTARTTRREC
PROCESSTRFROMTANK	IDNUM, TRIN, RESTARTNUM, RESTARTTR
PROCESSTRNORESTART	IDNUM, TRIN, TROUT
PURGETRUSER	IDNUM
READTRANSACTION	TRIN
RETURNLASTADDRESS	FILENUM, BLOCKNUM, OFFSET, IDNUM
RETURNLASTRESPONSE	IDNUM, TROUT
RETURNRESTARTINFO	IDNUM, TROUT
SEEKTRANSACTION	FILENUMBER, BLOCKNUMBER, OFFSETS
SWITCHTRFILE	(No parameters)

continued

Table 5–3. TPS Entry Points (cont.)

Entry Point	Parameters
TANKTRANSACTION	IDNUM, TRIN, RESTARTTR
TANKTRNORESTART	IDNUM, TRIN
TRUSERIDSTRING	IDSTRING, IDNUM

Using the Update Library

The update library is a collection of user-written transaction processing routines that serve as an interface between the transaction library and a DMSII database. The update library is the only user-written module within TPS that contains the database declaration and all the code that performs data management statements against the database.

The following paragraphs explain the conventions used to program the update library, describes methods used to declare transaction Use procedures, and covers methods used to declare ALGOL integer parameters so that they are passed to an update library from TPS entry points. The following text also describes the transaction processing statements that are extensions developed for the TPS program interface.

See Also

For information on TPS entry points, refer to "Using Transaction Library Entry Points" earlier in this section.

Update Library Programming Conventions

Although the update library is user-written, it must conform to conventions established by TPS so that it can be accessed by the transaction library.

The update library must provide one entry point that makes it accessible to the transaction library. The name of this entry point is ACCESSDATABASE. In a COBOL74 program, ACCESSDATABASE must be the program-name defined in the IDENTIFICATION DIVISION.

The ACCESSDATABASE entry point accepts certain parameters, which are explained in detail in the *DMSII TPS Programming Guide*.

See Also

- For information on programming the update library, refer to the *DMSII TPS Programming Guide.*
- For an example of an update library, refer to "TPS Programming Examples" later in this section.

Declaration of the Transaction Use Procedures

COBOL74 programs intended for the update library can declare as parameters untyped, parameterless procedures called SAVEINPUTTR and SAVERESPONSTR. To declare the procedures, you must name them in the USING clause of the PROCEDURE DIVISION header and declare them as section headers in the DECLARATIVES. You can use the CALL statement to invoke these procedures. The ability to pass procedures as parameters is needed because the update library must accept two procedures as input.

General Format

The general format of the section header for a transaction Use procedure is as follows:

```
section-name SECTION. USE AS [GLOBAL] TRANSACTION PROCEDURE .
```

Explanation of Format Element

GLOBAL

A transaction record or transaction record array that is declared in the host program can be declared global in the bound procedure, by using the GLOBAL option. The names of the procedures must be the same in the bound procedure and in the host program.

See Also

For more information on the USE statement syntax, semantics, and general rules, refer to Volume 1.

Examples of a Transaction Use Procedure Declaration

The following example shows the section header for the declaration of a transaction Use procedure:

```
PROCEDURE DIVISION USING TR1, TR2.
DECLARATIVES.
TR1 SECTION. USE AS TRANSACTION PROCEDURE.
TR2 SECTION. USE AS TRANSACTION PROCEDURE.
END DECLARATIVES.
PI. CALL TR1.
    CALL TR2.
    .
    .
    .
```

The following example uses the declaration for a bound procedure:

```
DECLARATIVES.
TR1 SECTION.   USE AS GLOBAL TRANSACTION PROCEDURE.
TR2 SECTION.   USE AS GLOBAL TRANSACTION PROCEDURE.
END DECLARATIVES.
```

Parameter Passing to an Update Library

For an update library written in COBOL74 to receive integer parameters from transaction library entry points (written in ALGOL), the parameters must be declared in the LINKAGE SECTION. The following are the two methods of declaring the parameters; also included are examples of the parameters used in the PROCEDURE DIVISION header.

- Declare the parameters in the LINKAGE SECTION as follows:

  ```
  77  FUNCTIONFLAG COMP PIC 9(8)
  ```

  In the PROCEDURE DIVISION, the parameters are used as follows:

  ```
  PROCEDURE DIVISION USING INTEGER(FUNCTIONFLAG),
  ```

- Declare the parameters in the LINKAGE SECTION as follows:

  ```
  77  BINARYFUNCTIONFLAG BINARY PIC 9(8)
  ```

  In the PROCEDURE DIVISION, the parameters are used as follows:

  ```
  PROCEDURE DIVISION USING BINARYFUNCTIONFLAG,
  ```

See Also

- For information about the update library, see "Using the Update Library" earlier in this section.
- For examples of an update library written in COBOL74, refer to "TPS Programming Examples" later in this section.

Transaction Processing Statements

Generally, you use the DMSII program interface statements for programming the update library in TPS. There are some extensions to these statements, however, that have been developed for the TPS program interface. The following extensions are required for the update library to synchronize TPS recovery with DMSII recovery:

- The TRUPDATE option for the OPEN statement
- Optional extensions to the BEGIN-TRANSACTION and END-TRANSACTION statements
- The MID-TRANSACTION statement

See Also

- For information on other statements used with DMSII and information on handling exceptions, refer to Section 4, "Using the DMSII Program Interface."
- For information about exception categories that are specific to the TPS transaction processing extensions, see the *DMSII TPS Programming Guide*.

OPEN Statement

The OPEN statement opens a database for subsequent access and specifies the access mode. It also performs an explicit CREATE statement on the restart data set (RDS).

The TRUPDATE option for the OPEN statement must be used in the update library.

General Format

The general format of the OPEN statement is as follows:

```
OPEN [ { INQUIRY  } ] data-name-1
     [ { TRUPDATE } ]
[ ON EXCEPTION { imperative-statement-1   } ] .
               { conditional-statement-1  }
               { NEXT SENTENCE            }
```

Explanation of Format Elements

INQUIRY

This option enforces read-only access to the database. This option is specified when no update operations are to be performed on the database. If the INQUIRY option is specified, the use of the BEGIN-TRANSACTION, MID-TRANSACTION, and END-TRANSACTION statements is not allowed.

TRUPDATE

This option opens the database for subsequent access.

data-name-1

This format element designates the name of the database to be opened.

ON EXCEPTION

This option handles exceptions. The program returns an exception if the following statements are used when the database is opened with the INQUIRY option:

ASSIGN	END-TRANSACTION	BEGIN-TRANSACTION
GENERATE	REMOVE	MID-TRANSACTION
DELETE	INSERT	STORE

The program also returns an exception if the database is already open. If an exception is returned, the state of the database is unchanged.

See Also

Refer to "Processing DMSII Exceptions" in Section 4, "Using the DMSII Program Interface," for information on the ON EXCEPTION option.

BEGIN-TRANSACTION Statement

The BEGIN-TRANSACTION statement places a program in transaction state. This statement is used only with audited databases. The statement must be able to pass a transaction record as well as pass parameters to procedures in the Accessroutines.

General Format

The general format of the BEGIN-TRANSACTION statement is as follows:

```
BEGIN-TRANSACTION (transaction-record-variable-id)
restart-data-set-name [ ON EXCEPTION { imperative-statement-1 } ] .
                                     { conditional-statement-1 }
                                     { NEXT SENTENCE           }
```

Explanation of Format Elements

transaction-record-variable-id

This identifier is the name of the formal input transaction record variable.

restart-data-set-name

This format element is the name of the RDS. The RDS is declared in the TFL source as parameters.

ON EXCEPTION

This option handles exceptions. The program returns an exception if the database is already open. If an exception is returned, the state of the database is unchanged.

See Also

Refer to "Processing DMSII Exceptions" in Section 4, "Using the DMSII Program Interface," for information on the ON EXCEPTION option.

MID-TRANSACTION Statement

The MID-TRANSACTION statement causes the compiler to generate calls on the given procedure immediately before the call on the DMSII procedure in the Accessroutines. This statement must be able to pass a transaction record as well as present parameters to procedures in the Accessroutines. The MID-TRANSACTION statement is used only with audited databases.

General Format

The general format of the MID-TRANSACTION statement is as follows:

```
MID-TRANSACTION (transaction-record-variable-id,
saveinputtr-procedure-identifier)
[ ON EXCEPTION { imperative-statement-1 }
               { conditional-statement-1 } ] .
               { NEXT SENTENCE           }
```

Explanation of Format Elements

transaction-record-variable-id

This identifier is the name of the formal transaction record.

saveinputtr-procedure-identifier

This format element is an identifier for the SAVEINPUTTR procedure. SAVEINPUT is the name of the SAVEINPUTTR formal procedure. SAVEINPUTTR is an untyped formal procedure with no parameters. The compiler generates a call on a particular procedure immediately before the call on the DMSII procedure in the Accessroutines.

ON EXCEPTION

This option handles exceptions. The program returns an exception if the database is already open. If an exception is returned, the state of the database is unchanged.

See Also

Refer to "Processing DMSII Exceptions" in Section 4, "Using the DMSII Program Interface," for information on the ON EXCEPTION option.

END-TRANSACTION Statement

The END-TRANSACTION statement takes a program out of transaction state. This statement must be able to pass a transaction record and present parameters to procedures in the Accessroutines.

General Format

The general format of the END-TRANSACTION statement is as follows:

```
END-TRANSACTION (transaction-record-variable-id,
saveresponsetr-procedure-identifier) restart-data-set-name
[SYNC] [ ON EXCEPTION { imperative-statement-1 }
                      { conditional-statement-1 } ] .
                      { NEXT SENTENCE           }
```

Explanation of Format Elements

transaction-record-variable-id

This identifier is the name of the formal input transaction record.

saveresponsetr-procedure-identifier

This format element is an identifier for the SAVERESPONSETR procedure. SAVERESPONSE is the name of the SAVERESPONSETR formal procedure. SAVERESPONSETR is an untyped formal procedure with no parameters. The compiler generates a call on a particular procedure immediately before the call on the DMSII procedure in the Accessroutines.

restart-data-set-name

This format element is the name of the RDS.

SYNC

This option forces a syncpoint.

ON EXCEPTION

This option handles exceptions. The program returns an exception if the database is already open. If an exception is returned, the state of the database is unchanged.

See Also

Refer to "Processing DMSII Exceptions" in Section 4, "Using the DMSII Program Interface," for information on the ON EXCEPTION option.

TPS Programming Examples

The following two programming examples illustrate the use of COBOL74 in programs for DMSII TPS. The first example is a complete COBOL74 program with related DASDL and TFL descriptions, and an update library. The second programming example includes the declarations for the transaction base and library entry points for a COBOL74 program.

DMSII TPS Complete Programming Example

This first programming example is a simple banking application in which bank accounts are created and deleted, deposits and withdrawals are made, and account balances are determined. The example includes the user-supplied DASDL and TFL descriptions, update library, and COBOL74 program. The update library as well as the user-written program are written in COBOL74.

DASDL Description

Example 5–1 shows the DASDL description for the COBOL74 banking transaction program (Example 5–4).

```
OPTIONS(AUDIT);
PARAMETERS(SYNCPOINT = 10 TRANSACTIONS);

ACCOUNT DATA SET            % SPECIFY A DATA SET TO HOLD THE ACCOUNT
(                           % NUMBERS AND INFO. ASSOCIATED WITH THEM.
    ACCOUNT-NUM NUMBER(6);
    NAME ALPHA(20);
    BALANCE REAL(S10,2);

    DEPOSIT UNORDERED DATA SET  % USED TO KEEP HISTORY OF THE DEPOSITS
    (                           % AND WITHDRAWALS MADE.
        TRANDATE REAL;
        OLD-BALANCE REAL(S10,2);
        AMOUNT REAL(S10,2);     % NEGATIVE FOR WITHDRAWAL
        NEW-BALANCE REAL(S10,2);
    );

);

ACCOUNT-SET SET OF ACCOUNT
    KEY ACCOUNT-NUM;

RDS RESTART DATA SET          % REMEMBER: MUST SPECIFY A RESTART DATASET
(
    X ALPHA(10);
);
```

Example 5–1. DASDL Description for Banking Transaction Program

TFL Description

Example 5–2 shows the TFL description for the banking transaction program in Example 5–4.

```
BANKTR TRANSACTION BASE;     % FIRST DECLARE THE NAME OF THE TRANSACTION
                             % BASE WE ARE ABOUT TO DESCRIBE.
PARAMETERS
(
    STATISTICS,
    DATABASE = BANKDB ON DISK,
    RESTARTDATASET = RDS,
    HOSTSYSTEM = SYS456
);

DEFAULTS          % HERE WE SPECIFY DEFAULTS FOR ITEMS OF TRANSACTION
                  % FORMATS AND FOR JOURNAL CONTROL AND DATA FILES.
(
    ALPHA (INITIALVALUE = BLANKS),
    BOOLEAN (INITIALVALUE = FALSE),
    NUMBER (INITIALVALUE = 0),
    REAL (INITIALVALUE = 0),
    CONTROL FILE
    (
        AREAS = 100,
        AREASIZE = 100 BLOCKS,
        BLOCKSIZE = 20 SEGMENTS,
        FAMILY = DISK,
        CHECKSUM = TRUE
    ),
    DATA FILE
    (
        AREAS = 100,
        AREASIZE = 100 BLOCKS,
        BLOCKSIZE = 30 SEGMENTS,
        FAMILY = DISK,
        CHECKSUM = TRUE
    )
);

CREATEACCT TRANSACTION FORMAT      % THE FOLLOWING FORMATS ARE
(                                  % USED IN THE APPLICATION
    ACCTNUM NUMBER(6);             % PROGRAM AND THE UPDATE
    NAME ALPHA(20);                % LIBRARY.
);

PURGEACCT TRANSACTION FORMAT
(
    ACCTNUM NUMBER(6);
);
DEPOSIT TRANSACTION FORMAT
(
```

Example 5–2. TFL Description for Banking Transaction Program

```
    ACCTNUM NUMBER(6);
    TRANDATE REAL;
    AMOUNT REAL(10,2);
);

WITHDRAWAL TRANSACTION FORMAT
(
   ACCTNUM NUMBER(6);
    AMOUNT REAL(10,2);
    TRANDATE REAL;
);


STATUS TRANSACTION FORMAT
(
    ACCTNUM NUMBER(6);
    BALANCE REAL(S10,2);
    G GROUP
    (  A ALPHA(20);
       B REAL; );
);

RESTARTDETANKER TRANSACTION FORMAT   % THIS FORMAT ILLUSTRATES POSSIBLE
(                                    % INFORMATION TO BE KEPT IN A
    TANKFILENUM FIELD(14);           % RESTART TRANSACTION RECORD.
    TANKBLOCKNUM FIELD(32);
    TANKOFFSET FIELD(16);
);

MANAGER TRANSACTION SUBBASE     % EXAMPLE SUBBASE THAT A MANAGER MIGHT
(                               % USE. NOTE THAT A GUARDFILE IS ATTACHED
    CREATEACCT,                 % TO IT FOR SECURITY.
    PURGEACCT,
    DEPOSIT,
    WITHDRAWAL,
    STATUS,
),
    GUARDFILE = BANKTR/MANAGER/GUARDFILE;

TELLER TRANSACTION SUBBASE     % EXAMPLE SUBBASE THAT A TELLER MIGHT USE
(
    DEPOSIT,
    WITHDRAWAL,
    STATUS
);

TRHISTORY TRANSACTION JOURNAL  % THIS IS AN EXAMPLE OF SPECIFYING
CONTROL FILE                   % EXPLICIT VALUES FOR THE ATTRIBUTES OF
(                              % THE TRHISTORY JOURNAL.
    AREAS = 100,
    AREASIZE = 100 BLOCKS,
```

Example 5–2. TFL Description for Banking Transaction Program (cont.)

```
    BLOCKSIZE = 20 SEGMENTS,
    FAMILY = DISK,
    CHECKSUM = TRUE
),
DATA FILE
(
    AREAS = 100,
    AREASIZE = 2 BLOCKS,
    BLOCKSIZE = 3 SEGMENTS,
    FAMILY = DISK,
    CHECKSUM = TRUE
);

TANK1 TRANSACTION JOURNAL      % EXAMPLE OF TANK JOURNAL ATTRIBUTE
CONTROL FILE                   % SPECIFICATION.
(
    USERCODE = SAMPLEUSER,
    FAMILY = PACK
),
DATA FILE
(
    USERCODE = SAMPLEUSER,
    DUPLICATED ON DISK
);
```

Example 5–2. TFL Description for Banking Transaction Program (cont.)

Update Library

Example 5–3 shows an update library capable of maintaining database consistency. Refer to "Using the Update Library" earlier in this section for more information about programming update libraries.

```
000100$SET SHARING = PRIVATE
000200$SET TEMPORARY
000300 IDENTIFICATION DIVISION.
000400 PROGRAM-ID. ACCESSDATABASE.
000500 ENVIRONMENT DIVISION.
000600 DATA DIVISION.
000700 DATA-BASE SECTION.
000800     DB DBASE = BANKDB ALL.
000900 TRANSACTION SECTION.
001000 TB  TRB = BANKTR
001100      USING STATIS = STATUS, CREATEACCT, RESTARTDETANKER, DEPOSIT,
001200      WITHDRAWAL, PURGEACCT, SYSTEMTR.
001300      01  TRIN.
001400      01  TROUT.
001500*
001600 WORKING-STORAGE SECTION.
001700 77  FUNCTIONFLAG    COMP    PIC 9(8).
```

Example 5–3. Update Library for Banking Transaction Program

```
001800 77  ACCT            COMP    PIC 9(6).
001900 77  OLDBAL          COMP    PIC S9(8)V99.
002000 77  NEWBAL          COMP    PIC S9(8)V99.
002100*
002200*****************************************************************
002300*
002400 PROCEDURE DIVISION USING FUNCTIONFLAG,TRIN,TROUT,SAVEINPUT,
002500      SAVERESPONSE.
002600 DECLARATIVES.
002700 SAVEINPUT SECTION. USE AS TRANSACTION PROCEDURE.
002800 SAVERESPONSE SECTION. USE AS TRANSACTION PROCEDURE.
002900 END DECLARATIVES.
003000*
003100 MAIN SECTION.
003200 P1.
003300      IF FUNCTIONFLAG EQUAL 1  PERFORM OPENUPDATE     ELSE
003400      IF FUNCTIONFLAG EQUAL 2  PERFORM OPENINQUIRY    ELSE
003500      IF FUNCTIONFLAG EQUAL 3  PERFORM UPDATEDB       ELSE
003600      IF FUNCTIONFLAG EQUAL 4  PERFORM FORCEABORT     ELSE
003700      IF FUNCTIONFLAG EQUAL 5  PERFORM CLOSEDB.
003800  UPDATELIB-EXIT.
003900      EXIT PROGRAM.
004000*
004100*****************************************************************
004200*
004300 OPENUPDATE.
004400************
004500     OPEN TRUPDATE DBASE.
004600*
004700*
004800 OPENINQUIRY.
004900 ************
005000     OPEN INQUIRY DBASE.
005100*
005200*
005300 UPDATEDB.
005400*********
005500     IF TRFORMAT(TRIN) EQUAL TRFORMAT(CREATEACCT) PERFORM MAKEACCT
005600                       PERFORM MAKEACCT ELSE
005700     IF TRFORMAT(TRIN) EQUAL TRFORMAT(PURGEACCT)  PERFORM ELIMACCT
005800                       PERFORM ELIMACCT ELSE
005900     IF TRFORMAT(TRIN) EQUAL TRFORMAT(STATIS)
006000                       PERFORM STATUSCHK ELSE
006100     IF TRFORMAT(TRIN) EQUAL TRFORMAT(DEPOSIT)
006200                       PERFORM DEPAMNT ELSE
006300     IF TRFORMAT(TRIN) EQUAL TRFORMAT(WITHDRAWAL)
006400                       PERFORM WITHAMNT ELSE
006500     DISPLAY "NO UPDATE ROUTINE FOR THE FORMAT PASSED IN" .
006600*
006700*
006800 FORCEABORT.
```

Example 5–3. Update Library for Banking Transaction Program (cont.)

```
006900***********
007000     CLOSE DBASE.
007100*
007200*
007300 CLOSEDB.
007400********
007500 CLOSE DBASE.
007600*
007700*
007800* This is the update routine for creating a new account.
007900*
008000 MAKEACCT.
008100*********
008200     CREATE ACCOUNT ON EXCEPTION PERFORM ERR.
008300     MOVE ACCTNUM OF CREATEACCT OF TRIN TO ACCOUNT-NUM.
008400     MOVE NAME OF CREATEACCT OF TRIN TO NAME OF ACCOUNT.
008500     BEGIN-TRANSACTION(TRIN) RDS ON EXCEPTION PERFORM ERR.
008600     MID-TRANSACTION(TRIN,SAVEINPUT) RDS ON EXCEPTION PERFORM ERR.
008700     STORE ACCOUNT ON EXCEPTION PERFORM ERR.
008800*
008900     RETURN INPUT TRANSACTION AS GOOD TRANSACTION RESPONSE RECORD.
009000*
009100     MOVE TRIN TO TROUT.
009200     END-TRANSACTION(TRIN,SAVERESPONSE) RDS ON EXCEPTION PERFORM
009300     ERR.
009400*
009500* This is the update routine for eliminating an existing account.
009600*
009700 ELIMACCT.
009800*********
009900     MOVE ACCTNUM OF PURGEACCT OF TRIN TO ACCT.
010100     LOCK ACCOUNT-SET AT ACCOUNT-NUM = ACCT ON EXCEPTION PERFORM
010200     ERR.
010300     BEGIN-TRANSACTION(TRIN) RDS ON EXCEPTION PERFORM ERR.
010400     MID-TRANSACTION(TRIN,SAVEINPUT) RDS ON EXCEPTION PERFORM ERR.
010500     DELETE ACCOUNT ON EXCEPTION PERFORM ERR.
010600*
010700* Return same transaction as good transaction response record.
010800*
010900     MOVE TRIN TO TROUT.
011100     END-TRANSACTION(TRIN,SAVERESPONSE) RDS ON EXCEPTION PERFORM
011200     ERR.
011300*
011400* This is an example of an inquiry routine. It returns the balance
011500* of a particular account.
011600*
011700 STATUSCHK.
011800***********
011900     MOVE ACCTNUM OF STATIS OF TRIN TO ACCT.
012000     FIND ACCOUNT-SET AT ACCOUNT-NUM = ACCT ON EXCEPTION PERFORM
012100     ERR.
```

Example 5–3. Update Library for Banking Transaction Program (cont.)

```
012200     MOVE TRIN TO TROUT.
012300     MOVE BALANCE OF ACCOUNT TO BALANCE OF STATIS OF TROUT.
012400*
012500* This is the update routine to place deposits into accounts.
012600*
012700 DEPAMNT.
012800********
012900     MOVE ACCTNUM OF DEPOSIT OF TRIN TO ACCT.
013000     LOCK ACCOUNT-SET AT ACCOUNT-NUM = ACCT ON EXCEPTION PERFORM
013100     ERR.
013200     MOVE BALANCE OF ACCOUNT TO OLDBAL.
013300     ADD OLDBAL, AMOUNT OF DEPOSIT OF TRIN GIVING NEWBAL.
013400     CREATE DEPOSIT OF ACCOUNT ON EXCEPTION PERFORM ERR.
013500     MOVE TRANDATE OF DEPOSIT OF TRIN TO TRANDATE OF DEPOSIT OF
013600     ACCOUNT.
013700     MOVE AMOUNT OF DEPOSIT OF TRIN TO
013800          AMOUNT OF DEPOSIT OF ACCOUNT.
013900     MOVE OLDBAL TO OLD-BALANCE.
014000     MOVE NEWBAL TO NEW-BALANCE.
014100     MOVE NEWBAL TO BALANCE OF ACCOUNT.
014200     BEGIN-TRANSACTION(TRIN) RDS ON EXCEPTION PERFORM ERR.
014300     MID-TRANSACTION(TRIN,SAVEINPUT) RDS ON EXCEPTION PERFORM ERR.
014400     STORE ACCOUNT ON EXCEPTION PERFORM ERR.
014500     STORE DEPOSIT OF ACCOUNT ON EXCEPTION PERFORM ERR.
014600     CREATE STATIS IN TROUT.
014700     MOVE NEWBAL TO BALANCE OF STATIS OF TROUT.
014800    MOVE ACCTNUM OF DEPOSIT OF TRIN TO
014900           ACCTNUM OF STATIS OF TROUT.
015000    END-TRANSACTION(TRIN,SAVERESPONSE) RDS ON EXCEPTION PERFORM
015100    ERR.
015200*
015300* This is the update routine that performs withdrawals against
015400* accounts.
015500*
015600 WITHAMNT.
015700*********
015800    MOVE ACCTNUM OF WITHDRAWAL OF TRIN TO ACCT.
015900    LOCK ACCOUNT-SET AT ACCOUNT-NUM = ACCT ON EXCEPTION PERFORM
016000    ERR.
016100    MOVE BALANCE OF ACCOUNT TO OLDBAL.
016200    SUBTRACT AMOUNT OF WITHDRAWAL OF TRIN FROM OLDBAL GIVING
016300    NEWBAL.
016400    CREATE DEPOSIT OF ACCOUNT ON EXCEPTION PERFORM ERR.
016500    MOVE TRANDATE OF WITHDRAWAL OF TRIN TO TRANDATE OF DEPOSIT OF
016600    ACCOUNT.
016700    MOVE AMOUNT OF WITHDRAWAL OF TRIN TO AMOUNT OF DEPOSIT OF
016800    ACCOUNT.
016900    MOVE OLDBAL TO OLD-BALANCE.
017000    MOVE NEWBAL TO NEW-BALANCE.
017100    MOVE NEWBAL TO BALANCE OF ACCOUNT.
017200    BEGIN-TRANSACTION(TRIN) RDS ON EXCEPTION PERFORM ERR.
```

Example 5–3. Update Library for Banking Transaction Program (cont.)

```
017300    MID-TRANSACTION(TRIN,SAVEINPUT) RDS ON EXCEPTION PERFORM ERR.
017400    STORE ACCOUNT ON EXCEPTION PERFORM ERR.
017500    STORE DEPOSIT OF ACCOUNT ON EXCEPTION PERFORM ERR.
017600    CREATE STATIS IN TROUT.
017700    MOVE NEWBAL TO BALANCE OF STATIS OF TROUT.
017800    MOVE ACCTNUM OF DEPOSIT OF TRIN TO ACCTNUM OF STATIS OF TROUT.
017900    END-TRANSACTION(TRIN,SAVERESPONSE) RDS ON EXCEPTION PERFORM
018000    ERR.
018100*
018200* Report any exceptions that were detected.
018300*
018400 ERR.
018500*****
018600    DISPLAY "UPDATE RESULT:".
018700    DISPLAY "CATEGORY, ERRORTYPE, STRUCTURE " DMSTATUS(DMCATEGORY)
018800            ", " DMSTATUS(DMERRORTYPE) ", " DMSTATUS(DMSTRUCTURE).
018900    GO TO UPDATELIB-EXIT.
```

Example 5–3. Update Library for Banking Transaction Program (cont.)

COBOL74 Banking Transaction Program

Example 5–4 shows how TPS can be used for a number of simple banking transactions. The DASDL and TFL descriptions and the update library are provided in Examples 5–1 through 5–3.

```
000100$SET FREE
000200 IDENTIFICATION DIVISION.
000300 ENVIRONMENT DIVISION.
000400 INPUT-OUTPUT SECTION.
000500 FILE-CONTROL.
000600     SELECT REMOTEFILE
000700        ASSIGN TO REMOTE.
000800     SELECT PRINTFILE
000900        ASSIGN TO PRINTER.
001000 I-O-CONTROL.
001100 DATA DIVISION.
001200 FILE SECTION.
001300 FD  REMOTEFILE
001400     RECORD CONTAINS 72 CHARACTERS.
001500 01  RMT-REC                                  PIC X(72).
001600 01  RMT-FIELDS.
001700     03 RMT-1                                 PIC X(10).
001800     03 RMT-2                                 PIC X(10).
001900     03 RMT-3                                 PIC X(10).
002000     03 RMT-4                                 PIC X(10).
002100     03 RMT-5                                 PIC X(10).
002200     03 RMT-6                                 PIC X(10).
002300     03 RMT-7                                 PIC X(12).
002400 FD PRINTFILE
002500    RECORD CONTAINS 132 CHARACTERS.
```

Example 5–4. Banking Transaction Program

```
002600 01  PRINT-REC                                  PIC X(132).
002700*
002800*****************************************************************
002900*
003000* Declare the transaction base to be used, and assign an internal
003100* name to the status format.
003200*
003300*****************************************************************
003400*
003500 TRANSACTION SECTION.
003600 TB  TRB = BANKTR
003700        USING STATIS=STATUS, CREATEACCT, RESTARTDETANKER,
003800              DEPOSIT, WITHDRAWAL, PURGEACCT, SYSTEMTR.
003900 01  TRIN.
004000 01  TROUT.
004100 01  LASTINPUT.
004200 01  RESTARTTRREC.
004300 01  LASTRESPONSE.
004400*
004500 WORKING-STORAGE SECTION.
004600 77  USER-N PIC 999 VALUE 0.
004700 77  USER-ID   PIC X(7).
004800 01  JOBPARAMCOMP COMP WITH LOWER-BOUNDS.
004900     03  FILLER PIC 9(11) OCCURS 20.
005000 01  JOBPARAM REDEFINES JOBPARAMCOMP.
005100     03  JOBPARAMCHAR PIC X OCCURS 99.
005200 77  IDNUM        COMP      PIC 9(8).
005300 77  N             COMP      PIC 9(8).
005400 77  X             COMP      PIC 9(8).
005500 77  OPT           COMP      PIC 9(8).
005600 77  TANKING       COMP      PIC 9(8).
005700 77  ACCT                    PIC X(6).
005800 77  TPS-RESULT    COMP      PIC 9(8).
005900 77  TOTAL         COMP      PIC 9(6).
006000 77  TIMEOUTV      COMP      PIC 9(8).
006100 77  USER-NAME               PIC X(20).
006200 77  OPT-X                   PIC X(1).
006300 77  AMT                     PIC X(6).
006400 77  FILENUMBER    COMP      PIC 9(4).
006500 77  BLOCKNUMBER   COMP      PIC 9(8).
006600 77  OFFSETS       COMP      PIC 9(8).
006700 77  TEMP                    PIC X(4).
006800 77  ID-X                    PIC X(17).
006900 77  STATOPTION              PIC 9(4).
007000 01  SP.
007100     03  SPCHAR PIC X OCCURS 80.
007200 01  LIBPARAM PIC LX(100) DEPENDING ON N.
007300*
007400 *****************************************************************
007500*
007600 PROCEDURE DIVISION.
```

Example 5–4. Banking Transaction Program (cont.)

```
007700 MAIN SECTION.
007800 P1.
007900     OPEN I-O REMOTEFILE.
008000     OPEN OUTPUT PRINTFILE.
008100     MOVE 3 TO TANKING.
008200*
008300*****************************************************************
008400*
008500*  Set the LIBPARAMETER before the first call on the library
008600*  entry point.
008700*
008800*****************************************************************
008900*
009000     MOVE "JOURNAL NAME?" TO RMT-REC.
009100     PERFORM WRITE-RMT.
009200     READ REMOTEFILE INTO USER-NAME ; INVALID GO TO ERR.
009300     CHANGE ATTRIBUTE LIBPARAMETER OF "BANKTR/CODE/HOSTLIB"
009400        TO USER-NAME.
009500*
009600     MOVE "WHAT DO YOU WANT TO DO?" TO RMT-REC.
009700     PERFORM WRITE-RMT.
009800     MOVE  "CHOICE (1=UPDATE, 2=INQUIRY, 3=TANK, 4=READ)"
009900        TO RMT-REC.
010000     PERFORM WRITE-RMT.
010100     READ REMOTEFILE INTO OPT-X  ; INVALID KEY GO TO EXT.
010200     MOVE OPT-X TO OPT.
010300     MOVE OPT TO RMT-REC. PERFORM WRITE-RMT.
010400     PERFORM OPENTRBASE.
010500     IF TPS-RESULT > 0 GO TO ERR.
010600*
010700*****************************************************************
010800*
010900* Create a restart transaction to be written to the TRHISTORY
011000* file along with the input transaction. Note that values are
011100* not assigned to the items of this record. Normally the values
011200* are assigned; however, for readability the code is not shown
011300* in this example.
011400*
011500*****************************************************************
011600*
011700     CREATE RESTARTDETANKER IN RESTARTTRREC.
011800     MOVE "USER ID ? (X17)" TO RMT-REC.
011900     PERFORM WRITE-RMT.
012000     READ REMOTEFILE INTO ID-X  ; INVALID KEY GO TO EXT.
012100     PERFORM LOGONTRUSER.
012200     IF TPS-RESULT  > 0 GO TO ERR.
012300     MOVE IDNUM TO RMT-2.
012400     PERFORM WRITE-RMT.
012500*
012600 LOOP.
012700     MOVE "FUNCTION NAME (OR HELP)" TO RMT-REC.
```

Example 5–4. Banking Transaction Program (cont.)

```
012800      PERFORM WRITE-RMT.
012900      READ REMOTEFILE INTO SP  ; INVALID KEY GO TO EXT.
013000      IF SP = "CREATEUSER"   PERFORM  CREATEUSERP             ELSE
013100      IF SP = "PURGEUSER"    PERFORM  PURGEUSERP              ELSE
013200      IF SP = "CREATE"       PERFORM  CREATEP                 ELSE
013300      IF SP = "PURGE"        PERFORM  PURGEP                  ELSE
013400      IF SP = "DEPOSIT"      PERFORM  DEPOSITP                ELSE
013500      IF SP = "WITHDRAWAL"   PERFORM  WITHDRAWLP              ELSE
013600      IF SP = "QUIT"         PERFORM  QUITP                   ELSE
013700      IF SP = "STAT"         PERFORM  STATUSP                 ELSE
013800      IF SP = "NEWUSER"      PERFORM  NEWUSERP                ELSE
013900      IF SP = "REOPEN"       PERFORM  REOPENP                 ELSE
014000      IF SP = "SEEK"         PERFORM  SEEKP                   ELSE
014100      IF SP = "R"            PERFORM  READP                   ELSE
014200      IF SP = "HELP"         PERFORM  HELPP                   ELSE
014300      IF SP = "GETLAST"      PERFORM  GETLASTP THRU GETLASTP-EXIT
014400                                                              ELSE
014500      IF SP = "SW"           PERFORM  SWITCHP                 ELSE
014600      IF SP = "STAT"         PERFORM  STATISTICSP             ELSE
014700      DISPLAY "DID NOT RECOGNIZE FUNCTION NAME".
014800      GO TO LOOP.
014900 EXT.
015000      STOP RUN.
015100*****************************************************************
015200 CREATEP.
015300*********
015400      MOVE "FUNCTION IS CREATE" TO RMT-REC.
015500      PERFORM WRITE-RMT.
015600      PERFORM GETACCT.
015700      MOVE "CUSTOMER NAME? (X17) " TO RMT-REC.
015800      PERFORM WRITE-RMT.
015900      READ REMOTEFILE INTO USER-NAME  ; INVALID KEY GO TO EXT .
016000      CREATE CREATEACCT IN TRIN.
016100      MOVE ACCT TO ACCTNUM OF CREATEACCT OF TRIN.
016200      MOVE USER-NAME TO NAME OF CREATEACCT OF TRIN.
016300      PERFORM PROCESSTR THRU PROCESSTR-EXIT.
016400*
016500 PURGEP.
016600********
016700      MOVE "FUNCTION IS PURGE" TO RMT-REC.
016800      PERFORM WRITE-RMT.
016900      PERFORM GETACCT.
017000      CREATE PURGEACCT IN TRIN.
017100      MOVE ACCT TO ACCTNUM OF PURGEACCT OF TRIN.
017200      PERFORM PROCESSTR THRU PROCESSTR-EXIT.
017300*
017400 STATUSP.
017500*********
017600      MOVE "FUNCTION IS STATUS" TO RMT-REC.
017700      PERFORM WRITE-RMT.
017800      PERFORM GETACCT.
```

Example 5–4. Banking Transaction Program (cont.)

```
017900      CREATE STATIS IN TRIN.
018000      MOVE ACCT TO ACCTNUM OF STATIS OF TRIN.
018100      PERFORM PROCESSTR THRU PROCESSTR-EXIT.
018200*
018200* If there is no tanking, display the status of an account.
018300*
018400      IF OPT NOT EQUAL TANKING
018500      PERFORM DISPLAYSTATUS.
018600*
018700 DEPOSITP.
018800***********
018900      MOVE "FUNCTION IS DEPOSIT" TO RMT-REC.
019000      PERFORM WRITE-RMT.
019100      PERFORM GETACCT.
019200      MOVE "AMOUNT OF DEPOSIT? (NNNNNN) " TO RMT-REC.
019300      PERFORM WRITE-RMT.
019400      READ REMOTEFILE INTO AMT  ; INVALID KEY GO TO EXT .
019500      CREATE DEPOSIT IN TRIN.
019600      MOVE ACCT TO ACCTNUM OF DEPOSIT OF TRIN.
019700      MOVE TIME(1) TO TRANDATE OF DEPOSIT OF TRIN.
019800      MOVE AMT TO   AMOUNT OF DEPOSIT OF TRIN.
019900      PERFORM PROCESSTR THRU PROCESSTR-EXIT.
020000*
020100* If there is no tanking, display the status of an account.
020100*
020200      IF OPT NOT EQUAL TANKING
020300      PERFORM DISPLAYSTATUS.
020400*
020500 WITHDRAWLP.
020600************
020700      MOVE "FUNCTION IS WITHDRAWAL" TO RMT-REC.
020800      PERFORM WRITE-RMT.
020900      PERFORM GETACCT.
021000      MOVE "AMOUNT OF WITHDRAWAL? (NNNNNN) " TO RMT-REC.
021100      PERFORM WRITE-RMT.
021200      READ REMOTEFILE INTO AMT  ; INVALID KEY GO TO EXT .
021300      CREATE WITHDRAWAL IN TRIN.
021400      MOVE ACCT TO ACCTNUM OF WITHDRAWAL OF TRIN.
021500      MOVE  AMT TO AMOUNT  OF WITHDRAWAL OF TRIN.
021600      MOVE TIME(1)  TO TRANDATE OF WITHDRAWAL OF TRIN.
021700      PERFORM PROCESSTR THRU PROCESSTR-EXIT.
021800*
021900* If there is no tanking, display the status of an account.
022000*
022100      IF OPT NOT EQUAL TANKING
022200      PERFORM DISPLAYSTATUS.
022300*
022400 NEWUSERP.
022500************
022600      MOVE "FUNCTION IS NEW USER" TO RMT-REC.
022700      PERFORM WRITE-RMT.
```

Example 5–4. Banking Transaction Program (cont.)

```
022800      MOVE "USER ID ? (X17)  " TO RMT-REC.
022900      PERFORM WRITE-RMT.
023000      READ REMOTEFILE INTO ID-X  ; INVALID KEY GO TO EXT.
023100      PERFORM LOGONTRUSER.
023200      IF TPS-RESULT > 0 GO TO ERR.
023300      MOVE "USER# =" TO RMT-1.
023400      MOVE IDNUM TO RMT-2.
023500      PERFORM WRITE-RMT.
023600*
023700 REOPENP.
023800*********
023900      MOVE "FUNCTION IS REOPEN" TO RMT-REC.
024000      PERFORM WRITE-RMT.
024100      PERFORM CLOSETRBASE.
024200      IF TPS-RESULT > 0 GO TO ERR.
024300      MOVE "WHAT DO YOU WANT TO DO?" TO RMT-REC.
024400      PERFORM WRITE-RMT.
024500      MOVE "CHOICE (1=UPDATE,2=INQ,3=TANK,4=READ)" TO RMT-REC.
024600      PERFORM WRITE-RMT.
024700      READ REMOTEFILE INTO OPT-X  ; INVALID KEY GO TO EXT .
024800      MOVE OPT-X TO OPT.
024900      PERFORM  OPENTRBASE.
025000      IF TPS-RESULT > 0 GO TO ERR.
025100*
025200 SEEKP.
025300********
025400      MOVE "FUNCTION IS SEEK" TO RMT-REC.
025500      PERFORM WRITE-RMT.
025600      MOVE "ENTER FILENUM (NNNN) " TO RMT-REC.
025700      PERFORM WRITE-RMT.
025800      READ REMOTEFILE INTO TEMP        ; INVALID KEY GO TO EXT .
025900      MOVE TEMP TO FILENUMBER.
026000      MOVE "ENTER BLOCK (NNNN) " TO RMT-REC.
026100      PERFORM WRITE-RMT.
026200      READ REMOTEFILE INTO TEMP        ; INVALID KEY GO TO EXT .
026300      MOVE TEMP TO BLOCKNUMBER.
026400      MOVE "ENTER OFFSET (NNNN) " TO RMT-REC.
026500      PERFORM WRITE-RMT.
026600      READ REMOTEFILE INTO TEMP     ; INVALID KEY GO TO EXT .
026700      MOVE TEMP TO OFFSETS.
026800      PERFORM SEEKTRANSACTION.
026900      IF TPS-RESULT > 0 GO TO ERR.
027000
027100 READP.
027200********
027300      MOVE "FUNCTION IS READ" TO RMT-REC.
027400      PERFORM WRITE-RMT.
027500      PERFORM READTRANSACTION.
027600      IF TPS-RESULT > 0 GO TO ERR.
027700      MOVE "F/B/O =" TO RMT-1.
027800      MOVE TRFILENUM(TRIN) TO RMT-2.
```

Example 5–4. Banking Transaction Program (cont.)

```
027900      MOVE TRBLOCKNUM(TRIN)  TO RMT-3.
028000      MOVE TROFFSET(TRIN)  TO RMT-4.
028100      MOVE TRFORMAT(TRIN)  TO RMT-5.
028200      MOVE TRSUBFORMAT(TRIN)  TO RMT-6.
028300      PERFORM WRITE-RMT.
028400*
028500 CREATEUSERP.
028600**************
028700      MOVE "FUNCTION IS CREATEUSER" TO RMT-REC.
028800      PERFORM WRITE-RMT.
028900      MOVE "USER ID ? (X17) " TO RMT-REC.
029000      PERFORM WRITE-RMT.
029100      READ REMOTEFILE INTO ID-X  ; INVALID KEY GO TO EXT.
029200      PERFORM CREATETRUSER.
029300      IF TPS-RESULT > Ø GO TO ERR.
029400      PERFORM LOGONTRUSER.
029500      IF TPS-RESULT > Ø GO TO ERR.
029600      MOVE "USER# =" TO RMT-1.
029700      MOVE IDNUM TO RMT-2.
029800      PERFORM WRITE-RMT.
029900*
030000 PURGEUSERP.
030100*************
030200      MOVE "FUNCTION IS PURGE USER" TO RMT-REC.
030300      PERFORM WRITE-RMT.
030400      PERFORM PURGETRUSER.
030500      IF TPS-RESULT > Ø   GO TO ERR.
030600*
030700 QUITP.
030800********
030900      MOVE "FUNCTION IS QUIT" TO RMT-REC.
031000      PERFORM WRITE-RMT.
031100      PERFORM CLOSETRBASE.
031200      GO TO EXT.
031300*
031400 SWITCHP.
031500*********
031600      MOVE "FUNCTION IS SWITCH" TO RMT-REC.
031700      PERFORM WRITE-RMT.
031800      PERFORM SWITCHTRFILE.
031900      IF TPS-RESULT > Ø GO TO ERR.
032000*
032100 STATISTICSP.
032200**************
032300      MOVE "FUNCTION IS STATISTICS" TO RMT-REC.
032400      PERFORM WRITE-RMT.
032500      MOVE "WHAT DO YOU WANT TO DO?" TO RMT-REC.
032600      PERFORM WRITE-RMT.
032700      MOVE "CHOICE(1=PRINT & RESET, 2=PRINT, 3=RESET)"
032800      TO RMT-REC.
032900      PERFORM WRITE-RMT.
```

Example 5–4. Banking Transaction Program (cont.)

```
033000     READ REMOTEFILE INTO TEMP; INVALID KEY GO TO
033100     EXT.
033200     MOVE TEMP TO STATOPTION.
033300     PERFORM HANDLESTATISTICS.
033400     IF TPS RESULT > 0 GO TO ERR.
033500*
033600 HELPP.
033700********
033800     MOVE "FUNCTIONS : TEST,CREATE,PURGE,DEPOSIT" TO RMT-REC.
033900     PERFORM WRITE-RMT.
034000     MOVE "            WITHDRAWAL,QUIT,STATUS,NEWUSER" TO RMT-REC.
034100     PERFORM WRITE-RMT.
034200     MOVE "            REOPEN,SEEK,READ,GETLAST" TO RMT-REC.
034300     PERFORM WRITE-RMT.
034400     MOVE "            CREATEUSER,PURGEUSER,SWITCH" TO RMT-REC.
034500     PERFORM WRITE-RMT.
034600     MOVE "STATISTICS,HELP."                         TO RMT-REC.
034700     PERFORM WRITE-RMT.
034800*
034900 ERR.
035000*****
035100     MOVE "RESULT =" TO RMT-1.
035200     MOVE TPS-RESULT TO RMT-2.
035300     PERFORM WRITE-RMT.
035400     GO TO LOOP.
035500*
035600 WRITE-RMT.
035700***********
035800     WRITE RMT-REC.
035900     WRITE PRINT-REC FROM RMT-REC.
036000     MOVE SPACES TO RMT-REC.
036100*
036200 PROCESSTR.
036300***********
036400     IF OPT NOT EQUAL TANKING GO TO PROCESSTRANS.
036500     PERFORM TANKTRNORESTART.
036600     IF TPS-RESULT > 0 GO TO ERR.
036700     GO TO PROCESSTR-EXIT.
036800 PROCESSTRANS.
036900     PERFORM PROCESSTRANSACTION.
037000     IF TPS-RESULT > 0 GO TO ERR.
037100 PROCESSTR-EXIT.
037200*
037300 GETACCT.
037400*********
037500     MOVE "ACCOUNT NUMBER ? (NNNNNN) " TO RMT-REC.
037600     PERFORM WRITE-RMT.
037700     READ REMOTEFILE INTO ACCT  ; INVALID KEY GO TO EXT.
037800*
037900 GETLASTP.
038000***********
```

Example 5–4. Banking Transaction Program (cont.)

```
038100      IF OPT NOT EQUAL TANKING   GO TO GETLASTRESTRTREC.
038200      PERFORM RETURNRESTARTINFO.
038300      IF TPS-RESULT > 0 GO TO ERR.
038400      MOVE "LAST RESTART (FILE,BLK,OFFSET,FMT):" TO RMT-REC.
038500      PERFORM WRITE-RMT.
038600      MOVE TRFILENUM(TROUT) TO RMT-2.
038700      MOVE TRBLOCKNUM(TROUT)  TO RMT-3.
038800      MOVE TROFFSET(TROUT)  TO RMT-9.
038900      MOVE TRFORMAT(TROUT)  TO RMT-5.
039000      PERFORM WRITE-RMT.
039100      GO TO GETLASTP-EXIT.
039200      GETLASTRESTRTREC.
039300      PERFORM RETURNRESTARTINFO.
039400      IF TPS-RESULT > 0 GO TO ERR.
039500      MOVE "LAST RESTART (FILE,BLK,OFFSET,FMT):" TO RMT-REC.
039600      PERFORM WRITE-RMT.
039700      MOVE TRFILENUM(TROUT) TO RMT-2.
039800      MOVE TRBLOCKNUM(TROUT)  TO RMT-3.
039900      MOVE TROFFSET(TROUT)  TO RMT-4.
040000      MOVE TRFORMAT(TROUT)  TO RMT-5.
040100      PERFORM WRITE-RMT.
040200*
040300      PERFORM RETURNLASTRESPONSE.
040400      MOVE "LAST RESPONSE(FILE,BLK,OFFSET,FMT)" TO RMT-REC.
040500      PERFORM WRITE-RMT.
040600      MOVE TRFILENUM(TROUT) TO RMT-2.
040700      MOVE TRBLOCKNUM(TROUT)  TO RMT-3.
040800      MOVE TROFFSET(TROUT)  TO RMT-4.
040900      MOVE TRFORMAT(TROUT)  TO RMT-5.
041000      PERFORM WRITE-RMT.
041100 GETLASTP-EXIT.
041200*
041300 DISPLAYSTATUS.
041400**********
041500      IF TRFORMAT(TROUT) NOT EQUAL TRFORMAT(STATIS) GO TO ERR.
041600      MOVE "ACCOUNT # " TO RMT-1.
041700      MOVE ACCTNUM OF STATIS OF TROUT TO RMT-2.
041800      MOVE "CUR BAL =" TO RMT-3.
041900      MOVE BALANCE OF STATIS OF TROUT TO TOTAL.
042000      MOVE TOTAL TO RMT-4.
042100      PERFORM WRITE-RMT.
042200*
042300*************************************************************************
042400*
042500* The following are the library entry point calls.
042600*
042700*************************************************************************
042800*
042900 CREATETRUSER.
043000      CALL "CREATETRUSER        OF BANKTR/CODE/HOSTLIB"
043100         USING ID-X , IDNUM
```

Example 5–4. Banking Transaction Program (cont.)

```
043200        GIVING TPS-RESULT.
043300*
043400 PURGETRUSER.
042500     CALL "PURGETRUSER         OF BANKTR/CODE/HOSTLIB"
042600        USING IDNUM
042700        GIVING TPS-RESULT.
042800*
042900 LOGONTRUSER.
043000     CALL "LOGONTRUSER         OF BANKTR/CODE/HOSTLIB"
043100        USING ID-X, IDNUM
043200        GIVING TPS-RESULT.
043300*
043400 LOGOFFTRUSER.
043500     CALL "LOGOFFTRUSER         OF BANKTR/CODE/HOSTLIB"
043600        USING IDNUM
043700        GIVING TPS-RESULT.
043800*
043900 RETURNRESTARTINFO.
044000     CALL "RETURNRESTARTINFO    OF BANKTR/CODE/HOSTLIB"
044100        USING IDNUM, TROUT
044200        GIVING TPS-RESULT.
044300*
044400 RETURNLASTRESPONSE.
044500     CALL "RETURNLASTRESPONSE    OF BANKTR/CODE/HOSTLIB"
044600        USING IDNUM, TROUT
044700        GIVING TPS-RESULT.
044800*
044900 TANKTRNORESTART.
045000     CALL "TANKTRNORESTART      OF BANKTR/CODE/HOSTLIB"
045100        USING IDNUM, TRIN
045200        GIVING TPS-RESULT.
045300*
045400 PROCESSTRANSACTION.
045500     CALL "PROCESSTRANSACTION      OF BANKTR/CODE/HOSTLIB"
045600        USING IDNUM, TRIN, TROUT, RESTARTTRREC
045700        GIVING TPS-RESULT.
045800*
045900 OPENTRBASE.
046000     CALL "OPENTRBASE              OF BANKTR/CODE/HOSTLIB"
046100        USING OPT, TIMEOUTV
046200        GIVING TPS-RESULT.
046300*
046400 CLOSETRBASE.
046500     CALL "CLOSETRBASE             OF BANKTR/CODE/HOSTLIB"
046600        GIVING TPS-RESULT.
046700*
048000 SEEKTRANSACTION.
048100     CALL "SEEKTRANSACTION         OF BANKTR/CODE/HOSTLIB"
048200        USING FILENUMBER, BLOCKNUMBER, OFFSETS
048300        GIVING TPS-RESULT.
048400*
```

Example 5–4. Banking Transaction Program (cont.)

```
048500 READTRANSACTION.
048600     CALL "READTRANSACTION          OF BANKTR/CODE/HOSTLIB"
048700        USING TRIN
048800        GIVING TPS-RESULT.
048900*
049000 SWITCHTRFILE.
049100     CALL "SWITCHTRFILE             OF BANKTR/CODE/HOSTLIB"
049200        GIVING TPS-RESULT.
049300*
049400 HANDLESTATISTICS.
049500     CALL "HANDLESTATISTICS         OF BANKTR/CODE/HOSTLIB"
049600        USING STATOPTION
049700        GIVING TPS-RESULT.
```

Example 5–4. Banking Transaction Program (cont.)

Transaction Base and Entry Points for a COBOL74 Programming Example

Example 5–5 shows how the transaction base and library entry points are declared. For information about calling the entry points in COBOL74, see "Using Transaction Library Entry Points" earlier in this section. For detailed information on the library entry points, refer to the *DMSII TPS Programming Guide*.

```
000100 IDENTIFICATION DIVISION.
000200 ENVIRONMENT DIVISION.
000300 .
000400 .
000500 .
000600 DATA DIVISION.
000700 FILE SECTION.
000800 .
000900 .
001000 .
001100 DATA-BASE SECTION.
001200 TRANSACTION SECTION.
001300*
001400* Specify the transaction record format names to be used.
001500*
001600 TB  TRB = BANKTR
001700     USING STATIS=STATUS, CREATEACCT, PURGEACCT, DEPOSIT,
001800          WITHDRAWAL, CHANGEACCT, STATEMENT, TEST,
001900          RESTARTDETANKER
002000 .
002100*
002200* Specify the transaction record variables.
002300*
002400 01  TRIN.
002500 01  TROUT.
002600 01  LASTINPUT.
```

Example 5–5. Declaring the Transaction Base and Entry Points

```
002700 01  LASTRESPONSE.
002800 WORKING-STORAGE SECTION.
002900 .
003000 .
003100*
003200*************************************************************************
003300 PROCEDURE DIVISION.
003400 MAIN SECTION.
003500 .
003600 .
003700 .
003800 STOP RUN.
003900*************************************************************************
004000*
004100 CREATETRUSER.
004200     CALL "CREATETRUSER        OF BANKTR/CODE/HOSTLIB"
004300        USING ID-X , IDNUM
004400        GIVING TPS-RESULT.
004500*
004600 PURGETRUSER.
004700     CALL "PURGETRUSER         OF BANKTR/CODE/HOSTLIB"
004800        USING ID-X
004900        GIVING TPS-RESULT.
005000*
005100 LOGONTRUSER.
005200     CALL "LOGONTRUSER         OF BANKTR/CODE/HOSTLIB"
005300        USING ID-X, IDNUM
005400        GIVING TPS-RESULT.
005500*
005600 LOGOFFTRUSER.
005700     CALL "LOGOFFTRUSER        OF BANKTR/CODE/HOSTLIB"
005800        USING IDNUM
005900        GIVING TPS-RESULT.
006000*
006100 RETURNRESTARTINFO.
006200     CALL "RETURNRESTARTINFO   OF BANKTR/CODE/HOSTLIB"
006300        USING IDNUM, TROUT
006400        GIVING TPS-RESULT.
006500*
006600 RETURNLASTRESPONSE.
006700     CALL "RETURNLASTRESPONSE   OF BANKTR/CODE/HOSTLIB"
006800        USING IDNUM, TROUT
006900        GIVING TPS-RESULT.
007000*
007100 TANKTRNORESTART.
007200     CALL "TANKTRNORESTART      OF BANKTR/CODE/HOSTLIB"
007300        USING IDNUM, TRIN
007400        GIVING TPS-RESULT.
007500*
007600 PROCESSTRNORESTART.
007700     CALL "PROCESSTRNORESTART   OF BANKTR/CODE/HOSTLIB"
```

Example 5–5. Declaring the Transaction Base and Entry Points (cont.)

```
007800         USING IDNUM, TRIN, TROUT
007900         GIVING TPS-RESULT.
008000*
008100 OPENTRBASE.
008200     CALL "OPENTRBASE           OF BANKTR/CODE/HOSTLIB"
008300         USING OPT, TIMEOUTV
008400         GIVING TPS-RESULT.
008500*
008600 CLOSETRBASE.
008700     CALL "CLOSETRBASE          OF BANKTR/CODE/HOSTLIB"
008800         GIVING TPS-RESULT.
008900*
009000 SEEKTRANSACTION.
009100     CALL "SEEKTRANSACTION      OF BANKTR/CODE/HOSTLIB"
009200         USING FILENUMBER, BLOCKNUMBER, OFFSETS
009300         GIVING TPS-RESULT.
009400*
009500 READTRANSACTION.
009600     CALL "READTRANSACTION      OF BANKTR/CODE/HOSTLIB"
009700         USING TRIN
009800         GIVING TPS-RESULT.
009900*
010000 SWITCHTRFILE.
010100     CALL "SWITCHTRFILE         OF BANKTR/CODE/HOSTLIB"
010200         GIVING TPS-RESULT.
010300*
010400 HANDLESTATISTICS.
010500     CALL "HANDLESTATISTICS     OF BANKTR/CODE/HOSTLIB"
010600         USING STATOPTION
010700         GIVING TPS-RESULT.
```

Example 5–5. Declaring the Transaction Base and Entry Points (cont.)

Section 6
Using the SDF Program Interface

Screen Design Facility (SDF) is a tool to help programmers design and process forms for applications. SDF provides form processing that eliminates the need for complicated format language or code, and it enables you to provide validation for data entered on forms by application users.

The program interface developed for SDF includes the following:

- Extensions that enable you to read and write forms easily
- Invocation of form data into your program as COBOL74 declarations
- Use of message keys for form processing
- Programmatic control over data manipulation and display on a form image

This section provides information about the extensions developed for SDF and explains the syntax for using message keys and programmatic controls in an application. Each extension is presented with its syntax and an example. Sample programs are included at the end of the section.

For an alphabetized list of the extensions, see "SDF Extensions" in Section 1, "Introduction to COBOL74 Program Interfaces." Information on general programming considerations and concepts is provided in the *A Series Screen Design Facility (SDF) Operations and Programming Guide*.

You can use SDF with the Advanced Data Dictionary System (ADDS) and with the Communications Management System (COMS).

When you use SDF with ADDS, you can take advantage of the following ADDS capabilities:

- Defining prefixes for entities, including DMSII database elements, COBOL74 data description items, or fields of SDF or SDF Plus forms
- Defining a synonym, which means referring to an entity by another name

For further information about defining prefixes and synonyms for entities, refer to the *InfoExec ADDS Operations Guide*.

Refer to the product documentation for information on the concepts and programming considerations for using ADDS and COMS with SDF. For more information on the extensions used with these products, refer to Section 2, "Using the ADDS Program Interface" and Section 3, "Using the COMS Program Interface."

Identifying the Dictionary

The dictionary is identified in the SPECIAL-NAMES paragraph of the program, using the DICTIONARY statement.

Note: *A program can invoke only one dictionary. Therefore, if a program accesses both a SIM database (from a dictionary) and SDF forms, both must be in the same dictionary.*

General Format

The general format of the DICTIONARY statement is as follows:

```
[, DICTIONARY IS literal-1]
```

Explanation of Format Elements

DICTIONARY

This clause enables you to identify the function name of the dictionary library.

literal-1

This literal is the function name that is equated to a library code file using the SL (System Library) system command. See the *SDF Operations and Programming Guide* for instructions on equating these names.

Example

The following example uses the DICTIONARY statement to identify the SCREENDESIGN function name:

```
001000 IDENTIFICATION DIVISION.
007000
008000 ENVIRONMENT DIVISION.
009000 CONFIGURATION SECTION.
011100
011200 SPECIAL-NAMES.
011300   DICTIONARY IS "SCREENDESIGN".
```

Invoking Data Descriptions

A data-description entry specifies the characteristics of a particular data item. You can use the FROM DICTIONARY clause in the DATA DIVISION to obtain a formlibrary from the dictionary. There are two optional clauses: SAME RECORD AREA and REDEFINES.

The SAME RECORD AREA clause, when used in the DATA DIVISION, applies only to SDF forms. The clause invokes all form descriptions in the formlibrary as redefinitions of the first form in the formlibrary. Only one record area is allocated for the formlibrary. If the SAME RECORD AREA clause is not used, each form in the formlibrary is invoked as a separate form description with its own record area.

The SAME RECORD AREA clause is used with general files in the ENVIRONMENT DIVISION; refer to Volume 1 for more information on its use in that division.

The REDEFINES clause enables the same storage area to be described by different data descriptions. In the data-description entry for SDF, the REDEFINES clause facilitates redefinition of a formlibrary that has previously been invoked with the SAME RECORD AREA clause. The REDEFINES clause enables multiple formlibraries to have the same record area. In addition, multiple redefinitions of the same formlibrary are allowed.

Only information specific to the REDEFINES clause used in the data-description entry for SDF is covered in this section. For the general syntax and rules of the REDEFINES clause, refer to Volume 1.

General Format

The general format of the data-description entry is as follows:

```
level-number-1 formlibrary-name-1
FROM DICTIONARY
[ ; VERSION IS literal-1] [ ; DIRECTORY IS literal-2]
[; SAME RECORD AREA]
[; REDEFINES formlibrary-name-2] .
```

Explanation of Format Elements

level-number-1

This entry must be at the 01-level.

formlibrary-name-1

This format element identifies a dictionary structure that is a collection of record descriptions defining screen formats. An alias cannot be specified for formlibrary-name-1.

The clauses can be written in any order. However, formlibrary-name-1 must immediately follow level-number-1 and FROM DICTIONARY must immediately follow form-library-name-1.

For information on dictionaries, refer to the *InfoExec ADDS Operations Guide*.

FROM DICTIONARY

This format element obtains a formlibrary from the dictionary.

VERSION IS literal-1

This format element is used only with ADDS and specifies a numeric literal that identifies a version of the file. Literal-1 comprises from 1 to 6 digits.

DIRECTORY IS literal-2

This format element is used only with ADDS and specifies a nonnumeric literal that identifies the directory under which the file is stored in the data dictionary.

SAME RECORD AREA and REDEFINES

The SAME RECORD AREA and REDEFINES options can be used only with formlibrary invocations in the WORKING-STORAGE, LINKAGE, and LOCAL-STORAGE sections. All other uses of the clauses cause a syntax error.

The formlibrary invocation using the REDEFINES option must immediately follow either the formlibrary-name-1 invocation that it is redefining, or another formlibrary invocation that is using a REDEFINES option of formlibrary-name-1. The REDEFINES option can be used to redefine only formlibraries that previously have been invoked using the SAME RECORD AREA option.

formlibrary-name-2

This format element identifies a dictionary structure that is a collection of record descriptions defining screen formats. Form-library-name-2 is used in the REDEFINES option.

Considerations for Use

Only a formlibrary can be invoked directly from the dictionary; a form cannot be invoked directly.

Formlibraries can be invoked within the FILE SECTION, the WORKING-STORAGE SECTION, the LINKAGE SECTION, or the LOCAL-STORAGE SECTION.

A formlibrary cannot be given an alias using the INVOKE clause.

Example

The following example shows code for a data-description entry:

```
Ø16ØØØ DATA DIVISION.
Ø17ØØØ FILE SECTION.
Ø18ØØØ FD REMFILE.
Ø212ØØ Ø1 SAMPLELIB  FROM DICTIONARY.
```

Reading Forms

The READ FORM statement is used to read a form from a station to a program. The options enable you to read forms with message keys and to use default forms.

When the logical records of a file are described with more than one record description, these records automatically share the same storage area. This sharing is equivalent to an implicit redefinition of the area. The contents of any data items that are beyond the range of the current data record depend on previous events.

The appropriate formlibrary reads data from the station for the form, validates the input for the record, performs input error handling if necessary, and passes the valid record or detected error condition to the program.

One of the error conditions that can be passed to the program is a COMS error message. For information on how to process COMS errors, refer to the *COMS Programming Guide*.

General Format

The general format of the READ FORM statement is as follows:

```
READ FORM file-name-1 USING {formlibrary-name-1
                            {form-name-1[FROM DEFAULT FORM]}
[INTO identifier-1]
[; ON ERROR imperative-statement-1]
```

Explanation of Format Elements

file-name-1

This format element identifies the file that must be open in the input or I/O mode when this statement is executed. When the READ FORM statement is executed, the value of the file status data item associated with file-name-1 is updated. The file status data item must be defined in the DATA DIVISION as a 2-character, alphanumeric data item. For further information on status reporting, see Volume 1.

If the read operation is successful, the data is made available in the record storage area associated with file-name-1. For general information on opening files, see Volume 1.

The storage area associated with identifier-1 and the record area associated with file-name-1 must not be the same area.

When form-name-1 is not invoked in the file description for file-name-1, the data is read into the file-name-1 record area and transferred to form-name-1. To avoid truncation of trailing characters in the message, make sure the record description for file-name-1 is as large as the declaration of the largest form to be used with the file.

formlibrary-name-1

This format element is used in the USING clause to read self-identifying forms. Self-identifying forms are forms that contain a Message Key field in the same location in all forms in the formlibrary. The Message Key field is used to determine programmatically which form record format has been read.

The READ FORM statement used with self-identifying forms requires the formlibrary name rather than the form name, because the form is identified by its message key.

If you specify the message key option for the formlibrary, you must define message keys for each form in the formlibrary. However, you can read a form using the READ FORM statement for either a specific or a self-identifying form.

Refer to the appropriate sample program later in this section for an illustration of using message keys. For an explanation of message keys and ways to specify them in SDF, refer to the *SDF Operations and Programming Guide*.

form-name-1

This format element is used in the USING clause to read specific forms that are identified by form-name-1.

FROM DEFAULT FORM

This option can be used only with a form name. When the FROM DEFAULT FORM option is specified, the formlibrary writes a form with default values in each field before performing the read and record validation.

INTO identifier-1

If the INTO option is specified, the record being read is moved from the record area to the area specified by the identifier, according to the rules specified for the MOVE statement. The sending area is considered to be a group item equal in size to the maximum record size for this file.

Identifier-1 is a record item declared in the WORKING-STORAGE SECTION that is used to store the information received as a result of the READ FORM statement.

The INTO option can be used only if form-name-1 or formlibrary-name-1 is invoked in the file description associated with file-name-1.

The implied move operation does not occur if the READ FORM statement cannot be executed. Any subscripting or indexing associated with the identifier is evaluated after the record is read and immediately before the record is moved to the data item.

When the INTO option is used, the record being read is available in both the input record area and the data area associated with the identifier. For more information on the MOVE statement, refer to Volume 1.

For general information on error-handling with remote files, see the *SDF Operations and Programming Guide.*

ON ERROR imperative-statement-1

Execution of the ON ERROR option is the same as for an Invalid Key condition. When the ON ERROR condition is recognized, imperative-statement-1 is performed. For information on the Invalid Key condition, see Volume 1.

Considerations for Use

Two run-time errors specific to forms can be handled programmatically by using the FILE STATUS clause in the INPUT-OUTPUT SECTION. For information on the FILE STATUS clause, see Volume 1. A form error message (82) is returned when either of the following occurs:

- A read is executed on a specific form and that form no longer resides in the formlibrary.
- The compile-time form version does not equal the run-time form version.

Examples

The following examples illustrate the use of the READ FORM statement.

Example of a Read Operation for a Self-Identifying Form When Message Keys Are Used

```
READ FORM REMFILE USING SAMPLELIB.
```

Example of Reading a Specific Form

```
READ FORM REMFILE USING SAMPLEFORM1.
```

Example of a Read Operation Using a Specific Default Form

```
READ FORM REMFILE USING SAMPLEFORM1 FROM DEFAULT FORM.
```

Example of the INTO and ON ERROR Options

```
.
.
WORKING-STORAGE SECTION.
   Ø1 TEMPSAMPLEFORM  PIC X(2ØØØ).
.
.
PROCEDURE DIVISION.
.
READ FORM REMFILE USING SAMPLEFORM1
      INTO TEMPSAMPLEFORM
      ON ERROR STOP RUN.
```

Writing Forms

The WRITE FORM statement is used to write forms from a program to a station. The DEFAULT FORM option enables you to write a form with default values in each field.

The execution of the WRITE FORM statement releases a logical record to a file. The file must be open in the output, I/O, or extend mode when the statement is executed. For general information on opening files, see Volume 1.

Execution of a WRITE FORM statement does not affect the contents or accessibility of the record area. If a file is named in a SAME RECORD AREA clause, the logical record is also available as a record of other files referenced in that SAME RECORD AREA clause.

The current record pointer also is unaffected by the execution of a WRITE FORM statement.

General Format

The general format of the WRITE FORM statement is as follows:

```
WRITE FORM { record-name USING form-name-2 } [ FROM { identifier-1      } ]
           { form-name-1                     }       { DEFAULT FORM     }
[; ON ERROR imperative-statement-1].
```

Explanation of Format Elements

record-name

This format element is the name of a logical record in the FILE SECTION of the DATA DIVISION and can be qualified. Record-name and identifier-1 must not reference the same storage area.

form-name-1 and form-name-2

These form names must be uniquely named forms in the formlibrary identified in the file description (FD) for this file. A formlibrary is a run-time library containing one or more form-processing procedures.

USING

This clause enables you to write the forms of a formlibrary previously invoked in the WORKING-STORAGE SECTION. The normal record area (record-name) of the file is ignored and the record area for form-name-2 is written.

FROM identifier-1

A WRITE FORM statement with the FROM option is equivalent to the statement *MOVE identifier-1 TO record-name* (according to MOVE statement rules) followed by the WRITE FORM statement without the FROM phrase. For more information on the MOVE statement, refer to Volume 1.

FROM DEFAULT FORM

The DEFAULT FORM option causes the formlibrary to write a form with default values in each field.

ON ERROR imperative-statement-1

Execution of the ON ERROR option is the same as for an Invalid Key condition. When the ON ERROR condition is recognized, imperative-statement-1 is performed. For information on the Invalid Key condition, see Volume 1.

The appropriate formlibrary performs the write. Any detected error condition is returned to the program. For general information on error-handling when using remote files, refer to the *SDF Operations and Programming Guide*.

Considerations for Use

Two run-time errors specific to forms can be handled programmatically by using the FILE STATUS clause in the INPUT-OUTPUT SECTION. For information on the FILE STATUS clause, see Volume 1.

A form error message (82) is returned when either of the following occurs:

- A write operation is executed on a specific form and that form no longer resides in the formlibrary.
- The compile-time form version does not equal the run-time form version.

Example

This example shows the WRITE FORM statement used with the ON ERROR option.

```
WRITE FORM SAMPLEFORM1
    ON ERROR STOP RUN.
```

Using the FORM-KEY Function

The FORM-KEY function enables the compiler to pass an internal binary number uniquely identifying the form name for programmatic purposes. This function is required for using the data communications interface (DCI) library with the Communications Management System (COMS) interface.

When using SDF with COMS, this function must be used to move the form key into the first word of the conversation area of output before a SEND statement can be executed.

General Format

The general format of the FORM-KEY function is as follows:

```
FORM-KEY (form-name-1)
```

Explanation of Format Element

form-name-1

This format element is an internal binary number uniquely identifying the form name.

See Also

- Refer to "SDF Sample Programs" later in this section for examples of using the FORM-KEY function in an application program.
- Information about using SDF extensions with COMS is provided later in this section in "Using SDF with COMS."
- Refer to Section 3, "Using the COMS Program Interface," for information about the extensions developed for COMS.

Examples

The following is an example of the function syntax within the MOVE statement. For more information on numeric functions or the MOVE statement, refer to Volume 1.

```
MOVE  FORM-KEY(FORMA)  TO SAVEKEY.
```

The following example shows the FORM-KEY function used with the COMS interface to move the form key SDFFORM to the first word of the conversation area:

```
MOVE FORM-KEY(SDFFORM) TO  COMS-OUT-CONVERSATION.
```

Manipulating Programmatic Controls

SDF provides programmatic controls at the form level and at the field level. The controls are fields identified by a scheme of names and suffixes. The naming conventions are described under "Programmatic Flag Setting" later in this section. The following paragraphs explain how these controls are manipulated in a COBOL74 application program.

The symbolic name for SDF default programmatic flags takes the form *<entity name><-suffix>*. If you do not use the default suffix names that SDF provides, the suffixes you specify must be unique for each flag type. The symbolic name, including its suffix, can contain a maximum of 30 characters.

Example

In the following example of manipulating data using a programmatic control, assume that a field is defined as EMPLOYEE. Assume also that you define the Cursor programmatic control for the EMPLOYEE field and use the default Cursor flag suffix (-CURSOR). These definitions generate a data area in your program, as follows:

```
Ø2 EMPLOYEE-CURSOR
```

In the program, you can assign a value for the flag setting, as shown in the following example:

```
Ø1 CONSTANTS.
  Ø2 TRUE-FLAG     PIC 9 COMP VALUE 1.
  Ø2 FALSE-FLAG    PIC 9 COMP VALUE Ø.
```

In the program, you can use the following syntax to cause the cursor to be placed in the EMPLOYEE field. SDFFORM is the name of the form to be written.

```
MOVE TRUE-FLAG TO EMPLOYEE-CURSOR.
WRITE SDFFORM.
```

See Also

- For a detailed discussion of programmatic controls, refer to the *SDF Operations and Programming Guide*.
- Refer to the sample programs in "SDF Sample Programs" at the end of this section for another example of using programmatic controls in an application.

Programmatic Flag Setting

Programmatic flags are set either by the formlibrary (with a read operation) or by a program (with a write operation), depending on the type of flag. Programmatic controls cause extra data items to be generated into the COBOL74 program record description.

SDF does not reset any form control flags. If you set a form control flag in a program, you must also reset that flag when you no longer desire the control function served by the flag.

The following tables summarize the programming information presented in the *SDF Operations and Programming Guide* and provide the COBOL74 picture representations generated by the flags.

Table 6–1 contains the programmatic control default suffixes and contains flag setting information for each programmatic control.

Table 6–1. Programmatic Flag Suffixes and Settings

Programmatic Control	Default Suffix	Flag Setting
Cursor	-CURSOR	Set before a write operation
Data only	-DATA	Set before a write operation
Field suppress	-SUPPRESS	Set before a write operation
Flag groups	-FLAGS	Set before a write operation
Highlight	-HIGHLIGHT	Set before a write operation
Input/Output	-IOTYPE	Set before a write operation
No input	-NOINPUT	Set by the formlibrary with a read operation
Page	-PAGE	Set before a write operation
Specify	-SPECIFY	Set by the formlibrary with a read operation

Table 6–2 lists the programmatic flags, their COBOL74 picture representation, and valid values.

Table 6–2. Programmatic Flag Information

Flag	COBOL74 Picture Representation	Valid Values
Cursor	PIC 9(1) COMP	• 0 (No cursor positioning) • 1 (Position cursor)
Data only	PIC 9(1) COMP	• 0 (No data only) • 1 (Data only)
Field suppress	PIC 9(1) COMP	• 0 (Not suppressed) • 1 (Suppressed)
Flag groups	Not applicable	Not applicable

continued

Table 6–2. Programmatic Flag Information (cont.)

Flag	COBOL74 Picture Representation	Valid Values
Highlight	PIC 9(1) COMP	Fixed highlighting: • 0 (Not specified) • 1 (Specified) Variable highlighting: • 0 (None) • 1 (Bright) • 2 (Reverse) • 3 (Secure) • 4 (Underline) • 5 (Blink)
Input/Output	PIC 9(1) COMP	• 0 (Input and output) • 1 (Input and output) • 2 (Input only) • 3 (Output) • 4 (Output) transmittable
No input	PIC 9(1) COMP	• 0 (Data input) • 1 (No data input)
Page	PIC 9(1) COMP	Terminal page 1 - 9
Specify	PIC 9(4) COMP	• 0 (Not specified) • >0 (Specified) The value of the flag field indicates the cursor position within the data field.

Flag Group Generation

The names of the flag groups must follow COBOL74 naming conventions and be unique for each entity. The hyphen (-) in the name is part of the user-specified group flag suffix or flag suffix.

The syntax for the default name of a group of all flags is

```
<form name><-group flag suffix>
```

The syntax for the default name of individual groups for each type of flag is

```
<form name><-flag suffix><-group flag suffix>
```

You can reset flags to 0 (zero), as follows:

- In a group for all flags in a record or in a group for each type of flag, you use the hexadecimal value *@00@*.
- In individual flag fields, you can use the figurative constant LOW-VALUES.

You cannot use the constant LOW-VALUES for flag groups because the destination is alphanumeric, causing spaces rather than zeros to be placed in the group. Spaces in the group cause unexpected results.

See Also

- For information on specifying and using flag groups, refer to the *SDF Operations and Programming Guide*.
- For information on the use of the LOW-VALUES figurative constant, refer to Volume 1.

Example

The following example shows the use of flag groups in an application. The statement to reset all flags to the value 0 at the group level is

```
MOVE ALL @00@ TO FORM-1-FLAGS.
```

The statement to reset all the highlight flags to the value 0 at the flag group level is

```
MOVE ALL LOW-VALUES TO FORM-1-HIGHLIGHT-FLAGS.
```

In the example, FORM-1 has two fields, FIELD-1 and FIELD-2. If each field had all the possible programmatic flags set, and the form had all possible programmatic flags set, the 01 record in COBOL74 would appear as shown in the example.

```
Ø1 FORM-1.
   Ø2 FIELD-1                              PIC X(1Ø).
   Ø2 FIELD-2                              PIC 9(6) V9(2).
   Ø2 FORM-1-FLAGS.
     Ø3 FORM-1-PAGE-FLAGS.
        Ø4 FORM-1-PAGE.                    PIC 9 COMP.
     Ø3 FORM-1-SPECIFY-FLAGS.
        Ø4 FORM-1-SPECIFY                  PIC 9(4) COMP.
        Ø4 FIELD-1-SPECIFY                 PIC 9(4) COMP.
        Ø4 FIELD-2-SPECIFY                 PIC 9(4) COMP.
     Ø3 FORM-1-IOTYPE-FLAGS.
        Ø4 FIELD-1-IOTYPE                  PIC 9 COMP.
        Ø4 FIELD-2-IOTYPE                  PIC 9 COMP.
     Ø3 FORM-1-CURSOR-FLAGS.
        Ø4 FIELD-1-CURSOR                  PIC 9 COMP.
        Ø4 FIELD-2-CURSOR                  PIC 9 COMP.
     Ø3 FORM-1-SUPPRESS-FLAGS.
        Ø4 FIELD-1-SUPPRESS                PIC 9 COMP.
        Ø4 FIELD-2-SUPPRESS                PIC 9 COMP.
     Ø3 FORM-1-HIGHLIGHT-FLAGS.
        Ø4 FIELD-1-HIGHLIGHT               PIC 9 COMP.
        Ø4 FIELD-2-HIGHLIGHT               PIC 9 COMP.
     Ø3 FORM-1-DATA-FLAGS.
        Ø4 FORM-1-DATA                     PIC 9 COMP.
        Ø4 FIELD-1-DATA                    PIC 9 COMP.
        Ø4 FIELD-2-DATA                    PIC 9 COMP.
     Ø3 FORM-1-NOINPUT-FLAGS.
        Ø4 FIELD-1-NOINPUT                 PIC 9 COMP.
        Ø4 FIELD-2-NOINPUT                 PIC 9 COMP.
```

Using SDF with COMS

You can use SDF with COMS to take advantage of COMS direct windows. This combination gives you enhanced routing capabilities for forms and also enables preprocessing and postprocessing of forms.

The following guidelines explain the extensions you can include in your application when using SDF and COMS together:

- You can use the REDEFINES and SAME RECORD AREA options when the formlibrary is invoked in the WORKING-STORAGE SECTION for use with the COMS direct-window interface.

  The SAME RECORD AREA clause makes efficient use of record space, but an additional WORKING-STORAGE SECTION might be needed to save SDF information from the last SEND or RECEIVE statement for each form.

  The following example illustrates the use of the SAME RECORD AREA clause in a COBOL74 program using both SDF and COMS, where SAMPLELIB is the SDF formlibrary name:

```
001800 WORKING-STORAGE SECTION.
002500 01 COMS-MESSAGE-AREA.
002510     02 SDF-MESSAGE-KEY   PIC X(011).
002600     02 SDF-MESSAGE       PIC X(2500).
002620 01 SAMPLELIB FROM DICTIONARY; SAME RECORD AREA.
```

  The SDF-MESSAGE-KEY size should be the same size as the SDF input message key. The total COMS-MESSAGE-AREA size must be large enough to hold any expected COMS input message.

- Unisys recommends that you program the main processing loop so that the RECEIVE statement uses a working storage area for the input message. Messages or errors might arrive at the program from sources other than your formlibrary. Once the program determines that a valid input has been received from a form in your formlibrary, you can process the data received.

  In the following example, COMS-IN is the COMS header name:

```
006300      RECEIVE COMS-IN MESSAGE INTO COMS-MESSAGE-AREA.
006310*     CHECK FOR COMS ERRORS
```

  Once the COMS error checking has been processed, you can determine which form was used for input and move the data in COMS-MESSAGE-AREA to the SDF form record for further processing. For further information about COMS error checking, refer to the *COMS Programming Guide*.

In the following example, SDF-MESSAGE-KEY is checked for the form message key and then the SDF form SAMPLEFORM1 or SAMPLEFORM2 is processed.

Note: *A hyphen (-) is not allowed as a character in a COMS trancode. The corresponding COMS trancode with a length of 3 would be "ADD" for "ADD-ITEM" and "MOD" for "MODIFY-ITEM".*

Furthermore, only uppercase letters and numbers are allowed for COMS trancodes, while SDF allows almost any combination of uppercase, lowercase, and special characters, and blanks for a message key, providing that the first 17 characters of the Message Key field are unique.

```
006320      IF SDF-MESSAGE-KEY = "ADD-ITEM   "
006322         MOVE COMS-MESSAGE-AREA TO SAMPLEFORM1
006325         PERFORM ADD-ITEM
006330      ELSE
006340      IF SDF-MESSAGE-KEY = "MODIFY-ITEM"
006342         MOVE COMS-MESSAGE-AREA TO SAMPLEFORM2
006350         PERFORM MODIFY-ITEM.
```

- Unisys recommends that you process all fields containing the *Specify* programmatic control before you check for *No input* programmatic control data and process other form input data.
- You must use the FORM-KEY function to move the form key into the first word of the conversation area of output. In the following example, COMS-OUT is the COMS header name and SAMPLEFORM2 is the form name. The FORM-KEY syntax must precede a SEND statement from the form. The following is an example of the code:

```
006355
006360   MOVE 1                       TO COMS-OUT-COUNT
006400   MOVE COMS-IN-STATION         TO COMS-OUT-DESTINATION
006700   MOVE 60                      TO COMS-OUT-TEXT-LENGTH
006720   MOVE SDF-AGENDA-DESIGNATOR TO COMS-OUT-AGENDA
006740   MOVE FORM-KEY(SAMPLEFORM2) TO COMS-OUT-CONVERSATION
006800   SEND COMS-OUT FROM SAMPLEFORM2
006810*  CHECK FOR COMS-OUT SEND ERRORS
007100 END-OF-JOB.
```

- You must do the following to transmit a default form (that is, a form with only default values):
  - Move spaces to display items.
  - Move zeros to numeric items and programmatic control.
- You cannot use a SEND or RECEIVE statement between BEGIN-TRANSACTION and END-TRANSACTION statements. Doing so violates the rules of synchronized recovery. COMS will not allow you to send a request to a station for data correction while your program is in transaction mode.

See Also

- Refer to the *COMS Programming Guide* for detailed information on the use of the COMS direct-window interface and for an illustration of the use of SDF and COMS in an application.
- For information about the extensions used with COMS, refer to the Section 3, "Using the COMS Program Interface."
- See "Using the FORM-KEY Function" earlier in this section for more information about that function.

SDF Sample Programs

The following are sample programs illustrating different uses of the SDF program interface. For information about handling remote file errors in an application program, refer to the *SDF Operations and Programming Guide*.

SDF Program Using the READ Statement

Example 6–1 is a program that uses a remote file and specific forms. The program contains a READ FORM statement for a default form. The formlibrary was created using SDF.

Sections of the program are explained in comment lines within the code.

```
001000 IDENTIFICATION DIVISION.
007000
008000 ENVIRONMENT DIVISION.
009000 CONFIGURATION SECTION.
011100
011105
011110*************************************************************
001115* Specify the dictionary that stores the formlibrary.   *
001120*************************************************************
011125
011200 SPECIAL-NAMES.
011300   DICTIONARY IS "SCREENDESIGN".
011500
011600
011700*********************************************
011800* Declare the remote file.                  *
011900*********************************************
011950
012000 INPUT-OUTPUT SECTION.
013000 FILE-CONTROL.
014000     SELECT REMFILE
014500       ASSIGN TO REMOTE.
015000
015100
```

Example 6–1. SDF Sample Program with a READ Statement

```
015200*****************************************************************
015300* Invoke the formlibrary in the FILE SECTION, and associate     *
015400* it with the file.  This action also causes the proper maximum *
015500* record size (MAXRECSIZE) to be designated for the file.       *
015600*****************************************************************
015700
016000 DATA DIVISION.
017000 FILE SECTION.
018000 FD REMFILE.
021200 01 SAMPLELIB  FROM DICTIONARY.
021400
021450*****************************************************************
021460* Include  program record descriptions for all forms in the     *
021470* formlibrary that are automatically invoked and copied into the*
021480* program during compilation (see the following dictionary data *
021490* lines identified with the D flag).                            *
021600*****************************************************************
023000
000100*--DICTIONARY                                              D
000110*--DICTIONARY FORMLIST <SAMPLELIB>.                        D
000130 01 SAMPLEFORM1.                                          D
000140   02 ACTION PIC X(10).                                   D
000150   02 ACCOUNT-NO PIC 9(9).                                D
000160   02 NAME PIC X(15).                                     D
000170   02 STREET PIC X(25).                                   D
000180   02 CITY PIC X(15).                                     D
000190   02 STATE PIC X(2).                                     D
000200   02 ZIP PIC 9(9).                                       D
024000 PROCEDURE DIVISION.
024500 MAIN-PARA.
025000
025200*******************************************************
025400*     Open remote file I/O                            *
025600*******************************************************
026000  OPEN I-O REMFILE.
026100
026200*************************************************************
026300* WRITE FORM statements are not necessary when you are      *
026350* sending a form with only default values to the            *
026400* screen.  The statement writes form SAMPLEFORM1            *
026500* with its default values in the fields, and then reads     *
026550* the form.                                                 *
026600*************************************************************
032000  READ FORM REMFILE USING SAMPLEFORM1 FROM DEFAULT FORM.
038000STOP RUN.
```

Example 6–1. SDF Sample Program with a READ Statement (cont.)

SDF Program Using the WRITE and READ Statements

Example 6–2 is a sample program that uses a remote file and specific forms. The program contains WRITE and READ statements. The formlibrary was created using SDF.

Sections of the program are explained in comment lines within the code.

```
11000Ø IDENTIFICATION DIVISION.
11100Ø
12000Ø ENVIRONMENT DIVISION.
13000Ø CONFIGURATION SECTION.
13200Ø
13300Ø*************************************************************
13400Ø* Specify the dictionary that stores the formlibrary.  *
13500Ø*************************************************************
13600Ø
14000Ø SPECIAL-NAMES.
15000Ø   DICTIONARY IS "SCREENDESIGN".
16000Ø
16100Ø*********************************************
16200Ø* Declare the remote file.                   *
16300Ø*********************************************
16400Ø
17000Ø INPUT-OUTPUT SECTION.
18000Ø FILE CONTROL.
19000Ø   SELECT REMFILE ASSIGN TO REMOTE.
19100Ø
19200Ø*********************************************************************
19300Ø* Invoke the formlibrary in the FILE SECTION, and associate    *
19400Ø* it with the file.  This action also causes the proper maximum*
19500Ø* record size (MAXRECSIZE) to be designated for the file.      *
19600Ø*********************************************************************
19700Ø
21000Ø DATA DIVISION.
22000Ø FILE SECTION.
23000Ø FD REMFILE.
24000Ø Ø1 SAMPLELIB FROM DICTIONARY.
24100Ø
24200Ø*************************************************************************
24500Ø* Include program record descriptions for all forms in the       *
24600Ø* formlibrary that are automatically invoked and copied into the  *
24700Ø* program during compilation (see the following dictionary data   *
24800Ø* lines identified with the D flag).                             *
24900Ø*************************************************************************
24950Ø
00100Ø *--DICTIONARY                                          D
00110Ø *--DICTIONARY FORMLIST <SAMPLELIB>.                    D
00120Ø  Ø1 SAMPLEFORM1.                                       D
00130Ø    Ø2 ACTION PIC X(1Ø).                                D
```

Example 6–2. SDF Sample Program with READ and WRITE Statements

```
00140     02 ACCOUNT-NO PIC 9(9).                          D
00150     02 NAME PIC X(15).                               D
00160     02 STREET PIC X(25).                             D
00180     02 CITY PIC X(15).                               D
00190     02 STATE PIC X(2).                               D
00200     02 ZIP PIC 9(9).                                 D
24000 PROCEDURE DIVISION.
24500 MAIN-PARA.
25000
26000*************************************************************
27000* Opens the remote file I/O.                               *
27100* If a file is not open, the formlibrary opens the         *
27200* file at the time of the write or read operation,         *
27300* using whatever file attributes are set.                  *
28000*************************************************************
29000
30000 OPEN I-O REMFILE.
40000
41000*************************************************
42000* The WRITE FORM statement writes a form from the*
43000* formlibrary to the station.  The statement     *
44000* uses a form name.                             *
45000*************************************************
46000
47000 WRITE FORM SAMPLEFORM1.
48000
49000*************************************************
50000* The READ FORM statement reads the form back to *
51000* the program when the user transmits.          *
52000*************************************************
53000
54000 READ FORM REMFILE USING SAMPLEFORM1.
55000
56000 STOP RUN.
```

Example 6–2. SDF Sample Program with READ and WRITE Statements (cont.)

SDF Program Using Programmatic Controls

Example 6–3 is a sample program that uses SDF, a remote file, specific forms, and programmatic controls.

Sections of the program are explained in comment lines within the code.

As you read this example, you should be aware of the following programming information about flags. (The example does not contain lines of code demonstrating this information.)

- If your forms use either *Specify* or *No input* flags, your program should check to see if these flags are set after a READ operation before you begin processing data from the form fields.
- If you use both *Specify* and *No input* flags, check the *Specify* flag first.
- If the value in any *Specify* flag field is greater than zero, the values in the data fields of the form are unchanged from the previous operation.

```
000200 IDENTIFICATION DIVISION.
000300 ENVIRONMENT DIVISION.
000400 CONFIGURATION SECTION.
000410*****************************************************************
000420* Specify the dictionary that stores the formlibrary.           *
000430*****************************************************************
000500 SPECIAL-NAMES.
000600   DICTIONARY IS "SCREENDESIGN".
000800 INPUT-OUTPUT SECTION.
000900 FILE-CONTROL.
000910*****************************************************************
000920* Declare the remote and disk files.                            *
000930*****************************************************************
001000 SELECT MTTERM ASSIGN TO REMOTE.
001100 SELECT DISK-FILE ASSIGN TO DISK.
001200
001300 DATA DIVISION.
001350
001400 FILE SECTION.
001450
001510*****************************************************************
001520* Invoke the formlibrary in the FILE SECTION.                   *
001530*****************************************************************
001500 FD MTTERM.
001550*****************************************************************
001575* The following attributes assure the correct record size       *
001600* for  write operations in which the form record can be         *
001550* longer than 80 characters.                                    *
001675*****************************************************************
001730   BLOCK CONTAINS 2200 CHARACTERS.
001740   RECORD CONTAINS 2200 CHARACTERS.
001745   VALUE OF MAXRECSIZE IS 2200.
001750   VALUE OF FILETYPE IS 3.
001760   VALUE OF MYUSE IS IO.
001770   CODE-SET IS EBCDIC.
001780
001800 01 VOTERLIB FROM DICTIONARY.
001900
```

Example 6–3. SDF Sample Program with Programmatic Controls

```
001910*****************************************************************
001920* Include  program record descriptions for all the forms in the*
001930* formlibrary that are automatically invoked and  copied into  *
001940* the program during compilation (see the following dictionary *
001945* data lines identified with the D flag).                       *
001950*****************************************************************
001960
000100 *--DICTIONARY                                                  D
000110 *--DICTIONARY FORMLIST<VOTERLIB>.                              D
000130  01 VRFORM.                                                    D
000140    02 PRECINCT PIC 9(4).                                      D
000150    02 LOCATION PIC X(28).                                     D
000160    02 VRNAME PIC X(54).                                       D
000170    02 ADDRESS PIC X(54).                                      D
000180    02 CITY PIC X(24).                                         D
000190    02 COUNTY PIC X(24).                                       D
000200    02 CONGRESSDIS PIC 9(4).                                   D
000210    02 REPRESDIS PIC 9(4).                                     D
000220    02 SENATEDIS PIC 9(4).                                     D
000230    02 COMMISSDIS PIC 9(4).                                    D
000240    02 VRDATE PIC 9(6).                                        D
000250    02 CLERK PIC X(29).                                        D
000260    02 VRNAME-CURSOR PIC 9(1) COMP.                            D
000270    02 VRNAME-HIGHLIGHT PIC 9(1) COMP.                         D
002000 FD DISK-FILE.
002100 01 DATA-RECORD                          PIC X(300).
002200
002300 WORKING-STORAGE SECTION.
002400
002500 PROCEDURE DIVISION.
002600
002700 MAIN-PARA.
002800
002900 ******************************************************
003000 *Open the remote file and disk file.                   *
003100 ******************************************************
003200 OPEN I-O MTTERM.
003300 OPEN OUTPUT DISK-FILE.
003400
003500*****************************************************************
003510* Move values to the fields of the form so that a form can be *
003520* written to be displayed with those values.                    *
003530*****************************************************************
003540
003500 MOVE SPACES TO VRFORM.
003600 MOVE ZEROS TO VRDATE.
003610*********************************************************
003620* The following code prevents highlighting from being   *
003630* set incorrectly.                                      *
003630*********************************************************
```

Example 6–3. SDF Sample Program with Programmatic Controls (cont.)

```
003610 MOVE Ø TO VRNAME-CURSOR
003620 MOVE Ø TO VRNAME-HIGHLIGHT
003700
003800 ************************************************************
003900 * Perform a loop to enter and store data in a disk file. *
004000 ************************************************************
004100
004200 PERFORM DATA-ENTRY UNTIL CLERK = "DONE".
004300
004400 END-MAIN-PARA.
004500 STOP RUN.
004600
004700 ************************************************************
004800 *This is the beginning of the data-entry loop.      *
004900 ************************************************************
005000
005010************************************************************
005020* Move values to the fields indicated.  The last values*
005030* entered in the form are displayed for those fields   *
005040* whose values are not changed explicitly.             *
005050************************************************************
005060
005100 DATA-ENTRY.
005200
005300 MOVE SPACES TO VRNAME.
005400 MOVE SPACES TO ADDRESS.
005500 MOVE SPACES TO CITY.
005600 MOVE SPACES TO COUNTY.
005700
005710*********************************************************************
005720* Use the WRITE statement to write the form name.  The READ     *
005730* statement reads the form from the terminal.  MOVE and WRITE   *
005740* statements store the form into a record file.                 *
005750*********************************************************************
005760
005800 WRITE FORM VRFORM
005810    ON ERROR STOP RUN.
005900 READ FORM MTTERM USING VRFORM
005910    ON ERROR STOP RUN.
006000 MOVE VRFORM TO DATA-RECORD.
006100 WRITE DATA-RECORD.
006110
006120*********************************************************************
006130* Uses programmatic control for cursor position (with the flag  *
006140* name and suffix VRNAME-CURSOR) and places the cursor in the   *
006150* VRNAME field when the form is displayed.  Note that the users *
006160* can tab back to the first fields to enter data if they wish. *
006170*********************************************************************
006180
006190
```

Example 6–3. SDF Sample Program with Programmatic Controls (cont.)

```
006200 MOVE 1 TO VRNAME-CURSOR.
006300
006400 END-DATA-ENTRY.
```

Example 6–3. SDF Sample Program with Programmatic Controls (cont.)

SDF Program Using Message Keys

Example 6–4 shows the use of message keys and of the independent record area in a COBOL74 program using SDF. SAMPLELIB is the SDF formlibrary name, and SAMPLEFORM1 and SAMPLEFORM2 are the form names. The Action field is defined as the Message Key field.

Sections of the program are explained in comment lines within the code.

```
000200 IDENTIFICATION DIVISION.
000300
000400 ENVIRONMENT DIVISION.
000500 CONFIGURATION SECTION.
000600
000610*************************************************************
000620* Specify the dictionary that contains the formlibrary. *
000630*************************************************************
000700 SPECIAL-NAMES.
000800   DICTIONARY IS "SCREENDESIGN".
001000
001010*****************************************
001020* Declare the remote file.              *
001030*****************************************
001040
001100 INPUT-OUTPUT SECTION.
001200 FILE-CONTROL.
001300     SELECT REMFILE
001400       ASSIGN TO REMOTE.
001500
001510*************************************************************
001520* Invoke the formlibrary in the FILE SECTION.         *
001530*************************************************************
001540
001600 DATA DIVISION.
001700 FILE SECTION.
001800 FD REMFILE
001810   BLOCK CONTAINS 2500 CHARACTERS
001820   RECORD CONTAINS 2500 CHARACTERS
001830   VALUE OF FILETYPE IS 3
001840   VALUE OF MYUSE IS IO
001850   CODE-SET IS EBCDIC
001860
002000 01 SAMPLELIB  FROM DICTIONARY.
002200
```

Example 6–4. SDF Sample Program with Message Keys

```
002210*****************************************************************
002220* Include  program record descriptions  of all forms in the      *
002230* formlibrary that are automatically invoked and copied into     *
002240* the program during compilation (see the following dictionary   *
002245* data lines identified with the D flag).                        *
002250*****************************************************************
002260
000100*--DICTIONARY                              D
000110*--DICTIONARY FORMLIST <SAMPLELIB>.        D
000130 01 SAMPLEFORM1.                          D
000140    02 ACTION PIC X(11).                  D
000150    02 ACCOUNT-NO PIC 9(9).               D
000160    02 NAME PIC X(15).                    D
000170    02 STREET PIC X(25).                  D
000180    02 CITY PIC X(15).                    D
000190    02 STATE PIC X(2).                    D
000200    02 ZIP PIC 9(9).                      D
000210 01 SAMPLEFORM2.                          D
000220    02 ACTION PIC X(11).                  D
000230    02 ACCOUNT-BALANCE PIC 9(9).          D
000240    02 PAYMENT-DUE-DATE PIC X(6).         D
000250    02 DATE-LAST-PAYMENT PIC X(6).        D
000260    02 FINANCE-CHARGE PIC 9(5).           D
002262
002264*****************************************************************
002266* The SDF-MESSAGE-KEY size should be the same size as the SDF    *
002268* input message key. The total SDF-MESSAGE-AREA size must be     *
002270* large enough to hold any SDF input message.                    *
002272*****************************************************************
002274
002280 WORKING-STORAGE SECTION.
002282 01 SDF-MESSAGE-AREA.
002284    02 SDF-MESSAGE-KEY   PIC X(011).
002286    02 SDF-MESSAGE       PIC X(2500).
002288
002300*****************************************************************
002306* You program the main processing loop so that the RECEIVE       *
002310* statement uses a working storage area, SDF-MESSAGE-AREA, for   *
002314* the input message. Messages or errors might arrive for the     *
002316* program from your formlibrary.                                 *
002320*****************************************************************
002322
002300 PROCEDURE DIVISION.
002400 MAIN-PARA.
002500
002600*************************************
002700*   Open remote file I/O.           *
002800*************************************
002900  OPEN I-O REMFILE.
003000
```

Example 6–4. SDF Sample Program with Message Keys (cont.)

```
003110*****************************************************************
003120* Move values to the fields of the form so that a write can be*
003120* done to display the form with those values.                   *
003130*****************************************************************
003200
003300  MOVE SPACES TO NAME.
003400  MOVE SPACES TO STREET.
003500  MOVE SPACES TO CITY.
003600  MOVE SPACES TO STATE.
003700  MOVE ZEROS TO ACCOUNT-NO.
003800  MOVE ZEROS TO ZIP.
003900
004000*****************************************************************
004100* These are WRITE FORM and READ FORM statements that explicitly  *
004110* state the form name.  Use the WRITE FORM statement to write    *
004120* the form from the formlibrary to the terminal with the changed *
004130* values.  The READ FORM statement uses the form name.  The form *
form40* name can be used even if message keys have been defined for    *
004150* the forms in the formlibrary.                                  *
004200*****************************************************************
004250
004300  WRITE FORM SAMPLEFORM1.
004350    ON ERROR STOP RUN.
004400  READ FORM REMFILE USING SAMPLEFORM1
004450    ON ERROR STOP RUN.
004500
004600*****************************************************************
004700* The forms in the formlibrary contain message keys; therefore,  *
004800* they are self-identifying forms. Note the syntax of the READ   *
004850* FORM statement.                                                *
004900*                                                                *
004910* Use the WRITE FORM statement to write the specific form when   *
004920* using message keys to identify input forms. The READ FORM      *
004930* statement uses the formlibrary name and a separate working     *
004940* storage area instead of the form name.  The program examines   *
004950* the field SDF-MESSAGE-KEY that contains the message key to     *
004955* identify the form.                                             *
004960*****************************************************************
004970
005100  READ FORM REMFILE USING SAMPLELIB
005105       INTO SDF-MESSAGE-AREA
005110    ON ERROR STOP RUN.
005115
005125*****************************************************************
005130* The value of the Message Key field SDF-MESSAGE-KEY determines  *
005135* the conditional function to be performed.                      *
005145*                                                                *
005150* Once the program determines that a valid input has been        *
005155* received from a form in your formlibrary, you can process the  *
005160* data received. The example uses the STOP RUN statement to      *
```

Example 6–4. SDF Sample Program with Message Keys (cont.)

```
005165* handle any error.                                                *
005170*                                                                  *
005175* After the SDF errors have been processed, you can determine     *
005180* which form was used for input and move the data in the          *
005185* SDF-MESSAGE-AREA to the SDF form record for further processing.*
005190* In the following example, SDF-MESSAGE-KEY is checked for the    *
005195* form message key and then the SDF form SAMPLEFORM1 or           *
005200* SAMPLEFORM2 is processed.                                        *
005205*                                                                  *
005210* Move the data in SDF-MESSAGE-AREA to the SDF form record before*
005215* further processing.                                              *
005220*******************************************************************
005230
005250  IF SDF-MESSAGE-KEY = "ADD-ITEM"
005252    MOVE SDF-MESSAGE-AREA TO SAMPLEFORM1
005255    PERFORM ADD-ITEM
005260  ELSE
005265  IF SDF-MESSAGE-KEY = "MODIFY-ITEM"
005267    MOVE SDF-MESSAGE-AREA TO SAMPLEFORM2
005270    PERFORM MODIFY-ITEM.
005272
005274*******************************************************************
005276* First process all Specify programmatic control data for the     *
005278* form before checking for No input programmatic control data and*
005280* processing other input data.                                     *
005282*******************************************************************
005284
005290  ADD-ITEM.
005295
005300****************************************
005305* Insert code to add an item.          *
005310****************************************
005311
005315  MODIFY-ITEM.
005316
005320****************************************
005325* Insert code to modify an existing    *
005330* item.                                *
005335****************************************
005340
005400STOP RUN.
```

Example 6–4. SDF Sample Program with Message Keys (cont.)

Section 7
Using the SDF Plus Program Interface

The Screen Design Facility Plus (SDF Plus) is a user interface management system that gives you the ability to define a complete form-based user interface for an application system. It is a programming tool for designing and processing forms simply and efficiently. SDF Plus provides form processing that eliminates the need for complicated format language or code, and validates data entered on forms by application users.

The program interface developed for SDF Plus includes

- Extensions that enable you to read and write form records or form record libraries easily
- Extensions that enable you to send and receive form records or form record libraries easily
- Extensions that enable you to invoke form record library descriptions into your program as COBOL74 declarations

This section provides information about the extensions developed for SDF Plus. Each extension is presented with its syntax and examples; sample programs are also included.

For an alphabetized list of the extensions, see "SDF Plus Extensions" in Section 1, "Introduction to COBOL74 Program Interfaces." Refer to the *A Series Screen Design Facility Plus (SDF Plus) Capabilities Manual* for information defining the concepts and principles of SDF Plus. For information on general implementation and operation considerations, and for information on describing the process of migrating SDF form libraries to SDF Plus, refer to the *A Series Screen Design Facility Plus (SDF Plus) Installation and Operations Guide*. Consult the *A Series Screen Design Facility Plus (SDF Plus) Technical Overview* for information on general programming concepts and considerations.

You can use SDF Plus with the Advanced Data Dictionary System (ADDS), the Semantic Information Manager (SIM), and the Communications Management System (COMS). Refer to the product documentation for information on the concepts and programming considerations for using these products with SDF Plus. More information on the extensions used with these products is provided in Section 2, "Using the ADDS Program Interface;" Section 3, "Using the COMS Program Interface;" and Section 8, "Using the SIM Program Interface," in this volume.

Understanding Interface Elements

The interface between COBOL74 application programs and SDF Plus form libraries can be achieved through either the remote file or the COMS interface. If you use the remote file interface, you can interact with SDF Plus applications by means of remote files. If you use the COMS interface, you can interact with SDF Plus applications through COMS windows, and have access to all COMS capabilities and features.

SDF Plus interface elements include

- Form record libraries
- Form records
- Form record numbers
- Transaction types
- Transaction numbers

Form Record Libraries

A form record library is a collection of form records and transaction types. You create this collection by using SDF Plus, and you can store the form record library in either the ADDS data dictionary or the SDF Plus dictionary. Form record libraries can be invoked by using COBOL74 syntax. See the *SDF Plus Installation and Operations Guide* for information on using SDF Plus to create a form record library. The form records can then be used in various COBOL74 statements to transfer data. Multiple form record libraries can be invoked in the COBOL74 program.

Form Records

Form records are elements of form record libraries. Form records represent records of data. This data is used either to output data from a form or to input data to a form. A form can require several form records; therefore, a one-to-one relationship between forms and form records does not exist.

In some manuals, the term *message type* is a synonym for *form record.*

Forms and form processing are established through the use of SDF Plus. The COBOL74 program reads and writes data to the forms. This arrangement provides complete separation between data entered on a terminal and actions completed within the program. A user interface can be completely reconstructed through the use of SDF Plus without modifying the application program, providing the form records are not changed.

Form Record Numbers

A form record number is a unique integer assigned at compile time for each form record in a form record library.

In some manuals, the term *message type number* is a synonym for *form record number.*

A form record number for a form record library is an attribute of the form record library. Form record numbers determine I/O operations for form record libraries, allowing the form record to be specified at run time.

A self-identifying read is used when the executing program has not established which form record in a specific form record library is being read. The program must access the

form record number attribute for the form record library to determine the form record that has been read.

A self-identifying write enables the executing program to specifically identify the form record to be written by placing the appropriate form record number value into the form record number attribute of the form record library.

Transaction Types

Transaction types are elements of form record libraries. A transaction type contains a pair of form records: an input form record and an output form record. A transaction type identifies the relationship of the two form records that are under it, namely, the input form record to the transaction type and the output form record from the transaction type.

Transaction Numbers

Transaction numbers are similar to form record numbers. A transaction number is a unique integer assigned at compile time to each transaction in a form record library.

A transaction number for a form record library is an attribute of the form record library. This attribute contains the transaction number of a specific transaction type. Transaction numbers provide another means to determine I/O operations for form record libraries at run time.

After a self-identifying read, the application program must access the transaction number attribute of the form record library being read in order to determine the transaction type that has been executed.

Example

The following example shows the invocation of form record library MSGKEYS. The syntax is constructed by the compiler. The form record library contains two transaction types and two forms. Each form has the same response message (MSGKEYSSR).

```
001800 01 MSGKEYS FROM DICTIONARY; DIRECTORY IS "SMITH".
001900*--DICTIONARY DIRECTORY  :SMITH.
002000*--DICTIONARY FORMLIST <MSGKEYS>.
002100*--SDF TRANSACTION (FORM1TT).
002200 01 FORM1.
002300       04 KEYFIELD  PIC X(5).
002400       04 DATAFIELD PIC X(4).
002450       04 QUITFIELD PIC X(1).
002500 01 MSGKEYSSR.
002600       04 MSGKEYSSRF PIC X(1).
002700*--SDF TRANSACTION (FORM2TT).
002800 01 FORM2.
002900       04 KEY2FLD  PIC X(5).
003000       04 DATA2FLD PIC 9(4).
003100       04 QUIT2FLD PIC X(1).
003200*01 MSGKEYSSR.
003300*01 FORM1.
003400*01 FORM2.
```

Identifying the Dictionary

The dictionary is identified in the SPECIAL-NAMES paragraph of the ENVIRONMENT DIVISION, using the DICTIONARY clause.

Note: *A program can invoke only one dictionary. Therefore, if a program accesses both a SIM database (from a dictionary) and SDF forms, both must be in the same dictionary.*

General Format

The general format for the DICTIONARY clause is as follows:

```
[, DICTIONARY IS literal-1]
```

Explanation of Format Elements

DICTIONARY

This clause enables you to identify the function name of the dictionary library.

literal-1

This format element is the function name that is equated to a library source code using the SL (support library) system command. See the *SDF Plus Installation and Operations Guide* for instructions on equating these names.

Example

The following example shows code that identifies the SDFPLUSDICT dictionary:

```
001000 IDENTIFICATION DIVISION.
007000
008000 ENVIRONMENT DIVISION.
009000 CONFIGURATION SECTION.
011100
011200 SPECIAL-NAMES.
011300   DICTIONARY IS "SDFPLUSDICT".
```

Invoking Data Descriptions

A data-description entry specifies the characteristics of a particular data item. There are six options: SAME RECORD AREA, SEPARATE RECORD AREA, VERSION, DIRECTORY, REDEFINES, and GLOBAL.

In the DATA DIVISION, either the SAME RECORD AREA option or the SEPARATE RECORD AREA option can be used, but not both; all other options can be present.

The clauses and options can be written in any order. However, form-record-library-name-1 must immediately follow the level number, and the REDEFINES option, when used, must immediately follow form-record-library-name-1.

General Format

The general format of the data-description entry is as follows:

```
level-number form-record-library-name-1
FROM DICTIONARY
[ { ; SAME RECORD AREA      }
  { ; SEPARATE RECORD AREA  }
  ; VERSION IS literal-1
  ; DIRECTORY IS literal-2
  ; REDEFINES form-record-library-name-2
  ; GLOBAL                 ].
```

Explanation of Format Elements

level-number

The level must be 01.

form-record-library-name-1

This format element identifies a dictionary structure that is a collection of record descriptions defining screen formats. It must immediately follow the level number.

FROM DICTIONARY

The FROM DICTIONARY clause is used in the DATA DIVISION to obtain a form record library from the dictionary. If the LIST compiler control option is also set, the form data descriptions are listed in the DATA DIVISION of the COBOL74 program.

SAME RECORD AREA

This option applies to SDF Plus form records only when the option is used in the DATA DIVISION; however, this option cannot be used in the FILE SECTION of the DATA DIVISION. This option invokes all form record descriptions in the form record library as redefinitions of the first form record description in the form record library.

SEPARATE RECORD AREA

This option applies to SDF Plus form records only when the option is used in the DATA DIVISION; however, this option cannot be used in the FILE SECTION of the DATA DIVISION. The SEPARATE RECORD AREA option is used to invoke each form record in the form record library as a separate data description with its own record area.

VERSION

This option imports the version of the form record library being used.

literal-1

This format element must be a numeric literal, consisting of up to six digits, that identifies a version of the form record library.

DIRECTORY

This option specifies the directory under which the form record library is stored in the data dictionary. This option enables you to access form record libraries stored under different usercodes.

literal-2

This format element must describe the directory under which the form record library is stored in the data dictionary.

REDEFINES form-record-library-name-2

The REDEFINES option applies to SDF Plus form records only when the option is used in the DATA DIVISION; however, this option cannot be used in the file description (FD) entry of the DATA DIVISION. The REDEFINES option enables different data descriptions to describe the same memory. This option allows multiple form record libraries to have the same record area. Multiple redefinitions of the same form record library are also allowed.

The REDEFINES option can be used to redefine only form record libraries that previously have been invoked using the SAME RECORD AREA option.

The form record library invocation using the REDEFINES option must immediately follow either the form-record-library-name-2 invocation that it is redefining, or another form record library invocation that is using a REDEFINES option for form-record-library-name-2.

GLOBAL

This option is required in subprograms to reference form record libraries that were declared in the host program. You cannot declare a local form record library in a subprogram unless there is a matching form record library in the host program.

Considerations for Use

Only a form record library can be invoked directly from the dictionary; neither a transaction nor a form record can be invoked directly.

Form record libraries can be invoked within the FILE SECTION, the WORKING-STORAGE SECTION, the LINKAGE SECTION, or the LOCAL-STORAGE SECTION.

The SAME RECORD AREA, SEPARATE RECORD AREA, and REDEFINES options can be used only with form record library invocations in the WORKING-STORAGE SECTION, the LINKAGE SECTION, and the LOCAL-STORAGE SECTION. All other uses of the options cause a syntax error.

The SAME RECORD AREA and SEPARATE RECORD AREA options cannot be used in the same form record library.

A form record library cannot be given an alias using the INVOKE clause.

See Also

See "Identifying Specific Entities" and "Assigning Alias Identifiers" in Section 2, "Using the ADDS Program Interface," for additional information on these two topics.

Examples

The following examples illustrate the use of the FROM DICTIONARY clause.

- In this example, the form record library SAMPLELIB is imported from the dictionary:

```
DATA DIVISION.
FILE SECTION.
FD REMFILE.
Ø1 SAMPLELIB  FROM DICTIONARY.
```

- In this example, version 2 of form record library SAMPLELIB is imported from the data dictionary. The directory is SMITH.

```
DATA DIVISION.
FILE SECTION.
FD REMFILE.
Ø1 SAMPLELIB  FROM DICTIONARY
        VERSION IS 2
        DIRECTORY IS "SMITH".
```

- In this example, the form record library SAMPLELIB is redefined as SAMPLELIB3:

```
DATA DIVISION.
FILE SECTION.
FD REMFILE.
WORKING-STORAGE SECTION.
Ø1 SAMPLELIB FROM DICTIONARY.
Ø1 SAMPLELIB3 FROM DICTIONARY;
         REDEFINES SAMPLELIB.
```

- In this example, the form record library SAMPLELIB is redefined as SAMPLELIB4 and SAMPLELIB5:

```
DATA DIVISION.
FILE SECTION.
FD REMFILE.
Ø1 REM-REC  PIC X(2400).
WORKING-STORAGE SECTION.
Ø1 SAMPLELIB FROM DICTIONARY; SAME RECORD AREA.
Ø1 SAMPLELIB4 FROM DICTIONARY;
         REDEFINES SAMPLELIB.
Ø1 SAMPLELIB5 FROM DICTIONARY;
         REDEFINES SAMPLELIB.
```

Selecting a Global Remote File

The SELECT statement is used to include a global remote file in your program. The statement is used in the FILE-CONTROL paragraph under the INPUT-OUTPUT SECTION of the ENVIRONMENT DIVISION.

General Format

The general format of the SELECT statement is as follows:

SELECT GLOBAL file-name-1 ASSIGN TO REMOTE.

Explanation of Format Elements

GLOBAL clause and file-name-1

This clause is used to import global form record libraries to subprograms. The global form record libraries and the remote file (file-name-1) are declared in host programs.

Examples

The remote file REMFILE is declared in the host program as follows:

```
INPUT-OUTPUT SECTION.
FILE-CONTROL.
       SELECT REMFILE
          ASSIGN TO REMOTE.
```

In the subprogram, REMFILE is declared as follows:

```
INPUT-OUTPUT SECTION.
FILE-CONTROL.
       SELECT GLOBAL REMFILE
           ASSIGN TO REMOTE.
```

READ FORM Statement

The READ FORM statement causes a form record to be read from the specified remote file and stored in the specified buffer.

The appropriate form record library reads the form record, validates the record, performs screen error handling, and returns to the program the valid form record or any detected error condition.

When the logical records of a file are described with more than one record description, these records automatically share the same storage area. This sharing is equivalent to an implicit redefinition of the area. The contents of any data items that are beyond the range of the current data record are undefined after you execute the READ FORM statement.

The implied move operation does not occur if the READ FORM statement cannot be executed. Any subscripting or indexing associated with the identifier is evaluated after the record is read and immediately before the record is moved to the data item.

General Format

The general format of the READ FORM statement is as follows:

```
READ FORM file-name-1 USING {form-record-library-name-1                          }
                            {form-record-name-1 [FROM DEFAULT FORM]              }
[INTO identifier-1]
[; ON ERROR {imperative-statement-1 }].
            {conditional-statement-1}
```

Explanation of Format Elements

file-name-1

The file must be open in the input or I/O mode at the time this statement is executed. When the READ FORM statement is executed, the value of the file status data item associated with file-name-1 is updated. If the read is successful, the data is made available in the record storage area associated with file-name-1. For general information on opening files, refer to Volume 1.

The storage area associated with identifier-1 and the record area associated with file-name-1 must not be the same storage area.

USING form-record-library-name-1

This clause is used to perform a self-identifying read. A self-identifying read is performed to read a form record when you do not know in advance the form record that is to be written. After a self-identifying read, you can check the form record number attribute of the form record library to determine the form record that was read.

USING form-record-name-1

This clause is used for reading specific form records.

When form-record-name-1 is invoked in the DATA DIVISION and not in the file description for file-name-1, the data is read into the file-name-1 record area and transferred to form-record-name-1. To avoid truncating the trailing characters in the message, you should make sure the record description for file-name-1 is as large as the declaration of the largest form to be used with the file.

FROM DEFAULT FORM

This option can be used only with a form record name. When the FROM DEFAULT FORM option is specified, the form record library writes a form record with default values in each field before performing the read and record validation.

INTO identifier-1

When the INTO option is used, the record being read is available in both the input record area and the data area associated with the identifier. For more information on the MOVE statement, refer to Volume 1.

If the INTO option is specified, the record being read is moved from the record area to the area specified by identifier-1, according to the rules specified for the MOVE statement. The sending area is considered to be a group item equal in size to the maximum record size for this file. The INTO option can be used only if form-record-name-1 or form-record-library-name-1 is invoked in the file description associated with file-name-1.

ON ERROR

The ON ERROR option actions are the same as for an Invalid Key condition. For information on the Invalid Key condition, see Volume 1. For general information on error handling with remote files, see the *SDF Plus Technical Overview*.

Considerations for Use

Two run-time errors specific to form records can be handled programmatically by using the FILE STATUS clause in the INPUT-OUTPUT SECTION. For information on the FILE STATUS clause, see Volume 1. A form error message (82) is returned when either of the following occurs:

- A read is executed on a specific form record and that form record no longer resides in the form record library.
- The compile-time form record version does not equal the run-time form record version.

Examples

The following examples illustrate the use of the READ FORM statement.

- The following example shows a READ FORM statement that causes a default form record to be read:

```
READ FORM REMFILE-A USING FORMLIB-A
     FROM DEFAULT FORM.
```

- This example shows a READ FORM statement that causes a specific form record to be read:

```
READ FORM REMFILE-A USING FORMREC-A.
```

- The following example illustrates the use of the USING, INTO, and ON ERROR clauses:

```
WORKING-STORAGE SECTION.
Ø1 TEMPSAMPLEFORM   PIC X(24ØØ).
PROCEDURE DIVISION.
    .
    .
    .
      READ FORM REMFILE-A
           USING FORMLIB-A
           INTO TEMPSAMPLEFORM
           ON ERROR STOP RUN.
```

WRITE FORM Statement

The WRITE FORM statement writes the contents a form record, form record library, or text array to a specified remote file. The information in the following paragraphs shows the specific formats of the WRITE FORM statement that write the contents of each of these entities to a remote file. Format 3 of the WRITE FORM statement enables you to write user-defined error messages at the bottom of the previous screen.

WRITE FORM Statement for a Form Record Library (Format 1)

Format 1 of the WRITE FORM statement performs the self-identifying write operation of a form record library to a specific remote file. The self-identifying write operation enables the program to write a form record by placing the appropriate form record number value into the form record number field of the form record library. The form record number attribute specifies the form records before the self-identifying WRITE statement.

The execution of the WRITE FORM self-identifying WRITE statement releases a logical record to a remote file. The file must be open in the output, I/O, or extend mode at the time this statement is executed. For general information on opening files, refer to Volume 1.

Execution of a WRITE FORM statement does not affect the contents or accessibility of the record area. If a file is named in the SAME RECORD AREA clause, the logical record is also available as a record of other files referenced in that SAME RECORD AREA clause.

If you are not using the attribute form record number to place the form record number into the form record number field of the form record library before the self-identifying write operation, the self-identifying WRITE statement uses the existing value in the form record number field of the form record library.

The appropriate form record library performs the write operation. Any detected error condition is returned to the program when the WRITE FORM statement is performed.

See Also

- For more information on the form record number attribute, see "Using the Form Record Number Attribute" later in this section.
- For general information on error handling when using remote files, refer to the *SDF Plus Technical Overview*.

General Format (Format 1)

The general format of the WRITE FORM statement to perform a self-identifying write operation is as follows:

```
WRITE FORM {form-record-library-name-1                          }
           {record-name USING form-record-library-name-2        }
[; ON ERROR {imperative-statement-1}].
            {conditional-statement-1}
```

Explanation of Format Elements

form-record-library-name-1

This format element is used when the form record library is invoked from the file description (FD) of the DATA DIVISION.

USING

If a form record library is invoked in the WORKING-STORAGE SECTION, the USING clause enables the form record library to be written. If a form record library is invoked as a record area in the FD entry for a remote file, no USING clause is needed.

record-name and form-record-library-name-2

Record-name is the name of the logical record area in a specified remote file. When a WRITE FORM statement that contains a USING clause is executed, the data in form-record-library-name-2 is written directly to the device without affecting the logical record area of the file. The specification of the record area only identifies the file to which the data in form-record-library-2 is written.

ON ERROR

When the ON ERROR condition occurs, the actions that follow are the same as for an Invalid Key condition. Refer to Volume 1 for a discussion of an Invalid Key condition on a WRITE statement.

Considerations for Use

Two run-time errors specific to forms can be handled programmatically by using the FILE STATUS clause in the INPUT-OUTPUT SECTION. A form error message (82) is returned when either of the following occurs:

- A write operation is executed on a specific form and that form no longer resides in the form record library.
- The compile-time form record version does not equal the run-time form record version.

Examples

The following examples illustrate the use of the WRITE FORM statement for a self-identifying WRITE statement.

- In this example, the CHANGE statement places the form record number of FORMREC-1 into the form record number field of the FORMRECLIB form record library before the self-identifying WRITE FORM statement is performed.

```
CHANGE ATTRIBUTE FORMRECNUM OF FORMRECLIB TO
    ATTRIBUTE FORMRECNUM OF FORMREC-1.
WRITE FORM FORMRECLIB.
```

- In this example, the form record library FORMRECLIB2 is invoked in the WORKING-STORAGE SECTION, not the FILE SECTION. F-REC is the logical record area for a remote file.

```
WRITE FORM  F-REC USING FORMRECLIB2.
```

WRITE FORM Statement for a Form Record (Format 2)

Format 2 of the WRITE FORM statement writes the contents of a form record to a specified remote file. Format 2 does not perform a self-identifying write operation. The DEFAULT FORM option enables you to write a form with default values in each field.

The execution of the WRITE FORM statement releases a logical record to a remote file. The file must be open in the output, I/O, or extend mode at the time this statement is executed. For general information on opening files, refer to Volume 1.

Execution of a WRITE FORM statement does not affect the contents or accessibility of the record area. If a file is named in the SAME RECORD AREA clause, the logical record is also available as a record of other files referenced in that SAME RECORD AREA clause.

The current record pointer is unaffected by the execution of a WRITE FORM statement.

The appropriate form record library performs the write operation. Any detected error condition is returned to the program. For general information on error-handling when using remote files, refer to the *SDF Plus Technical Overview*.

General Format (Format 2)

The general format of the WRITE FORM statement to write the contents of a form record is as follows:

```
WRITE FORM {form-record-name-1                          }
           {record-name-1 USING form-record-name-2      }
[FROM {identifier-1    }]
      {DEFAULT FORM    }
[; ON ERROR {imperative-statement-1  }].
            {conditional-statement-1 }
```

Explanation of Format Elements

form-record-name-1

This format element is used when the form record of a form record library is declared in the FILE SECTION of the DATA DIVISION.

record-name-1

This format element is the name of the logical record in the FILE SECTION of the DATA DIVISION and can be qualified. Record-name-1 and identifier-1 must not reference the same storage area.

Record-name-1 with the USING clause enables the write operation to write the form records of a form record library that is invoked in the WORKING-STORAGE SECTION.

USING

If a form record library is invoked in the WORKING-STORAGE SECTION, the USING clause enables its form records to be written. If a form record library is invoked as a record area in the FD entry for a remote file, no USING clause is needed.

form-record-name-2

When the WRITE FORM statement that contains a USING clause is executed, the data in form-record-name-2 is written directly to the device without affecting the logical record area of the file. The specification of the record area only identifies the file to which the data in form-record-2 is written.

FROM

A WRITE FORM statement with the FROM phrase is equivalent to the statement *MOVE identifier-1 TO form-record-name-1* (according to MOVE statement rules)

followed by the WRITE FORM statement without the FROM phrase. For more information on the MOVE statement, refer to Volume 1.

identifier-1

This format element must be an 01-level record that is declared in the DATA DIVISION.

DEFAULT FORM

This option is an SDF Plus feature. The DEFAULT FORM option causes the form record library to write a form record with default values in each field.

ON ERROR

When the ON ERROR condition is recognized, the actions that follow are the same as for an Invalid Key condition. Refer to Volume 1 for a discussion of an Invalid Key condition on a WRITE statement.

Considerations for Use

Two run-time errors specific to forms can be handled programmatically by using the FILE STATUS clause in the INPUT-OUTPUT SECTION. A form error message (82) is returned when either of the following occurs:

- A write is executed on a specific form and that form no longer resides in the form record library.
- The compile-time form record version does not equal the run-time form record version.

Examples

The following examples show the use of the WRITE FORM statement to write the contents of a form record.

- In this example, the form record FORMREC-A is written. FORMREC-A is invoked from the FILE SECTION.

```
WRITE FORM  FORMREC-A.
```

- In this example, the WRITE FORM statement writes the form record FORMREC-B. FORMREC-B is invoked from the WORKING-STORAGE SECTION. The record name for the remote file is REMF-REC.

```
WRITE FORM  REMF-REC USING FORMREC-B.
```

- In this example, the contents of the identifier TEMP-FORM-A are moved to form record FORMREC-C before FORMREC-C is written.

```
WRITE FORM FORMREC-C USING FROM TEMP-FORM-A.
```

WRITE FORM Statement for Error Messages (Format 3)

Format 3 of the WRITE FORM statement enables you to reject the data received during the preceding READ FORM statement and to display an appropriate transaction error message to the user.

You define transaction errors using the SDF Plus Forms Editor, which assigns a unique transaction error number to each transaction error you define for the transaction. For each transaction error, you define an error message and a list of fields that are highlighted when the error occurs. The transaction error number you specify using format 3 identifies which of the transaction errors is to be processed by SDF Plus.

You can perform more than one WRITE FORM FOR ERROR MESSAGE statement in order to indicate multiple transaction errors.

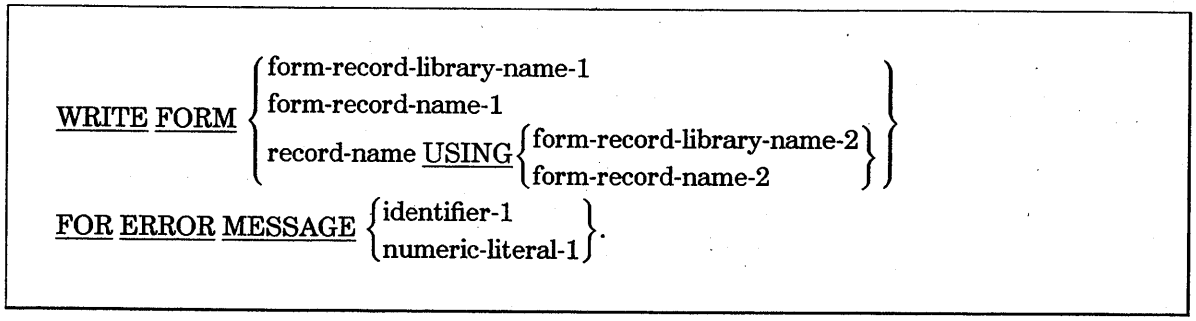
Each WRITE FORM FOR ERROR MESSAGE statement must be followed by an additional WRITE FORM statement, either format 1 or format 2, in order to initiate SDF Plus transaction error processing. At that point, SDF Plus redisplays the form that resulted in the transaction error, including the user's data, any highlights associated with the transaction error, and the transaction error message specified for the transaction error. The cursor is positioned in the first highlighted field.

Because the last WRITE FORM statement is used only to initiate transaction error processing, SDF Plus ignores any data received as the result of the last WRITE FORM statement (format 1 or format 2).

You use format 3 of the WRITE FORM statement only with update transactions; you cannot use it with prefill transactions.

General Format (Format 3)

The general format of the WRITE FORM FOR ERROR MESSAGE statement is as follows:



Explanation of Format Elements

form-record-library-names, form-record-names, record-name, and USING

For explanations of these format elements, refer to the explanations of format 1 and format 2 of the WRITE FORM statement.

FOR ERROR MESSAGE

This format element indicates that a transaction error has occurred and identifies which transaction error has occurred.

identifier-1

This format element must be a numeric data item. Identifier-1 contains the transaction error number corresponding to the user-predefined transaction error to be processed by SDF Plus.

numeric-literal-1

This format element must be an integer. It specifies the transaction error number corresponding to the user-predefined transaction error to be processed by SDF Plus.

Examples

The following examples show the use of the WRITE FORM FOR ERROR MESSAGE statement.

- In this example, the WRITE FORM statement indicates that transaction error number 5 has occurred:

```
WRITE FORM  SAMPLEFORMLIB FOR ERROR MESSAGE 5.
```

- In this example, the WRITE FORM statement indicates that the transaction error stored in identifier A1 has occurred:

```
WRITE FORM  SAMPLEFORMLIB FOR ERROR MESSAGE A1.
```

WRITE FORM Statement for Text Arrays (Format 4)

The WRITE FORM TEXT statement is an extended format of the WRITE FORM statement that causes the contents of text arrays to be written to a designated remote file by using a form record library.

The WRITE FORM TEXT statement must be used with the standard WRITE FORM statement. A standard WRITE FORM statement immediately after a WRITE FORM TEXT statement causes the contents of the text array to be written at the bottom of the standard form.

A WRITE FORM TEXT statement has no effect unless the next I/O related to SDF Plus that is performed on the remote file is a WRITE FORM statement.

General Format (Format 4)

The general format of the WRITE FORM TEXT statement is as follows:

```
WRITE FORM { form-record-library-name-1 }
           { form-record-name-1         } USING TEXT data-name-1
           { record-name-1              }
[FOR { integer-1    } CHARACTERS].
     { identifier-1 }
```

Explanation of Format Elements

form-record-library-name-1 and form-record-name-1

Form-record-library-name-1 is the name of a form record library. Form-record-name-1 is the name of a form record. Both these format elements are invoked in the FD entry of the DATA DIVISION.

If a form record library is invoked within an FD entry, the WRITE FORM TEXT statement is performed by using either form-record-name-1 or form-record-library-name-1, followed by the USING TEXT clause.

record-name-1

This format element is the logical record area for a remote file that is used in the SDF Plus remote file interface. If a form record library is invoked in the WORKING-STORAGE SECTION, record-name-1 followed by the USING TEXT clause writes the text.

USING TEXT

This clause writes text to the end of the previous write form.

data-name-1

This format element specifies the name of the text array. Data-name-1 must be at the 01-level. Data-name-1 is moved to record-name-1 before the write occurs.

FOR integer-1 CHARACTERS

This format element identifies the number of characters of the text array to be written.

FOR identifier-1 CHARACTERS

This format element identifies the numeric identifier used to specify the number of characters that are to be written.

Examples

The following examples illustrate the use of the WRITE FORM TEXT statement.

- In the following example, CHAR-COUNT is the preassigned number of characters to be written. The FORMRECLIB form record library is invoked in the FD section. Format 1 or Format 2 of the WRITE FORM statement must follow this code to provide a meaningful context for the example.

```
WRITE FORM FORMRECLIB USING TEXT S-TEXT
        FOR CHAR-COUNT CHARACTERS.
```

- In this example, the first 100 characters of the text array TEXT-A are written. The form record library is invoked from the WORKING-STORAGE SECTION; therefore, the FILE-FD-REC logical record area for the remote file writes the text. Format 1 or Format 2 of the WRITE FORM statement must follow this code to provide a meaningful context for the example.

```
WRITE FORM FILE-FD-REC USING TEXT TEXT-A
       FOR 100 CHARACTERS.
```

- In this example, FORMRECLIB-2 is invoked under the FD section. The TEXT-B text array is written to the bottom of the FORMREC-1 form record for FORMRECLIB-2.

```
WRITE FORM FORMREC-1 USING TEXT TEXT-B.
WRITE FORM FORMREC-1.
```

- In this example, the form record library FORMRECLIB-3 is invoked under the FD section. Text TEXT-C is written to the bottom of FORMRECLIB-3. A self-identifying WRITE FORM statement follows the WRITE FORM TEXT statement.

```
WRITE FORM FORMREC-3 USING TEXT TEXT-C.
WRITE FORM FORMRECLIB-3.
```

Using the Form Record Number Attribute

The form record number attribute is used with either individual form records or form record libraries. In some manuals, the term *message type number* is used as a synonym for *form record number*.

The form record number attribute associated with individual form records is preassigned by SDF Plus at compile time.

The form record number attribute associated with form record libraries is used to perform self-identifying read operations and self-identifying write operations at run time.

General Format

The general format of the form record number attribute is as follows:

```
ATTRIBUTE FORMRECNUM { OF form-record-library-name-1 }
                     { OF form-record-name-1         }
```

Explanation of Format Elements

form-record-library-name-1

When the form record number attribute is used with form record libraries, the attribute accesses the form record number of a particular form record that is returned on a self-identifying read operation. This is necessary when you do not know the form record that has been read, as is the case with self-identifying read operations.

Before a self-identifying write operation can be completed, you must move the form record number attribute of the form record to the form record number attribute of the form record library.

The value of the form record number attribute might change for each self-identifying I/O operation (read or write). The returned value is valid immediately after a self-identifying I/O operation is performed and continues to be valid until the next self-identifying I/O operation is performed.

form-record-name-1

When the form record number attribute is used with individual form records, the attribute enables the compiler to return the form record number integer of the form record name for programmatic purposes.

Examples

The following examples illustrate the use of the form record number attribute.

- In the following example, NUM-1 is assigned the unique value of the form record number of form record FORMREC-A:

```
MOVE ATTRIBUTE FORMRECNUM OF FORMREC-A TO NUM-1.
```

- In this example, the form record number of FORMLIB-A is assigned the form record number of FORMREC-A:

```
CHANGE ATTRIBUTE FORMRECNUM OF FORMLIB-A
    TO ATTRIBUTE FORMRECNUM OF FORMREC-A.
```

- The following example illustrates the use of a check to determine whether the form record number of form record library FORM-LIB is equal to the form record number of FORMREC-B after a self-identifying read operation:

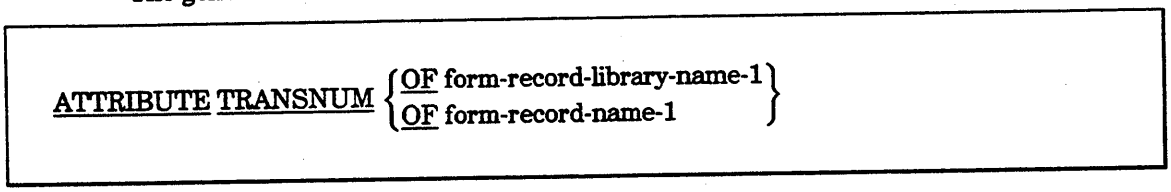
```
IF ATTRIBUTE FORMRECNUM OF FORMREC-B
      = ATTRIBUTE FORMRECNUM OF FORM-LIB THEN
   PERFORM PROCESS-FORM-B.
```

Using the Transaction Number Attribute

The transaction number attribute is used with either individual transactions or form record libraries. There is a pair of form records under each transaction to either output data from a form or to input data to a form. The transaction number attribute provides information for the form record that is under the current transaction.

General Format

The general format of the transaction number attribute is as follows:



Explanation of Format Elements

form-record-library-name-1

This format element identifies the name of the form record library. When used with form record libraries, the transaction number attribute accesses the transaction number of a particular form record that is returned on a self-identifying read operation. You can retrieve the returned value from the transaction number attribute that indicates the transaction that was performed on a self-identifying read operation.

The value of a transaction number attribute might change for each self-identifying read operation. The returned transaction number attribute value is valid immediately after the self-identifying read operation is performed and continues to be valid until the next self-identifying read operation is executed.

form-record-name-1

This format element identifies the form record name. When used with individual form records, the transaction number attribute returns the preassigned transaction number of a designated transaction.

Examples

The following examples illustrate the use of the transaction number attribute.

- In this example, NUM-1 is assigned the transaction number that was last read using the self-identifying syntax:

```
MOVE ATTRIBUTE TRANSNUM OF FORMREC-1 TO NUM-1.
```

- In the following example, a check is completed to determine whether the transaction number attribute of form record library FORM-LIB is equal to the transaction number attribute of the form record FORMREC-1 after a self-identifying read:

```
IF ATTRIBUTE TRANSNUM OF FORMREC-1TT
      =ATTRIBUTE TRANSNUM OF FORM-LIB THEN
   CHANGE ATTRIBUTE FORMRECNUM OF FORM-LIB
      TO ATTRIBUTE FORMRECNUM OF FORMREC-1.
```

Using SDF Plus with COMS

SDF Plus can be used with COMS to take advantage of COMS direct windows. This feature gives you enhanced routing capabilities for form records and also enables preprocessing and postprocessing of form records. Example 7–2 at the end of this section shows an application program interacting with users through a COMS window.

Refer to the *COMS Programming Guide* for detailed information on the use of the COMS direct-window interface. The following guidelines explain the steps to follow when using SDF Plus with COMS.

Using COMS Input/Output Headers

SDF Plus supports the use of COMS headers. Three fields are defined within the headers for use with SDF Plus. These fields are SDFINFO, SDFFORMRECNUM, and SDFTRANSNUM. A description of each follows.

The SDFINFO field is used to identify specific form message processing requests (on output) or to return form message processing errors (on input).

On the output (sending) path, the SDFINFO field can contain the following values:

Value	Explanation
0	Normal form message processing
100	Last transaction error
101	More than one transaction error
200	Text message processing

On the input (receiving) path, the SDFINFO field can contain the following values, which indicate the status of the requested form message processing:

Value	Explanation
0	No error
–100	Form message timestamp mismatch
–200	Incorrect form record number specified from a send operation
–300	Incorrect transaction number specified from a send operation
–400	Invalid message key

The SDFFORMRECNUM field is used to specify the form record to be written (on output) or the form record that was received (on input).

The SDFTRANSNUM field is meaningful only on input and contains the number of the SDF Plus transaction that was received. This field should not be altered by the user application.

Sending and Receiving Messages

When using SDF Plus and COMS together, follow the usual statements for each product, with the following guidelines:

- Unisys recommends open notification for the window. Using open notification ensures that the correct initialization is sent to the program. The program can then be written to display the correct initial form without user input. If the initialization test has a transaction code (trancode), the trancode must be created under the default input agenda.
- Message sending requires the application program to first move the value 0 (zero) into the SDFINFO field of the output header. The application program must also move the form record number of the form record library into the SDFFORMRECNUM field. The buffer of the form record library must be passed as the message area in the SEND statement.
- Message receiving has the following significance for the application program:
  - If the SDFINFO field contains the value 0 (zero), then the application program can query the form record number and transaction number attributes for the form record library from the SDFFORMRECNUM and SDFTRANSNUM fields of the input header.
  - If the SDFINFO field contains a value less than 0 (zero), then this field contains an error code that indicates a problem with message processing. In addition, the FUNCTION-INDEX field of the input header contains the value 100.
- SEND or RECEIVE statements cannot be used between BEGIN-TRANSACTION and END-TRANSACTION statements. Doing so violates the rules of synchronized recovery. COMS will not allow you to send a request to a station for data correction while SDF Plus is in transaction mode.

Sending Transaction Errors

SDF Plus supports the ability to send error codes in response to incorrect data received by the user application. These error codes are sent as integer values, which SDF Plus uses to process a user-defined error procedure for the form record library.

To send transaction errors, the user application must do the following:

- Move the value 100 into the SDFINFO field of the output header.
- Move the value of the transaction error into the SDFFORMRECNUM field of the output header.
- Move the SDFTRANSNUM field from the input header to the output header.
- Send the output header to display the message.

The user program can send any arbitrary message area along with the output header. SDF Plus processes only the information within the output header.

Example

The following example shows code in which transaction error number 1 is sent. SDF-BUFFER is the user-defined buffer area.

```
MOVE Ø TO TEXTLENGTH OF COMS-OUT.
MOVE 1ØØ TO SDFINFO OF COMS-OUT.
MOVE 1 TO SDFFORMRECNUM OF COMS-OUT.
MOVE SDFTRANSNUM OF COMS-IN TO
   SDFTRANSNUM OF COMS-OUT.
SEND COMS-OUT FROM SDF-BUFFER.
```

Sending Text Messages

SDF Plus supports the ability to send text messages for display on the text area of a form.

To send a text message, the application program must do the following:

- Move the value 200 into the SDFINFO field of the output header.
- Move the text message into a message area to be sent through COMS.
- Use the SEND statement to store the text message.
- Move the value 0 (zero) to the SDFINFO field of the output header.
- Send the form to display the text message.

See Also

For information about the extensions used with COMS, refer to Section 3, "Using the COMS Program Interface."

Example

The following example shows code in which literal text is moved into the message area. The form to display the text message is FORM1. The first MOVE statement moves the length of the message to COMS_OUT.

```
MOVE 38 TO TEXTLENGTH OF COMS_OUT
MOVE 2ØØ TO SDFINFO OF COMS-OUT
MOVE "This is an example of application text" TO SDF-BUFFER
SEND COMS-OUT FROM SDF-BUFFER
MOVE Ø TO SDFINFO OF COMS-OUT
MOVE ATTRIBUTE FORMRECNUM OF FORM1 TO
   SDFFORMRECNUM OF COMS-OUT
SEND COMS-OUT FROM FORM1
```

SDF Plus Sample Programs

The following are two sample programs showing the SDF Plus program interface used with a remote file interface and with a COMS interface. Both programs are in the transaction mode. For information about handling remote file errors in an application program, refer to the *SDF Plus Technical Overview*.

SDF Plus Program with a Remote File Interface

The sample program in Example 7–1 begins by performing a READ FORM statement. The program then examines the transaction number attribute to determine the form record that was read. The program indicates the appropriate response by setting the form record number attribute.

The program accepts two string or binary inputs from a remote file, concatenates or adds them together, and returns the original input and the results as output on the terminal screen. The form record library was created in SDF Plus.

```
001000 IDENTIFICATION DIVISION.
007000
008000 ENVIRONMENT DIVISION.
009000 CONFIGURATION SECTION.
011100
011200 SPECIAL-NAMES.
011300   DICTIONARY IS "SDFPLUSDICT".
011500*
012000 INPUT-OUTPUT SECTION.
013000 FILE-CONTROL.
014000     SELECT REMFILE ASSIGN TO REMOTE
014500     STATUS IS REMFILE-STATUS
014750     ACTUAL KEY IS REMFILE-RSN.
015000*
016000 DATA DIVISION.
017000 FILE SECTION.
018000 FD REMFILE
018050    RECORD CONTAINS 2400 CHARACTERS
018060    VALUE OF BLOCKSTRUCTURE IS EXTERNAL,
019070             MAXRECSIZE IS 2400.
021200 01 DATATYPE6B FROM DICTIONARY
021250              DIRECTORY IS "SMITH".
022500*--DICTIONARY DIRECTORY : SMITH                    D
022550*--DICTIONARY FORMLIST < DATATYPE6B >.             D
022600*--SDF TRANSACTION ( APUTALPHASTT ).               D
022650 01 APUTALPHAS.                                   D
022700       04 PASTRING1  PIC X(25).                   D
022900       04 PASTRING2  PIC X(25).                   D
022950 01 DATATYPE6BSR.                                 D
023000       04 DATATYPE6BSRF PIC X(1).                 D
023150*--SDF TRANSACTION ( AGETALPHASPTT ).              D
023200 01 AGETALPHASPRE.                                D
```

Example 7–1. SDF Plus Program with a Remote File Interface

```
023250       04 GASTRING1  PIC X(25).                                  D
023300       04 GASTRING2  PIC X(25).                                  D
023350       04 GASTRING   PIC X(50).                                  D
023400*01 AGETALPHASPRE.                                               D
023500*--SDF TRANSACTION (AGETALPHASTT ).                              D
023550 01 AGETALPHAS.                                                  D
023600       04 GASTRING1 PIC X(25).                                   D
023650       04 GASTRING2 PIC X(25).                                   D
023700       04 GASTRING  PIC X(50).                                   D
023750*01 DATATYPE6BSR.                                                D
023800*--SDF TRANSACTION ( APUTBINARYTT ).                             D
023850 01 APUTBINARY.                                                  D
023900       04 PBNUMBER1 PIC 9(3) BINARY.                             D
023950       04 PBNUMBER2 PIC 9(3) BINARY.                             D
024000*01 DATATYPE6BSR.                                                D
024025*--SDF TRANSACTION ( AGETBINARYPTT ).                            D
024050 01 AGETBINARYPRE.                                               D
024075       04 GBNUMBER1 PIC 9(3) BINARY.                             D
024100       04 GBNUMBER2 PIC 9(3) BINARY.                             D
024125       04 GBNUMBER  PIC 9(4) BINARY.                             D
024150*01 AGETBINARYPRE.                                               D
024175*--SDF TRANSACTION (AGETBINARYTT ).                              D
024200 01 AGETBINARY.                                                  D
024225       04 GBNUMBER1 PIC 9(3) BINARY.                             D
024250       04 GBNUMBER2 PIC 9(3) BINARY.                             D
024500       04 GBNUMBER  PIC 9(4) BINARY.                             D
024510*01 DATATYPE6BSR.                                                D
024530*01 APUTALPHAS.                                                  D
024540*01 AGETALPHAS.                                                  D
024545*01 APUTBINARY.                                                  D
024547*01 AGETBINARY.                                                  D
024549*
024550 01 OUT-LINE          PIC X(72).
024600*
024650 WORKING-STORAGE SECTION.
024660*
024670 01 REMFILE-STATUS    PIC X(02).
024680 01 REMFILE-RSN       PIC 9(04).
024700 01  CONSTANTS.
024750    02 TRUE-FLAG      PIC 9 COMP VALUE 1.
024800    02 FALSE-FLAG     PIC 9 COMP VALUE 0.
024850
024900 01  END-PGMV          PIC 9 COMP VALUE 0.
024950 01  TRANSTYPE         PIC 9(8)   VALUE 0.
025000 01  MYSTRING.
025100     03 MYSTRING1        PIC X(25)  VALUE SPACES.
025150     03 MYSTRING2        PIC X(25)  VALUE SPACES.
025200 01  MYBNUMBER         PIC 9(4) BINARY VALUE ZERO.
025250 01  MYBNUMBER1        PIC 9(3) BINARY VALUE ZERO.
025300 01  MYBNUMBER2        PIC 9(3) BINARY VALUE ZERO.
025350
```

Example 7–1. SDF Plus Program with a Remote File Interface (cont.)

```
025375*
025400 PROCEDURE DIVISION.
025450 MAIN-PARA.
025475***
025500     OPEN   I-O  REMFILE.
025700     PERFORM MAIN-FORM THRU MAIN-FORM-EXIT
025750       UNTIL END-PGMV = TRUE-FLAG.
025800     CLOSE REMFILE.
025850     STOP RUN.
025875***
025900 MAIN-FORM.
025950     READ FORM REMFILE USING DATATYPE6B
026000         ON ERROR DISPLAY "READ FORM ERROR"
026100                  MOVE TRUE-FLAG TO END-PGMV
026150                  GO TO MAIN-FORM-EXIT.
026200     IF ATTRIBUTE TRANSNUM OF DATATYPE6B =
026250        ATTRIBUTE TRANSNUM OF AGETALPHASPTT THEN
026300         CHANGE ATTRIBUTE FORMRECNUM OF DATATYPE6B TO
026350          ATTRIBUTE FORMRECNUM OF AGETALPHASPRE
026400         PERFORM GETALPHAS
026450     ELSE IF ATTRIBUTE TRANSNUM OF DATATYPE6B =
026500        ATTRIBUTE TRANSNUM OF AGETBINARYPTT THEN
026550         CHANGE ATTRIBUTE FORMRECNUM OF DATATYPE6B TO
026600          ATTRIBUTE FORMRECNUM OF AGETBINARYPRE
026650         PERFORM GETBINARY
026700     ELSE IF ATTRIBUTE TRANSNUM OF DATATYPE6B =
026750        ATTRIBUTE TRANSNUM OF APUTALPHASTT THEN
026800         CHANGE ATTRIBUTE FORMRECNUM OF DATATYPE6B TO
026850          ATTRIBUTE FORMRECNUM OF DATATYPE6BSR
026900         PERFORM CONCATSTRINGS
026950     ELSE IF ATTRIBUTE TRANSNUM OF DATATYPE6B =
027000        ATTRIBUTE TRANSNUM OF APUTBINARYTT THEN
027100         CHANGE ATTRIBUTE FORMRECNUM OF DATATYPE6B TO
027150          ATTRIBUTE FORMRECNUM OF DATATYPE6BSR
027200         PERFORM BINARYADD
027205     ELSE IF ATTRIBUTE TRANSNUM OF DATATYPE6B =
027210        ATTRIBUTE TRANSNUM OF AGETALPHASTT OR
027220             ATTRIBUTE TRANSNUM OF DATATYPE6B =
027230        ATTRIBUTE TRANSNUM OF AGETBINARYTT THEN
027240         CHANGE ATTRIBUTE FORMRECNUM OF DATATYPE6B TO
027245          ATTRIBUTE FORMRECNUM OF DATATYPE6BSR
027250     ELSE MOVE TRUE-FLAG TO END-PGMV.
027275*
027300     WRITE FORM DATATYPE6B ON ERROR
027350         DISPLAY "WRITE FORM ERROR"
027400         MOVE TRUE-FLAG TO END-PGMV.
027425 MAIN-FORM-EXIT.
027430         EXIT.
027450***
027500 CONCATSTRINGS.
027550     MOVE PASTRING1 TO MYSTRING1.
```

Example 7–1. SDF Plus Program with a Remote File Interface (cont.)

```
027600      MOVE PASTRING2 TO MYSTRING2.
027675      STRING MYSTRING1, MYSTRING2, DELIMITED BY SIZE INTO MYSTRING
027680          ON OVERFLOW DISPLAY "STRING ERROR ", MYSTRING.
027690*
028000 BINARYADD.
028100      MOVE PBNUMBER1 TO MYBNUMBER1.
028150      MOVE PBNUMBER2 TO MYBNUMBER2.
028200      COMPUTE MYBNUMBER = MYBNUMBER1 + MYBNUMBER2.
028250
028300 GETALPHAS.
028350      MOVE MYSTRING1 TO GASTRING1 IN AGETALPHASPRE.
028400      MOVE MYSTRING2 TO GASTRING2 IN AGETALPHASPRE.
028450      MOVE MYSTRING TO GASTRING IN AGETALPHASPRE.
028500
028550 GETBINARY.
028600      MOVE MYBNUMBER1 TO GBNUMBER1 IN AGETBINARYPRE.
028650      MOVE MYBNUMBER2 TO GBNUMBER2 IN AGETBINARYPRE.
028700      MOVE MYBNUMBER TO GBNUMBER IN AGETBINARYPRE.
028750*
028800 LAST-PROC.
028850*
028900      EXIT.
```

Example 7–1. SDF Plus Program with a Remote File Interface (cont.)

SDF Plus Program with a COMS Interface

Example 7–2 shows the same programming logic as Example 7–1; however, the following COMS interface example shows the application program interacting with users through a COMS window. The SDFTRANSNUM field, which is imported from the COMS input header, is interrogated to determine the form record that was read. The program indicates its response by setting the SDFFORMRECNUM field. This field was imported from the COMS output header. Additionally, the program accepts two string or binary inputs from COMS into a message area declared in the DATA DIVISION.

```
001800 IDENTIFICATION DIVISION.
001900 ENVIRONMENT    DIVISION.
002000 CONFIGURATION  SECTION.
002100 SPECIAL-NAMES.
002400      DICTIONARY IS "SDFPLUSDICT".
002500 DATA          DIVISION.
002600 WORKING-STORAGE SECTION.
002700 01 SDFMSG         PIC X(2400).
003000 01 COMS-NAME      PIC X(72).
003950*************************************
004400* This is the form record library. *
004550*************************************
004500 01 DATATYPE6B FROM DICTIONARY.                    D
004700              DIRECTORY IS "SMITH".               D
000100*--DICTIONARY DIRECTORY : SMITH.                  D
```

Example 7–2. Using SDF Plus with a COMS Interface

```
000110*--DICTIONARY FORMLIST< DATATYPE6B >.                    D
000120*--SDF TRANSACTION( APUTALPHASTT ).                       D
000140 01 APUTALPHAS.                                          D
000150       04 PASTRING1 PIC X(25).                           D
000160       04 PASTRING2 PIC X(25).                           D
000180 01 DATATYPE6BSR.                                        D
000190       04 DATATYPE6BSRF PIC X(1).                        D
000200*--SDF TRANSACTION( AGETALPHASPTT ).                      D
000220 01 AGETALPHASPRE.                                       D
000230       04 GASTRING1 PIC X(25).                           D
000240       04 GASTRING2 PIC X(25).                           D
000250       04 GASTRING PIC  X(50).                           D
000270*01 AGETALPHASPRE.                                       D
000280*--SDF TRANSACTION( AGETALPHASTT ).                       D
000300 01 AGETALPHAS.                                          D
000310       04 GASTRING1  PIC X(25).                          D
000320       04 GASTRING2  PIC X(25).                          D
000330       04 GASTRING   PIC X(50).                          D
000350*01 DATATYPE6BSR.                                        D
000590*--SDF TRANSACTION( APUTBINARYTT ).                       D
000610 01 APUTBINARY.                                          D
000620       04 PBNUMBER1 PIC 9(3) BINARY.                     D
000630       04 PBNUMBER2 PIC 9(3) BINARY.                     D
000650*01 DATATYPE6BSR.                                        D
000660*--SDF TRANSACTION( AGETBINARYPTT ).                      D
000680 01 AGETBINARYPRE.                                       D
000690       04 GBNUMBER1 PIC 9(3) BINARY.                     D
000700       04 GBNUMBER2 PIC 9(3) BINARY.                     D
000710       04 GBNUMBER  PIC 9(4) BINARY.                     D
000730*01 AGETBINARYPRE.                                       D
000740*--SDF TRANSACTION( AGETBINARYTT ).                       D
000760 01 AGETBINARY.                                          D
000770       04 GBNUMBER1 PIC 9(3) BINARY.                     D
000780       04 GBNUMBER2 PIC 9(3) BINARY.                     D
000790       04 GBNUMBER  PIC 9(4) BINARY.                     D
000810*01 DATATYPE6BSR.                                        D
001540*01 APUTALPHAS.                                          D
001560*01 AGETALPHAS.                                          D
001620*01 APUTBINARY.                                          D
001640*01 AGETBINARY.                                          D
001770*
004800 01 SDF-AGENDA-NAME      PIC X(17) VALUE "JONES".
004900 77 SDF-AGENDA-DESIGNATOR   USAGE REAL.
005000 77 SDF-CALL-ERROR       PIC S9(11) USAGE BINARY.
005200 01 CONSTANTS.
005300    02 TRUE-FLAG         PIC 9 COMP VALUE 1.
005400    02 FALSE-FLAG        PIC 9 COMP VALUE 0.
005500
005600 01 END-PGMF             PIC 9 COMP VALUE 0.
005800 01 MYSTRING.
005900    03 MYSTRING1         PIC X(25) VALUE SPACES.
```

Example 7–2. Using SDF Plus with a COMS Interface (cont.)

```
006000    03 MYSTRING2          PIC X(25) VALUE SPACES.
006400 01 MYBNUMBER             PIC 9(4) BINARY VALUE ZERO.
006500 01 MYBNUMBER 1           PIC 9(3) BINARY VALUE ZERO.
006600 01 MYBNUMBER 2           PIC 9(3) BINARY VALUE ZERO.
009400*
009500* In both the input header and output header, SDFINFO,
009600* SDFFORMRECNUM, and SDFTRANSNUM are imported.
009700* These fields must be set correctly.
009900*
010000 COMMUNICATION SECTION.
010100 INPUT HEADER INHDR.
010300*
010400 OUTPUT HEADER OUTHDR
010500     CONVERSATION AREA.
010600     01 OUT-CA PIC REAL.
010700 PROCEDURE DIVISION.
010800     MAIN-PARA.
011000     PERFORM START-UP.
011100     PERFORM MAIN-FORM THRU MAIN-FORM-EXIT.
011200        UNTIL STATUSVALUE OF INHDR = 99.
011300     STOP RUN.
011500 START-UP.
011600     MOVE ATTRIBUTE NAME OF ATTRIBUTE EXCEPTIONTASK OF
011700         ATTRIBUTE EXCEPTIONTASK OF MYSELF TO COMS-NAME.
011800     CHANGE ATTRIBUTE TITLE OF "DCILIBRARY" TO COMS-NAME.
011900     ENABLE INPUT INHDR KEY "ONLINE".
012000     CALL "GET_DESIGNATOR_USING_NAME IN DCILIBRARY"
012100         USING SDF-AGENDA-NAME
001220                , VALUE AGENDA
012300                , SDF-AGENDA-DESIGNATOR
012400         GIVING SDF-CALL-ERROR.
012440
012600 MAIN-FORM.
012700     RECEIVE INHDR  MESSAGE INTO SDFMSG.
012800     IF STATUSVALUE OF INHDR NOT = 99
012900     IF NOT FUNCTIONSTATUS OF INHDR < 0 THEN
013000         MOVE 1                    TO DESTCOUNT OF OUTHDR
013100         MOVE STATION OF INHDR     TO DESTINATIONDESG OF OUTHDR.
013700     IF SDFTRANSNUM OF INHDR =
013800        ATTRIBUTE TRANSNUM OF AGETALPHASPTT THEN
013900         MOVE ATTRIBUTE FORMRECNUM OF AGETALPHASPRE TO
014000          SDFFORMRECNUM OF OUTHDR
014100         PERFORM GETALPHAS
014150         MOVE AGETALPHASPRE TO SDFMSG
014700     ELSE IF SDFTRANSNUM OF INHDR =
014800        ATTRIBUTE TRANSNUM OF AGETBINARYPTT THEN
014900         MOVE ATTRIBUTE FORMRECNUM OF AGETBINARYPRE TO
015000          SDFFORMRECNUM OF OUTHDR
015100         PERFORM GETBINARY
015150         MOVE AGETBINARYPRE TO SDFMSG
019700     ELSE IF SDFTRANSNUM OF INHDR =
```

Example 7–2. Using SDF Plus with a COMS Interface (cont.)

```
019800         ATTRIBUTE TRANSNUM OF APUTALPHASTT THEN
019900          MOVE ATTRIBUTE FORMRECNUM OF DATATYPE6BSR TO
020000           SDFFORMRECNUM OF OUTHDR
020050          MOVE SDFMSG TO APUTALPHAS
020100          PERFORM CONCATSTRINGS
020700      ELSE IF SDFTRANSNUM OF INHDR =
020800         ATTRIBUTE TRANSNUM OF APUTBINARYTT THEN
020900          MOVE ATTRIBUTE FORMRECNUM OF DATATYPE6BSR TO
021000           SDFFORMRECNUM OF OUTHDR
021050          MOVE SDFMSG TO APUTBINARY
021100          PERFORM BINARYADD
023700      ELSE IF SDFTRANSNUM OF INHDR =
023800         ATTRIBUTE TRANSNUM OF AGETALPHASTT OR
024100              SDFTRANSNUM OF INHDR =
024200         ATTRIBUTE TRANSNUM OF AGETBINARYTT THEN
024900          MOVE ATTRIBUTE FORMRECNUM OF DATATYPE6BSR TO
025000             SDFFORMRECNUM OF OUTHDR
025700      ELSE MOVE TRUE-FLAG TO END-PGMV.
025800
025850      MOVE 0 TO SDFINFO OF OUTHDR.
026300      MOVE SDF-AGENDA-DESIGNATOR TO AGENDA OF OUTHDR.
026350      MOVE 2400 TO TEXTLENGTH OF OUTHDR
026400      SEND OUTHDR FROM SDFMSG.
026425 MAIN-FORM-EXIT.
02650       EXIT.
026500
026800***
026900 CONCATSTRINGS.
027000      MOVE PASTRING1 TO MYSTRING1.
027100      MOVE PASTRING2 TO MYSTRING2.
027400
028000 BINARYADD.
028100      MOVE PBNUMBER1 TO MYBNUMBER1.
028200      MOVE PBNUMBER2 TO MYBNUMBER2.
028300      COMPUTE MYBNUMBER = MYBNUMBER1 + MYBNUMBER2.
028400
032900 GETALPHAS.
033000      MOVE MYSTRING1 TO GASTRING1 OF AGETALPHASPRE.
033100      MOVE MYSTRING2 TO GASTRING2 OF AGETALPHASPRE.
033200      MOVE MYSTRING TO GASTRING OF AGETALPHASPRE.
033300
033900 GETBINARY.
034000      MOVE MYBNUMBER1 TO GBNUMBER1 OF AGETBINARYPRE.
034100      MOVE MYBNUMBER2 TO GBNUMBER2 OF AGETBINARYPRE.
034200      MOVE MYBNUMBER TO GBNUMBER OF AGETBINARYPRE.
034300
038900
038900
039000 END-OF-JOB.
```

Example 7–2. Using SDF Plus with a COMS Interface (cont.)

Section 8
Using the SIM Program Interface

The Semantic Information Manager (SIM) is a database system that provides for the control, retrieval, and maintenance of data. SIM is a member of the InfoExec family of products.

The compiler translates statements into symbolic SIM queries and generates calls to SIM at compile time and run time to process the queries. The program interface for SIM consists of Unisys extensions to COBOL74 that perform the following functions:

- Use the RESERVE clause to recognize SIM keywords as reserved words in a program.
- Declare a database.
- Map SIM types into COBOL74.
- Qualify attributes in single-perspective and multiple-perspective queries.
- Declare query variables and records used in a query.
- Open and close a database.
- Use statements for transaction states and transaction points.
- Declare an entity reference variable to explicitly hold a reference to a database entity.
- Use functions and selection expressions to manipulate data and determine entities or values within database statements.
- Select a set of entities and associate it with the query.
- Retrieve entities to make them available to the program.
- Deactivate a query using the DISCARD statement.
- Alter level values in retrieval that involve transitive closure.
- Update entities with single-statement or multiple-statement update statements, assign database attributes, and delete entities.
- Process SIM exceptions.

In an InfoExec environment, the Advanced Data Dictionary System (ADDS) is used to define Data Management System II (DMSII) or SIM databases. For information about the extensions available with the ADDS program interface, refer to Section 2, "Using the ADDS Program Interface." Some optional features available to SIM through ADDS include the following:

- Declaring a dictionary using the DICTIONARY statement when the location of the SIM database is in a data dictionary other than the ADDS default DATADICTIONARY.
- Specifying program tracking in the DICTIONARY statement. Program tracking provides information about databases as components invoked in a program. If you do not specify program tracking, it is not used.

When opening a SIM database, if you declared a dictionary in the DICTIONARY statement, the dictionary is used to locate the database. If you did not declare a dictionary in the DICTIONARY statement, or if the database was not found in the declared dictionary, the standard A Series file searching conventions are used to locate the database.

If a dictionary is not declared in the DICTIONARY statement, then the REPOSITORY option setting of the release specification in the InfoExec Configuration file determines if a dictionary is declared by default. If the REPOSITORY option is set to REQUIRED, then the ADDS default DATADICTIONARY is used. If the REPOSITORY option is set to OPTIONAL, then no dictionary is used.

For further information about the REPOSITORY option of the release specifications in the InfoExec Configuration file, refer to the *A Series Data Management Software Installation Guide.*

Note: *A program can invoke only one dictionary. Therefore, if a program accesses both a SIM database (from a dictionary) and SDF forms, both must be in the same dictionary.*

The information in this section explains how to use COBOL74 to manipulate data in a SIM database and provides samples of typical applications used with SIM.

For an alphabetical list of the extensions used with SIM, refer to "SIM Extensions" in Section 1, "Introduction to COBOL74 Program Interfaces."

Refer to the *A Series InfoExec Semantic Information Manager (SIM) Programming Guide* for the concepts and programming considerations for using SIM. For information on defining COBOL74 files and elements and for information about program tracking in SIM, refer to the *InfoExec ADDS Operations Guide.*

An overview of the concepts underlying SIM is provided in the *A Series InfoExec Semantic Information Manager (SIM) Technical Overview.*

Using the RESERVE Option

The RESERVE option notifies the compiler that SIM keywords are recognized as reserved words for the extent of the program. If the RESERVE option is not included for a particular class of keywords, the keywords in that class can be used as normal identifiers in the program. If the functionality of the class is needed, the RESERVE option must be included for that class, and the keywords are then interpreted as reserved words.

The RESERVE SEMANTIC option must be specified if databases with the clause VALUE OF DBKIND IS SEMANTIC are declared in the program.

If you use COBOL74 reserved words in a SIM database, the program will not compile. If you must use a COBOL74 reserved word, declare it with an internal name and reference the reserved word as a literal.

The RESERVE option appears in the SPECIAL-NAMES paragraph of the CONFIGURATION SECTION of the ENVIRONMENT DIVISION.

See Also

For information on the database declaration, refer to "Declaring a Database in SIM," later in this section.

General Format

The general format of the RESERVE option is as follows:

```
SPECIAL-NAMES.
[RESERVE WORDS LIST IS {SEMANTIC}...CAPABLE].
       {NETWORK }
```

Explanation of Format Elements

SEMANTIC

The following words are reserved only in programs that specify RESERVE SEMANTIC in the SPECIAL-NAMES paragraph:

APPLY	DMMOREEXCEPTIONS	EXCLUSIVE
CALLED	DMRESULT	EXISTS
CAT	DMSTATE	INCLUDE
DISCARD	DMSTRUCTURENAME	INVERSE
DISTINCT	DMSTRUCTURENUM	ISA
DIV	DMSUBCATEGORY	LEVEL
DMCATEGORY	DMSUBEXCEPTION	MOD
DMDBNAME	DMUPDATECOUNT	ORDER
DMERRORFLAG	DMVERIFYNAME	QD
DMEXCEPTION	ENTITYREFERENCE	RETRIEVE
DMEXCEPTIONINFO	EQUIV	SOME
DMFUNCTION	EXCLUDE	TRANSITIVE
DMLUCNAME	EXCLUDES	

NETWORK

The reserved words for programs that specify the NETWORK option are explained in Volume 1.

Example

The following example uses the RESERVE SEMANTIC option. The internal database name is equated with the name of the database as specified in ADDS.

```
ID DIVISION.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SPECIAL-NAMES.  RESERVE SEMANTIC.
DATA DIVISION.
DATA-BASE SECTION.
DB  PROJEMP = "ORGANIZATION"  VALUE OF DBKIND IS SEMANTIC.
QD  PEMPQ OF PROJECT-EMPLOYEE.
QD  ASSIQ OF ASSIGNMENT.
```

Declaring a Database in SIM

A database declaration specifies the name and type of database that you are using in a query. Database declarations appear in the DATA-BASE SECTION of the DATA DIVISION.

You can specify an internal and an external name for the database. You can also declare the database to be either a SIM or a DMSII database. If multiple SIM databases are specified, the database name can be used as the final item to resolve qualification conflicts.

The COBOL74 interface supports the use of both DMSII and SIM databases in the same program and in separately compiled modules. A program can also have DMSII and SIM access to a single database. This support enables you to migrate programs with DMSII databases to SIM databases more easily.

The maximum number of SIM databases that can be declared in a single program is 47, of which 5 can be open simultaneously. The system can have up to 50 databases open at one time.

General Format

The general format for a database declaration is as follows:

```
DB data-name-1 [ = data-name-2]
[GLOBAL]
[VALUE OF DBKIND IS {DMSII   }]
                    {SEMANTIC}
[VERSION IS integer-1]

[DIRECTORY IS literal-1]
[STATUS IS {PRODUCTION}]
           {TEST      } .
```

Explanation of Format Elements

data-name-1

This format element identifies the internal name of the database.

data-name-2

This format element identifies the name of the database as specified in ADDS.

By default, if no dictionary is specified, the system searches for the database in the ADDS default DATADICTIONARY. If the SIM database is in an ADDS data dictionary other than the default DATADICTIONARY, you must use the DICTIONARY statement to identify the name of the dictionary.

GLOBAL

This option declares a database to be global, enabling a separately compiled procedure or program to reference the database declared in your COBOL74 program. The database reference in the separately compiled procedure or program must match the database reference in your COBOL74 program exactly.

VALUE OF DBKIND

This option specifies the database management system that is to manage the specified database.

DMSII or SEMANTIC

If the DBKIND is SEMANTIC, the RESERVE clause with the SEMANTIC option must be specified. For more information, refer to "Using the RESERVE Clause" earlier in this section. DMSII is the default database management system.

VERSION IS integer-1

This option provides the version of the database. Integer-1 must be a numeric literal with a maximum of 6 digits. The literal must be a valid version number in the dictionary.

DIRECTORY IS literal-1

This optional literal provides the name of the directory under which the database is stored. It is valid only when the database is specified as SEMANTIC. It must be a valid directory in the dictionary.

STATUS IS PRODUCTION or STATUS IS TEST

You can define the status of the database in the dictionary to be PRODUCTION or TEST. Use of these options restricts the invocation of entities to those with a particular status or subordinate status. The status specified in the database declaration takes precedence over any program status that is declared using the DICTIONARY compiler control option.

Considerations for Use

When binding is applied, a database is bound to any other database having the same internal name, if the database is either in one of the following locations or in a combination of both:

- The main program
- Separately compiled procedures or programs when GLOBAL is specified

See Also

- For more information on using the DICTIONARY statement, see "Identifying a Dictionary" in Section 2, "Using the ADDS Program Interface."
- For more information on database status, see the *InfoExec ADDS Operations Guide*.
- For more information on the DICTIONARY compiler control option, see "Accessing Entities with a Specific Status" in Section 2, "Using the ADDS Program Interface."
- More information on the RESERVE clause with the SEMANTIC option is provided in "Using the RESERVE Clause," earlier in this section.

Example

The following example shows database declarations with the VALUE OF, VERSION, DIRECTORY, and STATUS options:

```
DATA DIVISION.
DATA-BASE SECTION.
DB  UNIVERSITY-DATABASE
        VALUE OF DBKIND IS SEMANTIC.
DB  MY-DATABASE = ACCOUNTING-DATABASE
        GLOBAL
        VALUE OF DBKIND IS SEMANTIC.
DB  NEW-DATABASE = PAYROLL-DATABASE
        VALUE OF DBKIND IS SEMANTIC
        VERSION IS 2
        DIRECTORY IS "FRED"
        STATUS IS PRODUCTION.
DB  PAYROLL-DB.
DB  STAT-DB
        VALUE OF DBKIND IS DMSII.
```

Mapping SIM Types into COBOL74

The type and size of COBOL74 items in program variables and query records can differ from the type and size of their corresponding SIM descriptions. If valid COBOL74 types are used, as shown in Table 8–1, the compiler performs type coercion to convert the COBOL74 types to valid SIM types for use with the SIM system.

Table 8–1 describes the COBOL74 data types and indicates those that are valid for use with corresponding SIM types.

Table 8–1. Mapping SIM Types into COBOL74

SIM Type	Query Record Types	Program Variable Types
Integer, Population	BINARY	BINARY, COMP, DOUBLE, PIC 9(n) DISPLAY, REAL
Subrole	PIC X(n) DISPLAY	PIC X(n) DISPLAY
Date, Time	DISPLAY PIC X(8) or PIC X(10)	DISPLAY PIC X(8) or PIC X(10)
Real	REAL	BINARY, COMP, DOUBLE, PIC 9(n) DISPLAY, REAL
Number	COMP	BINARY, COMP, DOUBLE, PIC 9(n) DISPLAY, REAL
Character	PIC X(n) DISPLAY	PIC X(n) DISPLAY, BINARY, COMP, DOUBLE, REAL
Fixed string, Variable string, Symbolic	PIC X(n) DISPLAY	PIC X(n) DISPLAY
Kanji character	PIC X(n) KANJI	PIC X(n) KANJI
Kanji string	PIC X(n) KANJI	PIC X(n) KANJI
Boolean	PIC 9(n) COMP	BINARY, COMP, DOUBLE, PIC 9(n) DISPLAY, REAL
Compound attribute	Group	Group
Entity reference	ENTITYREFERENCE	ENTITYREFERENCE
Range	Base type	Base type
Enumeration	Base type	Base type

Symbolic SIM types (in the SIM Type column of Table 8–1) can be of any PIC X(n) DISPLAY query record type; however, the preferred representation is PIC X(30).

A literal value must be declared in the database as an allowable value without quotation marks for the specific SIM item of symbolic type to which it is assigned. For example, ASSIGN SINGLE TO MSTATUS gives the desired result, but ASSIGN "SINGLE" TO MSTATUS receives a syntax error.

The entry MOVE "SINGLE" TO TEMP-FIELD followed by ASSIGN TEMP-FIELD produces the desired result if TEMP-FIELD is declared PIC X(n), where *n* equals or is greater than 6.

Example of Assigning a Literal Value as a SIM Item of Symbolic Type

The following example shows the assignment of a literal value as a SIM item of symbolic type:

```
ASSIGN SINGLE TO MSTATUS OF PEMPQ.
INCLUDE [ MS TO DEGREE-OBTAINED,
Ø3370Ø     IF GRADE-POINT-AVG NOT < 3.5Ø
        EDUCATION OF EMPQ.
```

The SIM types for date and time (Table 8–1) must use a representation that enables the slash or colon dividers to be included in the appropriate form, as shown below. The MM/DD/YY form is valid only for input to SIM.

Date/Time	Form	Representation	Example
Date	MM/DD/YY	PIC X(8) DISPLAY	03/22/87
	MM/DD/YYYY	PIC X(10) DISPLAY	09/08/1988
Time	HH:MM:SS	PIC X(8) DISPLAY	14:54:32

Note that to reference a compound attribute, you must explicitly list one or more of its constituents. If you do not include at least one constituent, the program might compile but you could get unexpected results.

Example of Referencing a Compound Attribute

The following example shows the listing of individual components of a compound attribute:

```
QD EMP-Q.
Ø1 EMP-REC.
   Ø2  E-NAME.
       Ø3  E-LASTNAME     PIC X(2Ø).
       Ø3  E-MIDINITIAL   PIC X.
       Ø3  E-FIRSTNAME    PIC X(15).
   Ø2  E-RESIDENCE.
       Ø3  E-STREET       PIC X(3Ø).
       Ø3  E-CITY         PIC X(2Ø).
```

Table 8–2 describes all the COBOL74 data types, indicating with Y (yes) those types that are valid for use with the corresponding SIM types. The types that are labeled N (no) are not valid for use in any context in SIM.

The SIM data types are shown horizontally across the top, and the COBOL74 usage is shown vertically in the left-hand column.

Table 8–2. Valid Corresponding Types

SIM Type	Integer	Real	BCD	Boolean	EBCDIC String/ Value	Kanji String/ Value	Cursor	EVA
COBOL Query	N	N	N	N	N	N	Y	N
Compu-tational	Y	Y	Y	Y	N	N	N	N
REAL/ Double	Y	Y	Y	Y	N	N	N	N
EBCDIC PIC X	N	N	N	N	Y	N	N	N
EBCDIC PIC 9	Y	Y	Y	Y	N	N	N	N
ASCII	N	N	N	N	N	N	N	N
Kanji	N	N	N	N	N	Y	N	N
Index	N	N	N	N	N	N	N	N
Event/Lock	N	N	N	N	N	N	N	N
Control Point	N	N	N	N	N	N	N	N
Entity-reference	N	N	N	N	N	N	N	Y

Legend

BCD	Binary-coded decimal
COBOL	COBOL74
EVA	Entity-valued attribute

Qualifying Attributes

The following information explains the qualification that is needed for single-perspective and multiple-perspective queries.

Qualification for Single-Perspective Queries

In queries with one perspective, all attributes in the target list must connect to the perspective. If the attributes are immediate attributes of the perspective class, they are connected to the class by implicit qualification.

Examples

The examples presented in the following text use the following declaration:

```
QD   EMP-Q.
Ø1   EMP-REC.
     Ø2   E-LASTNAME        PIC X(2Ø).
     Ø2   E-SALARY          PIC 9(9)V2.
     Ø2   MNGR-LASTNAME     PIC X(2Ø).
```

Example of Single-Perspective Query by Implicit Qualification

In the following example, LAST-NAME and EMPLOYEE-SALARY are immediate attributes of the EMPLOYEE class. Thus, they are connected to the EMPLOYEE class by implicit qualification.

```
SELECT EMP-Q FROM EMPLOYEE
   (E-LASTNAME = LAST-NAME OF NAME,
    E-SALARY = EMPLOYEE-SALARY)
   WHERE EMPLOYEE-SALARY > 2ØØØØ.
```

Example of Qualification of a Single-Perspective Query with the OF Clause

The following example shows that if the attributes are extended attributes of the perspective class, you qualify them by using the OF clause with the name of an entity-valued attribute. In the following example, the explicitly declared perspective class is EMPLOYEE with the immediate attributes LAST-NAME and EMPLOYEE-SALARY. The extended attribute is also LAST-NAME but it connects to the MANAGER class through the EMPLOYEE-MANAGER attribute.

```
SELECT EMP-Q FROM EMPLOYEE
  (E-LASTNAME = LAST-NAME OF NAME,
   E-SALARY = EMPLOYEE-SALARY,
   MNGR-LASTNAME = LAST-NAME OF NAME OF EMPLOYEE-MANAGER)
  WHERE EMPLOYEE-SALARY > 2ØØØØ.
```

Example of a Qualification Path of a Single-Perspective Query over Several Classes

A qualification path can range over several classes through the use of several OF clauses. The path can even return to the perspective class. You can also access such circular paths with transitive closure, which uses the SET statement. In the following example, the path ranges over the DEPARTMENT class, through the use of the DEPT-TITLE OF DEPT-ASSIGNED attribute, and over the PROJECT class, through the use of the SUB-PROJECT-OF OF PROJECT attribute.

```
DEPT-TITLE OF DEPT-ASSIGNED OF SUB-PROJECT-OF OF PROJECT
```

See Also

See "SET Statement" in this section for information on accessing circular paths with transitive closure.

Qualification for Multiple-Perspective Queries

In queries with multiple-perspective classes, you designate each perspective class after the FROM clause. If an attribute is common to two or more classes, you need to explicitly qualify the attribute with the OF clause.

Example

The following example uses the following declaration:

```
QD   EMP-Q.
Ø1   EMP-Q-REC.
     Ø2   E-NAME       PIC X(2Ø).
     Ø2   STATUS       PIC X(1Ø).
     Ø2   MNGR-NAME    PIC X(2Ø).
     Ø2   BONUS        PIC 9(9).
```

The example shows the PROJECT-EMPLOYEE and MANAGER classes appearing as perspectives in the FROM clause. The LAST-NAME attribute is common to both the PROJECT-EMPLOYEE and the MANAGER classes and is appropriately qualified. The other attributes are unique to each class and are thus implicitly qualified.

```
SELECT EMP-Q FROM PROJECT-EMPLOYEE, MANAGER
  (E-NAME = LAST-NAME OF NAME OF PROJECT-EMPLOYEE,
   STATUS = STATUS,
   MNGR-NAME = LAST-NAME OF NAME OF MANAGER,
   BONUS = BONUS)
  WHERE BONUS = 1ØØ.
```

Considerations for Use

If you are accessing more than one database at a time, you can use a specific database with the OF clause. For example, assume DEPARTMENT exists in the PERSONNEL-PROJECT and BRANCH-PROJECT databases. To qualify DEPARTMENT as belonging to the PERSONNEL-PROJECT database, use the clause DEPARTMENT OF PERSONNEL-PROJECT.

Using the Query Declaration to Declare a Query

A query declaration identifies the query variable and lists the classes and query record variables used in the query. A query refers to both inquiry and update requests to a SIM database and consists of three elements:

- The query statement
- The query variable
- The query record

The query statement is sent to SIM to instruct the database about the actions to be performed. Query statements are constructed by the compiler from the SELECT, MODIFY, START MODIFY, INSERT, START INSERT, and DELETE statements. (Multiple-statement update assignments are constructed as query statement fragments.)

The query variable contains information about the state of the query.

The query record is basically a normal COBOL74 record (data area) into which data is placed when a retrieval query is executed. A query record can be used for multiple query statements, as long as the structure of the record is compatible with the data to be retrieved.

The two basic types of queries are

- Retrieval queries
- Update queries

Retrieval queries are always used with the SELECT statement. A retrieval query always has only one record (at the 01-level) per query. Update queries are used in transaction state to update entities using the attribute assignment statements (START MODIFY, START INSERT), and can be used for limited purposes with the SELECT statement.

Instead of using the SIM program interface, you can submit update and retrieval queries against a SIM database by using the SIM library entry points. A description of these entry points is provided in the *A Series InfoExec Semantic Information Manager (SIM) Object Manipulation Language (OML) Programming Guide.*

The CURRENT function can be used with both types of queries.

See Also

- For information on functions, refer to "Using Functions."
- For information on the CURRENT function, refer to "Special Constructs."
- Refer to the *InfoExec SIM Programming Guide* for information about query concepts, use of update and retrieval queries, and the use of the CURRENT function with queries.

- For more information about the statements used with queries, refer to "Using Transactions," "Updating and Deleting Entities," and "Selecting and Retrieving Entities."
- For information on deactivating a query, refer to "DISCARD Statement."

General Format

The general format for a query declaration is as follows:

```
DATA-BASE SECTION.
[QD query-name-1 [GLOBAL]
      {[OF data-name-1.]}
      {.              }
      [01 data-name-2.
            [level-number-1 data-name-3
                  [usage-clause-1]
                  [picture-clause-1].
                  [level-number-2 data-name-4
                        [usage-clause-2]
                        [picture-clause-2].]]...]
```

Explanation of Format Elements

QD query-name-1

The level indicator QD identifies the beginning of a query declaration and must precede the name of the query variable. The name of the query variable is identified by query-name-1.

GLOBAL

This option declares a database to be global, enabling a separately compiled procedure or program to reference the database declared in your COBOL74 program. The database reference in the separately compiled procedure or program must match the database reference in your COBOL74 program exactly.

OF

If the query is used for retrieving data, a query record is required. For updates, or for CURRENT-only retrieval, the OF option is required. A query record cannot contain a FILLER item at any level.

data-name-1

This format element must identify a database class.

01 data-name-2

The 01 data-name-2 option is the COBOL74 form for the query record, which allocates a data area for the query.

level-number-1 and level-number-2

These format elements can be any number from 02 to 49. Level-number-1 and level-number-2 are generally the same number. In the case of components of compound attributes, level-number-2 is greater than level-number-1 to show nesting.

data-name-3 and data-name-4

These format elements are associated with attributes of the database in the SELECT statement. Data-name-3 and data-name-4 can be in any COBOL74 identifier format.

picture-clause-1, picture-clause-2, usage-clause-1, and usage-clause-2

These format elements are described in Volume 1. Only the PICTURE and USAGE clauses are allowed on levels below 01.

Considerations for Use

When binding is applied, a query declaration is bound to any other query declaration with the same internal name when the query declaration is either in one of the following places, or in a combination of both:

- The main program
- Separately compiled procedures or programs when the GLOBAL option is specified

See Also

For more information on records, levels, and level numbers in COBOL74, refer to Volume 1.

Examples

Example of a Query Declaration with the OF Option

The following example shows a query declaration with the OF option:

```
QD  PROJ-Q  OF PROJECT.
```

Example of a Query Declaration with the Same Level Number

The following example shows a query declaration with PEMP-REC as the COBOL74 form for the query record. Two elementary items are declared at the 02 level.

```
DATA DIVISION.
DATA-BASE SECTION.
QD  PEMP-Q.
Ø1  PEMP-REC.
    Ø2  PEMP-LASTNAME  PIC X(2Ø).
    Ø2  PEMP-AGE       PIC 9(3) .
```

Example of a Query Declaration with Compound Attributes

The following example shows a query declaration with more than one level number:

```
QD MAN-Q.
Ø1 MAN-Q-REC.
   Ø2  MAN-LASTNAME PIC X(2Ø).
   Ø2  MAN-RES.
       Ø3 MAN-STREET PIC X(2Ø).
       Ø3 MAN-ZIP PIC 9(5) COMP.
   Ø2  MAN-AGE PIC 9(11) BINARY.
```

Opening and Closing a Database

After you declare a database, you can use the OPEN and CLOSE statements to open and close the database.

OPEN Statement

The OPEN statement opens a database for subsequent access and specifies the access mode.

General Format

The general format for the OPEN statement is as follows:

```
OPEN [INQUIRY | UPDATE] data-name-1
[ ON EXCEPTION { imperative-statement-1 | conditional-statement-1 | NEXT SENTENCE } ] .
```

Explanation of Format Elements

INQUIRY

This option enforces read-only access to the database. It is specified when no update operations are to be performed on the database. The default option is INQUIRY. If this option is specified, an exception is returned to the following statements:

ABORT-TRANSACTION	END-TRANSACTION
APPLY INSERT	EXCLUDE
APPLY MODIFY	INCLUDE
BEGIN-TRANSACTION	INSERT
CANCEL TRANSACTION POINT	MODIFY
DELETE	SAVE TRANSACTION POINT

UPDATE

This option enables the program to modify the database. An exception is returned if the database is already open. If an exception is returned, the state of the database remains unchanged.

data-name-1

This format element identifies a database.

ON EXCEPTION

The ON EXCEPTION option is described under "Handling SIM Exceptions" later in this section.

Example

The following example shows the OPEN statement used with the UPDATE option:

```
OPEN UPDATE PROJEMPDB.
```

CLOSE Statement

The CLOSE statement closes a database when further access is no longer required.

General Format

The general format for the CLOSE statement is as follows:

```
CLOSE data-name-1
[ ON EXCEPTION { imperative-statement-1 }
               { conditional-statement-1 } ].
               { NEXT SENTENCE           }
```

Explanation of Format Elements

data-name-1

This format element identifies the name of a database.

ON EXCEPTION

The ON EXCEPTION option is described under "Handling SIM Exceptions" later in this section. An exception is returned if the database is not open.

Example

The following example shows the use of the CLOSE statement:

```
CLOSE PROJEMPDB.
```

Using Transactions

You usually use two or more query statements as a unit; however, for SIM to treat two or more statements as a unit, you must group them within a transaction. During transaction state, you can update the database.

A transaction consists of a BEGIN-TRANSACTION statement and an END-TRANSACTION statement that bind a series of statements. When you use transaction statements, updated data is applied but not actually committed until the END-TRANSACTION statement is executed. When the ABORT-TRANSACTION statement is used, all updates that have been applied since the BEGIN-TRANSACTION statement are backed out and the program is taken out of transaction state. If the database involved in a transaction is closed before the END-TRANSACTION statement is executed, the transaction is terminated by the database system and all applied data is backed out.

You can create and cancel intermediate transaction points within the transaction with the SAVE TRANSACTION POINT and CANCEL TRANSACTION POINT statements. The advantage of using intermediate transaction points is that you can better control and correct transactions. Changes made against a database can be undone and the progress of one or more transactions can be rolled back to a previously consistent state. For example, if you use the SAVE TRANSACTION POINT statement to create intermediate points, a CANCEL TRANSACTION POINT statement can be used to back out data applied since an intermediate point or since the BEGIN-TRANSACTION statement, without terminating the transaction or leaving transaction state.

Although you can select and retrieve either in or out of transaction state, you can update the database only in transaction state. The following statements can be executed only in transaction state:

ABORT-TRANSACTION	EXCLUDE
APPLY INSERT	INCLUDE
APPLY MODIFY	INSERT
CANCEL TRANSACTION POINT	MODIFY
DELETE	SAVE TRANSACTION POINT
END-TRANSACTION	

A transaction can update only one database. The database to be updated is identified by the first operation, which can be one of the above update operations or a data retrieval operation. Updating a database that is different from the one identified causes an exception condition.

See Also

- For information on updating a database, see "Updating and Deleting Entities" later in this section.
- Information on selecting and retrieving data is provided in "SELECT Statement" and "RETRIEVE Statement" later in this section.

Example

The following example shows the use of transaction statements in a program:

```
P-1.
    BEGIN-TRANSACTION.
    DELETE PERSON WHERE FIRST-NAME OF NAME = "John"
          AND LAST-NAME OF NAME = "Doe".
       ON EXCEPTION
          ABORT-TRANSACTION
          GO TO P-1-EXIT.
    SAVE TRANSACTION POINT 1.
    MODIFY PERSON
       ASSIGN 34 TO AGE
       WHERE FIRST-NAME OF NAME = "Mary"
          AND LAST-NAME OF NAME = "Jones".
    DELETE PERSON WHERE FIRST-NAME OF NAME = "John"
          AND LAST-NAME OF NAME = "Smith".
       ON EXCEPTION
          CANCEL TRANSACTION POINT.
P-1-EXIT.
    EXIT.
```

The following information explains the statements used to begin, cancel, and end transaction state. It also describes the statements used to create and cancel intermediate transaction points. The statements are presented in alphabetical order.

ABORT-TRANSACTION Statement

The ABORT-TRANSACTION statement backs out all data applied since the BEGIN-TRANSACTION statement and takes the program out of transaction state.

General Format

The general format for an ABORT-TRANSACTION statement is as follows:

```
ABORT-TRANSACTION [COMS-header-name-1]
[ ON EXCEPTION { imperative-statement-1   } ] .
               { conditional-statement-1  }
               { NEXT SENTENCE            }
```

Explanation of Format Elements

COMS-header-name-1

This option specifies the COMS header. This call is made before the exception-handling procedure is executed.

The COMS-header-name-1 option is used only with COMS. The COMS-header-name-1 option causes the COBOL74 program to call the DCIENTRYPOINT of a Data Communications Interface (DCI) library when an exception condition is detected. This program call enables a program interfacing with COMS to support synchronized transactions and recovery.

ON EXCEPTION

The ON EXCEPTION option is described under "Handling SIM Exceptions" later in this section.

See Also

Refer to Section 3, "Using the COMS Program Interface" for more information on COMS.

BEGIN-TRANSACTION Statement

The BEGIN-TRANSACTION statement places a program in transaction state.

General Format

The general format of a BEGIN-TRANSACTION statement is as follows:

```
BEGIN-TRANSACTION [EXCLUSIVE]
[COMS-header-name-1 [USING identifier-1]]
[ ON EXCEPTION { imperative-statement-1 }
               { conditional-statement-1 } ].
               { NEXT SENTENCE           }
```

Explanation of Format Elements

EXCLUSIVE

This option is used to perform a long transaction against a SIM database. If you specify this option, you are assigned to an exclusive transaction state. In this transaction state, you are guaranteed exclusive access to the database and have exclusive control of all structures in the database. However, to prevent deadlocks, the program must wait until all other programs have finished processing before it can gain exclusive access. Because exclusive access has a high performance cost, it should be used sparingly.

COMS-header-name-1 USING identifier-1

This option is used only with COMS. Identifier-1 specifies the message area. These constructs enable a program interfacing with COMS to support transactions and a batch option when using archival recovery.

COMS-header-name-1 specifies the COMS input header. If the optional COMS-header-name-1 phrase is used and no exception condition is detected, the call on the DCI library is made and the message area specified by identifier-1 is passed to the DCIENTRYPOINT of the DCI library.

ON EXCEPTION

The ON EXCEPTION option is described under "Handling SIM Exceptions" later in this section.

See Also

For more information on recovery, refer to the *COMS Programming Guide*.

CANCEL TRANSACTION POINT Statement

The CANCEL TRANSACTION POINT statement backs out all updates in a transaction to an intermediate transaction point or to the beginning of the transaction. The program remains in transaction state after the CANCEL is performed.

General Format

The general format of a CANCEL TRANSACTION POINT statement is as follows:

```
CANCEL TRANSACTION POINT [arithmetic-expression-1]
[ ON EXCEPTION { imperative-statement-1 }
               { conditional-statement-1 } ].
               { NEXT SENTENCE           }
```

Explanation of Format Elements

arithmetic-expression-1

The CANCEL TRANSACTION POINT statement backs out all database changes made between the current point in the transaction and the point specified by arithmetic-expression-1.

If arithmetic-expression-1 is not specified, all data updated since the BEGIN-TRANSACTION statement is backed out.

ON EXCEPTION

The ON EXCEPTION option is described under "Handling SIM Exceptions" later in this section.

See Also

For details on arithmetic expressions, see "Arithmetic Expressions" later in this section.

END-TRANSACTION Statement

The END-TRANSACTION statement commits all updates applied within a transaction, and takes a program out of transaction state.

General Format

The general format of an END-TRANSACTION statement is as follows:

```
END-TRANSACTION [COMS-header-name-1 [USING data-name-1]]
[ ON EXCEPTION { imperative-statement-1   } ] .
               { conditional-statement-1  }
               { NEXT SENTENCE            }
```

Explanation of Format Elements

COMS-header-name-1

This option specifies the COMS output header. This call is made before the exception-handling procedure is executed.

COMS-header-name-1 is used only with COMS. COMS-header-name-1 causes the COBOL74 program to call the DCIENTRYPOINT of a DCI library when an exception condition is detected. This program call enables a program interfacing with COMS to support synchronized transactions and recovery.

USING data-name-1

The USING option enables the message area indicated by data-name-1 to be passed to the DCIENTRYPOINT when the call is made on the DCI library.

ON EXCEPTION

The ON EXCEPTION option is described under "Handling SIM Exceptions" later in this section.

See Also

Refer to Section 3, "Using the COMS Program Interface," for more information on COMS.

SAVE TRANSACTION POINT Statement

The SAVE TRANSACTION POINT statement establishes an intermediate transaction point.

General Format

The general format of a SAVE TRANSACTION POINT statement is as follows:

```
SAVE TRANSACTION POINT arithmetic-expression-1
[ ON EXCEPTION { imperative-statement-1   } ] .
               { conditional-statement-1  }
               { NEXT SENTENCE            }
```

Explanation of Format Elements

arithmetic-expression-1

For details on arithmetic expressions, see "Arithmetic Expressions" later in this section.

The SAVE TRANSACTION POINT statement assigns a marker (denoted by arithmetic-expression-1) to the present execution point in the transaction. SIM requires that the marker be a positive, nonzero, unique number after it is truncated to an integer.

ON EXCEPTION

The ON EXCEPTION option is described under "Handling SIM Exceptions" later in this section.

Declaring an Entity Reference Variable

An entity reference variable is used to contain an explicit reference to a database entity.

Variables that contain entity reference values are called entity reference variables. These variables can be assigned and compared only with other entity reference variables that have been declared with ENTITYREFERENCE usage of the same class. Entity reference values are valid only in the transaction state in which they are retrieved.

To declare an item as an entity reference variable, use the USAGE clause.

See Also

For detailed information about entity reference variables, refer to the *InfoExec SIM Programming Guide*.

General Format

The general format of the declaration for the entity reference variable is as follows:

```
level-number-1 data-name-1
USAGE IS ENTITYREFERENCE OF data-name-2
[{OCCURS} integer-1 [TIMES]].
 {OC    }
```

Explanation of Format Elements

level-number1

This level number can be any number from 01 to 49.

data-name-1

This format element is associated with the entity-valued attributes of the database.

USAGE IS ENTITYREFERENCE OF

An item with ENTITYREFERENCE usage is compatible only with other items of the same class having ENTITYREFERENCE usage. The item is not alphabetic, alphanumeric, or numeric, so it cannot be used in statements that associate it with such items.

An item with ENTITYREFERENCE usage can be accessed only within transaction state.

data-name-2

This format element identifies a database class.

OCCURS or OC

This option functions in the same way as any COBOL74 OCCURS clause. See Volume 1 for details on the use of this clause.

Example

The following is an example of a declaration for an entity reference variable:

```
Ø1  PEMP-1 USAGE ENTITYREFERENCE OF PROJECT-EMPLOYEE.
```

Using Functions and Expressions

This explanation includes the COBOL74 formats for using functions and expressions, and presents some tables summarizing SIM information for easy reference.

See Also

For a detailed discussion of the use of functions and expressions in SIM, refer to the *InfoExec SIM Programming Guide*.

Using Functions

SIM supports the following functions:

- Aggregate functions
- Arithmetic functions
- Special constructs
- String functions
- Symbolic functions
- Time and date functions

Aggregate Functions

Table 8–3 summarizes the aggregate functions available in the COBOL74 program interface with SIM. Aggregate functions apply a function to a set of entities to produce a single scalar result. Some of these functions appear in the diagrams for expressions shown later in this section.

Table 8–3. Aggregate Functions

Function	Explanation
AVG	Average. Calculates the mean of a collection of numeric values.
COUNT	Can be used with classes or attributes. Can be used to count class entities or nonnull values, or to indicate null or nonnull attribute values.
MIN	Minimum. Can be applied to any path expression that results in a collection of values with an ordered type.
MAX	Maximum. Can be applied to any path expression that results in a collection of values from an ordered type.
SUM	Yields the arithmetic sum of the collection of numeric values to which it is applied.

Example

The following example shows the use of the AVG aggregate function:

```
DMFUNCTION AVG (AGE OF PROJECT-TEAM)
```

Arithmetic Functions

SIM supports the arithmetic functions shown in Table 8–4.

Table 8–4. Arithmetic Functions

Function	Explanation
ABS (expression)	Absolute value. Returns the absolute value of the argument. The result is of the same type as the argument.
ROUND (expression)	Returns an expression rounded to the nearest integer.
SQRT (expression)	Square root. Returns a real number that is the square root of the argument. The value of the expression must evaluate to a nonnegative number.
TRUNC (expression)	Truncate. Returns the integer portion of the argument.

Special Constructs

Table 8–5 summarizes the special constructs available in the COBOL74 interface with SIM. These constructs cannot be categorized as traditional functions. Note that the introductory word DMFUNCTION does not precede these constructs when they are used in a program.

The quantifiers ALL, NO, and SOME can be used when specifying entities for selection to

- Create a new reference variable that ranges over the class or multivalued attribute (MVA) represented by the identifier
- Apply to the results of a path expression
- Embed quantifiers in an extended qualification

You cannot quantify a reference variable created by using the CALLED construct.

For the EQUIV construct, the ordering sequence is the arrangement of members of a character set according to a predetermined scheme. Different elements can have the same ordering attribute, for example, the letters *a* and *A*.

Table 8–5. Special Constructs

Construct	Explanation
ALL	Indicates that each value of the attribute must meet the condition.
CALLED	Assigns a variable to a certain set of entities for reference by the variable.
CURRENT	Refers to an entity specified in a selection expression for an active retrieval query.
EQUIV	Performs an equivalent comparison by using the ordering sequence values (OSVs) of characters. This construct is valid only for string expressions. The syntax for the EQUIV construct is as follows: EQUIV (string1 Relational Operator clause string2)
INVERSE	Accesses the inverse of an entity-valued attribute.
NO	Indicates that no value can meet the condition.
SOME	Indicates that at least one value must meet the condition.
TRANSITIVE	Returns the transitive closure of a cyclical path expression. An END clause can be used to specify levels of recursion to be included, if a complete closure is not performed. The TRANSITIVE function can be used only with the embedded SELECT statement. Additional information is provided in the *InfoExec SIM Programming Guide*.

Examples

The following are examples of special constructs:

```
CURRENT OF PEMP-Q

EQUIV (LAST-NAME-1 IS GREATER THAN LAST-NAME-2)

TRANSITIVE (SUB-PROJECTS)
```

String Functions

String functions can perform either or both of the following:

- Take one or more strings as arguments.
- Produce a string as the function value.

Table 8–6 describes the string functions and their use with SIM.

Table 8–6. String Functions

Function	Syntax	Explanation
Character	CHR (hexnum[, hexnum...])	Constructs a string that is a concatenation of the EBCDIC characters represented by the hexadecimal numbers that are its arguments. This is useful for including nonprintable characters in a string.

Extract	EXT (string,integer1,integer2)	Returns a substring of the string from the character position indicated by integer1 to the character position indicated by integer2.
		Integer1 can be any positive integer.
		Integer2 must be greater than integer1.
		To denote the end of the string, use an asterisk (*) in place of integer2.
		For example,
		`EXT("Monday",1,3) = "Mon"`
		`EXT("Monday",4,*) = "day"`

Length	LENGTH (string)	Returns the number of characters in the string. For example, `LENGTH("December") = 8` If the string is from a fixed-length string, the value returned is the declared length. For a variable-length string, the value returned is the maximum number of characters the string can contain.

continued

Table 8–6. String Functions (cont.)

Function	Syntax	Explanation

Position	POS (string1,string2,integer)	Returns the character position of string1 in string2, at or after the character position indicated by the integer. The value returned is always relative to the position of the first character in string2. If the integer is omitted, the position of the first appearance of string1 is returned.
		
		For example,
		
		`POS("e","December")  = 2`
		`POS("e","December",3) = 4`
		`POS("e","December",5) = 7`

Repeat	RPT (string,integer)	Returns the string repeated the number of times indicated by the integer. For example, `RPT("ab",3) = "ababab"`

Example

The following example shows the use of the LEN function to find the length of a string:

```
DMFUNCTION LEN (PROJECT-TITLE OF CURRENT-PROJECT)
```

Symbolic Functions

Table 8–7 lists the symbolic functions available in SIM for use in a COBOL74 program. Symbolic functions operate upon SIM symbolic types, which have values that are identifiers.

Table 8–7. Symbolic Functions

Function	Explanation
PRED	Predecessor. This function operates on ordered symbolics and returns the previous symbolic value.
SUCC	Successor. This function operates on ordered symbolics and returns the successive symbolic value.

Time and Date Functions

Table 8–8 summarizes the time and date functions available in SIM. Some of these functions appear in the expression formats shown in this section.

See Also

For information on the SIM types for date and time, refer to "Mapping SIM Types into COBOL74," earlier in this section.

Table 8–8. Time and Date Functions

Function	Explanation
ADD-DAYS	Takes the date and a positive or negative increment in days to produce a new date. A positive increment returns a future date and a negative increment returns a previous date. The date must be a valid constant without quotation marks, for example 09/08/88.
ADD-TIME	Takes the time and an increment in seconds to produce a new time. A positive increment returns a future time and a negative increment returns a previous time. The time must be a valid constant without quotation marks, for example 14:01:22.
CURRENT-DATE	Parameterless function of type DATE that returns the date for today.
CURRENT-TIME	Parameterless function of type TIME that returns the current time of day.
DAY	Returns a day of the month, in numeric form 1 through 31.
DAY-OF-WEEK	Returns a day of the week, as a string.
ELAPSED-DAYS	Returns elapsed days between two dates.
ELAPSED-TIME	Returns the difference in seconds between two expressions of time.
HOUR	Returns the hour portion of a time expression.
MINUTE	Returns the minute portion of a time expression.
MONTH	Returns the month portion of a date expression, in numeric form 1 through 12.
MONTH-NAME	Returns the name of the month, as a string.
SECOND	Returns the seconds portion of a time expression.
YEAR	Returns the year, as a 4-digit numeric.

Examples

The following are examples of date and time functions.

Examples of Time Functions

The following are examples of the CURRENT-TIME, HOUR, MINUTE, and SECOND time functions:

```
DMFUNCTION CURRENT-TIME

DMFUNCTION HOUR (EST-PERSON-HOURS)

DMFUNCTION MINUTE (EST-PERSON-HOURS)

DMFUNCTION SECOND (EST-PERSON-HOURS)
```

The following are examples of the ELAPSED-TIME and ADD-TIME time functions:

```
DMFUNCTION ELAPSED-TIME(STARTING-TIME, QUITTING-TIME)

DMFUNCTION ADD-TIME(EST-PERSON-HOURS,3600)
```

Examples of Date Functions

The following examples show the CURRENT-DATE, YEAR, MONTH, and DAY date functions:

```
DMFUNCTION CURRENT-DATE

DMFUNCTION YEAR(EMPLOYEE-HIRE-DATE)

DMFUNCTION MONTH(EMPLOYEE-HIRE-DATE)

DMFUNCTION DAY(EMPLOYEE-HIRE-DATE)
```

The following are examples of the ELAPSED-DAYS and ADD-DAYS date functions:

```
DMFUNCTION ELAPSED-DAYS (START-DATE OF PROJECT,END-DATE OF PROJECT)

DMFUNCTION ADD-DAYS(EMPLOYEE-HIRE-DATE,7)
```

The following examples show the DAY-OF-WEEK and MONTH-NAME date functions:

```
DMFUNCTION DAY-OF-WEEK(EMPLOYEE-HIRE-DATE)

DMFUNCTION MONTH-NAME(EMPLOYEE-HIRE-DATE)
```

Using Expressions

Data management (DM) expressions are used within database statements to determine values or entities. The selection expression is a special kind of DM expression used to select entities for retrieval and update.

The following expressions are available in the program interface. The formats and conventions of each expression are explained in the discussion that follows this list.

- Arithmetic expressions
- Conditional expressions
- Selection expressions
- String expressions
- Formats for expressions used with SIM

See Also

- For a detailed explanation of the use of selection expressions with SIM, refer to *InfoExec SIM Programming Guide*.
- For general information on conditions and arithmetic expressions, refer to Volume 1.

Arithmetic Expressions

Arithmetic expressions can contain constants, built-in or user-defined functions, and arithmetic attributes. Only one multivalued attribute (MVA) can be used in an arithmetic expression.

If either operand of an arithmetic expression is null, the result is null. Note that at least one space must appear between each operand and operator within an expression.

The following expressions are valid:

Expression	Meaning
+	Addition
–	Subtraction
*	Multiplication
/	Division
DIV	Integral division
MOD	Modulus
**	Exponentiation

Conditional Expressions

Conditional expressions identify conditions that are tested so the program can choose between two paths of control, depending on the truth value of the condition.

Table 8–9 specifies the valid operators. These operators can be used to test ordered types. Two type-compatible operands can be compared using IS EQUAL TO or IS NOT EQUAL TO. Only IS EQUAL TO or IS NOT EQUAL TO can be used with compound types or any nonordered types (such as Kanji or entity). A compound type can be compared only to another compound type with the same sequence of components; a compound type cannot be compared to any COBOL74 type. To compare a compound type to a COBOL74 group, you must write individual comparisons for the components and elementary items. Scalar relational operators can be used with multivalued expressions.

Table 8–9. Conditional Expression Operators

Operator	Explanation
AND	Logical conjunction.
EXISTS	Unary postfix operator that evaluates to TRUE if the argument is not a null value. Can be used with either immediate or extended data-valued or entity-valued attributes.
IS EQUAL TO or =	Equal to. Valid with all types.
IS NOT EQUAL TO	Not equal to. Valid with all types.
IS GREATER THAN or >	Greater than. Valid only with ordered types.
IS LESS THAN or <	Less than. Valid only with ordered types.
IS AFTER	Equivalent to IS GREATER THAN. Especially appropriate for types DATE and TIME, and for user-defined ordered types. Can be used with any ordered type.
IS BEFORE	Equivalent to IS LESS THAN. Especially appropriate for types DATE and TIME, and for user-defined ordered types. Can be used with any ordered type.
ISA	Tests to discover if an entity plays a particular role in a class. Evaluates to TRUE only if the entity identified by the first operand is a member of the class identified by the second operand.
NOT	Logical negation or reversal of truth value.
OR	Logical inclusive OR.

See Also

- For information on the use of operators in the Condition and Relational Operator expression clauses used with SIM, see "Expression Formats" later in this section.
- For information on the IS IN operator, refer to "String Expressions" later in this section.
- For general information about conditional expressions, refer to Volume 1.

Selection Expressions

A selection expression can be used to identify the set of entities upon which a query is to operate. It narrows the group of entities in the perspective class and in classes of interest for the scope of the query. A selection expression can be either global or local.

A global selection expression applies to the whole query. A local selection expression applies only to specific attributes.

If a local selection expression is applied to a class or an entity-valued attribute, the expression can contain only attributes that are immediate or extended attributes of the class to which the WITH clause refers (the class named, or the class that the entity-valued attribute ranges over).

General Format for a Global Selection Expression

The format for a global selection expression is as follows:

```
WHERE selection-expression-1
```

Example of a Global Selection Expression

The following example shows the code for a global expression:

```
SELECT PEMP-Q FROM PROJECT-EMPLOYEE
(NUM-HOURS = EST-PERSON-HOURS OF ASSIGNMENT-RECORD)
     WHERE EST-PERSON-HOURS OF ASSIGNMENT-RECORD > 40.
```

General Format for a Local Selection Expression

The format for a local selection expression is as follows:

```
WITH selection-expression-1
```

Explanation of Format Element

WITH

This clause can appear only in the target list of a query, or within a built-in function. The selection expression in the WITH clause affects only the attribute to which it is applied.

When you use RETRIEVE statements with statements that contain local selection expressions, each RETRIEVE statement obtains the next entity in the query and then compares the values of the entity against the criteria in the local selection expression. If the values of an entity do not satisfy the local selection expression, a null value is returned to the program for the attribute with the local selection expression, instead of the actual value of the attribute. An error exception only occurs when there are no more entities to be retrieved.

Example of a Local Selection Expression

The following example shows the use of a local selection expression:

```
SELECT PEMP-Q FROM PROJECT-EMPLOYEE
    (NUM-HOURS = EST-PERSON-HOURS OF ASSIGNMENT-RECORD
     WITH EST-PERSON-HOURS OF ASSIGNMENT-RECORD > 40).
```

String Expressions

Concatenation is supported in SIM. The operator used in COBOL74 for concatenation is CAT.

Pattern matching occurs when a set of strings is established as a pattern. A string expression is then matched with the pattern. The string expression can be either a literal string or a string variable. The pattern must be a literal string consisting of the literal characters and the special characters shown in Table 8–10. The syntax pattern for pattern matching is

```
<string expression> IS IN <pattern>
```

Table 8–10. Special Characters Used for Pattern Matching in SIM

Character	Explanation
_	Matches any single character.
{ }	Denotes any character inside as a match.
..	Use only with the braces ({}). Denotes a range of matching letters and numbers. For example, "a..z" denotes a range of characters from "a" to "z". You cannot use another special character, such as a dash (–), within this range.
( )	Groups an expression.
|	Matches the pattern on either side of the bar.
""	Denotes all characters inside each set as literals, including special characters.

continued

Table 8–10. Special Characters Used for Pattern Matching in SIM (cont.)

Character	Explanation
*	Matches all instances of the expression preceding it. This expression can consist of a single character, an expression in parentheses, or a set of characters enclosed in braces or quotation marks.
?	Matches zero or more instances of any character. (This character performs the same function as _*).

Examples

The following examples use the IS IN operator for pattern matching.

- The code in this example returns the value FALSE because the only item generated from the pattern "123ABC" is "123ABC". The string expression "ABC" is not equal to "123ABC".

```
"ABC"  IS IN "123ABC"
```

- The code in this example returns the value TRUE because the pattern "123|ABC" contains two string items: 123 and ABC. The string expression "ABC" matches the ABC in the pattern.

```
"ABC"  IS IN "123|ABC"
```

- This example returns the value TRUE because the pattern "{ABC}*" contains all the strings that have zero or more concatenations of the literal ABC. Examples of these concatenations include blank ( ), ABC, ABCABC, and ABCABCABC.

```
"ABC"  IS IN "{ABC}*"
```

- The code in this example returns the value TRUE if MY-STRING contains zero or more concatenations of the literal 123 followed by AB.

```
MY-STRING IS IN "{123}*AB"
```

See Also

For additional information on the string expression, refer to the *InfoExec SIM Programming Guide*.

Expression Formats

Expressions are used to perform the following:

- Specify attributes or limit a set or group for selection.
- Select and assign values.

A DM expression begins with the Expression clause. A selection expression begins with the Boolean Expression clause. These expressions can be used in combination.

Information on expression formats is presented in three parts: a list that contains each DM expression clause and its description; a presentation of the general format for each expression clause; and an explanation of the components of each general format.

The clauses for DM expressions and selection expressions include the following:

Expression	Description
Boolean Expression clause	Selection expression used with Boolean operators AND, NOT, and OR
Boolean Primary clause	Essential part of the Boolean Expression clause
Condition clause	Indication of a relationship to or evaluation of TRUE or FALSE
Expression clause	DM expression indicator that is a program variable or an arithmetic result
Primary clause	Essential component of an Expression clause
Qualification Term clause	Designation of the terms to be qualified
Relational Operator clause	Component of a condition that presents the relational operators for the condition
Transitive Specification clause	Control for the level at which an entity search is done in transitive closure

General Formats

The general format for each expression format follows.

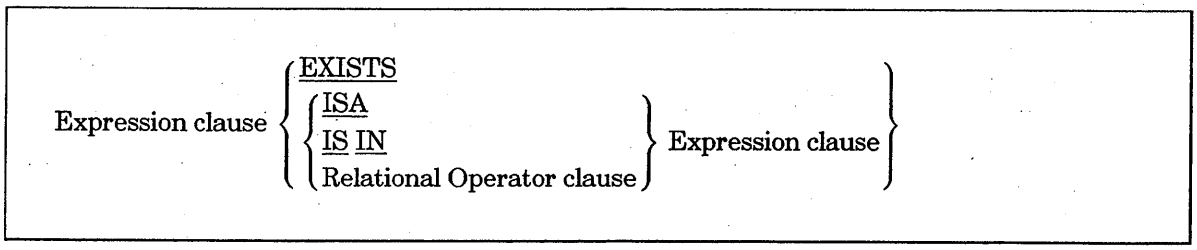
General Format for a Boolean Expression Clause

```
[NOT] Boolean Primary clause [ {AND} [NOT] Boolean Expression clause ]
                             {OR }
```

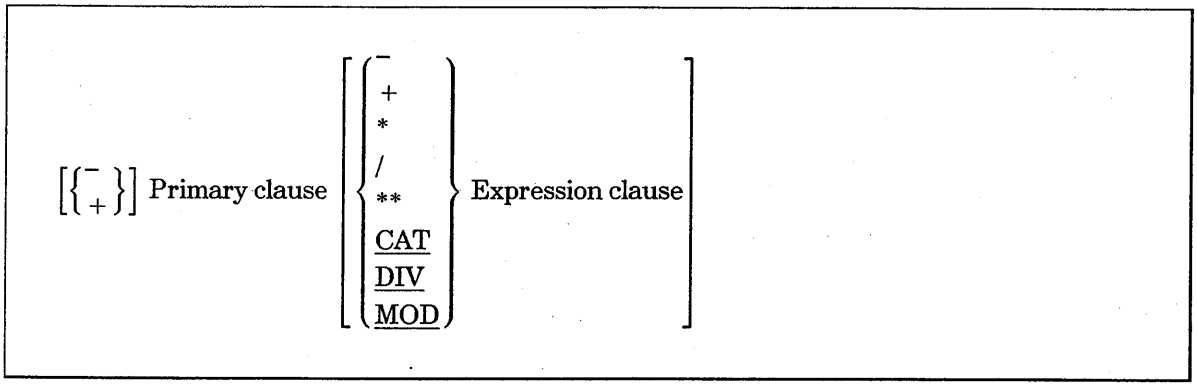
General Format for a Boolean Primary Clause

```
{ TRUE
  FALSE
  Qualification Term clause [OF Qualification Term clause]...
      [OF Transitive Specification clause]
  (Boolean Expression clause)
  Condition clause }
```

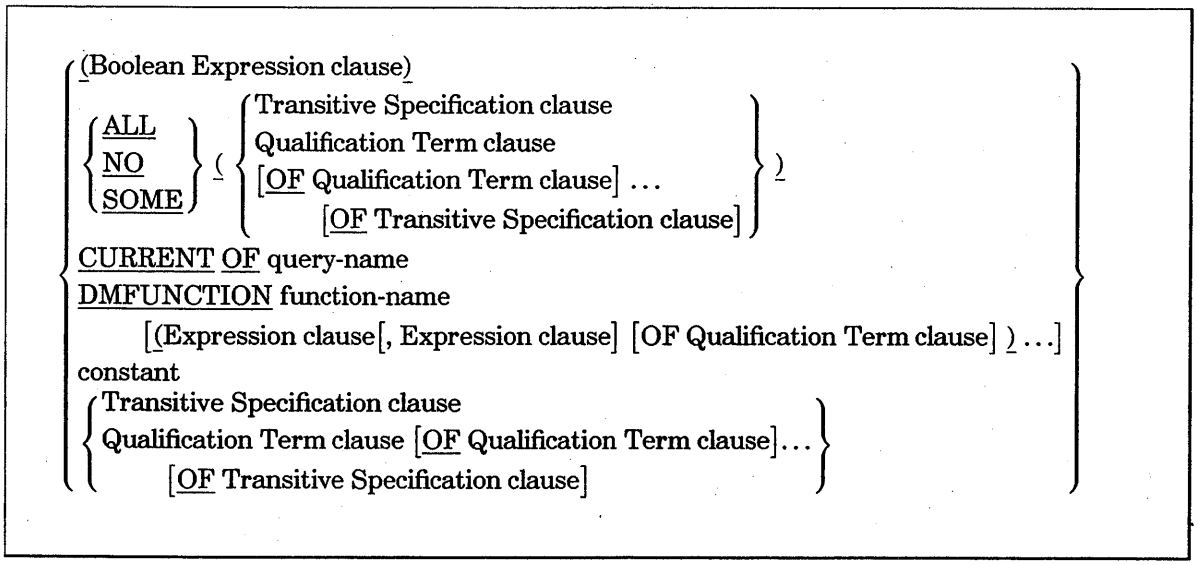
General Format for a Condition Clause



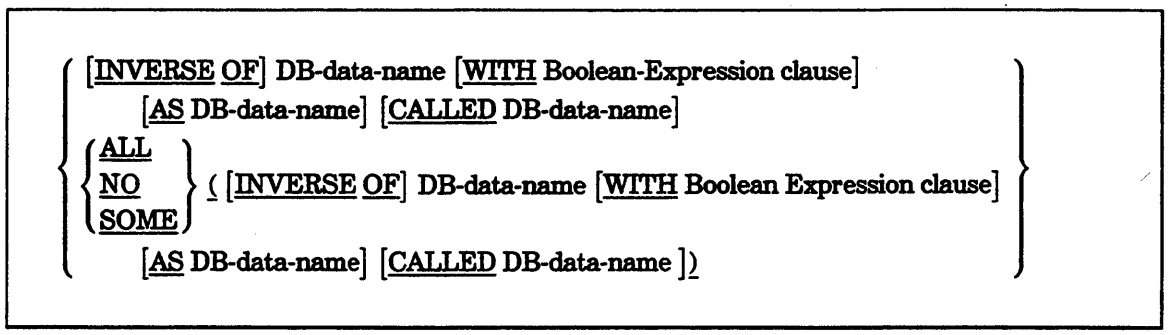
General Format for an Expression Clause



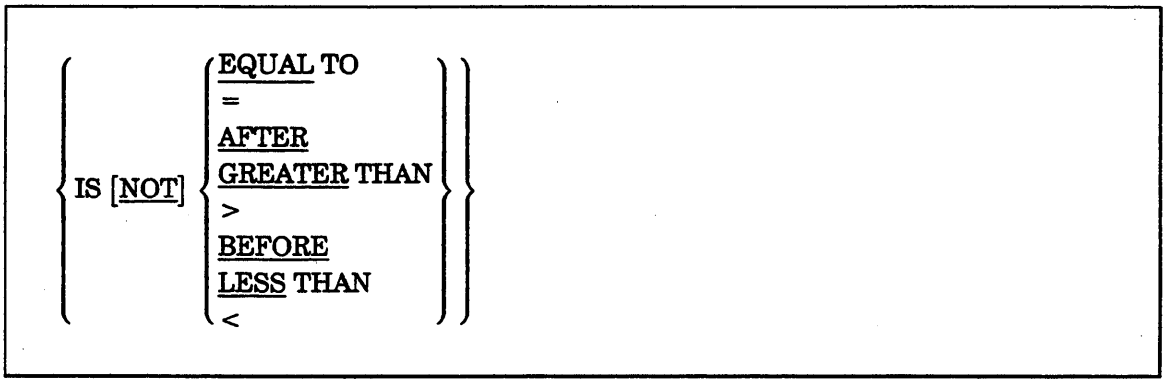
General Format for a Primary Clause



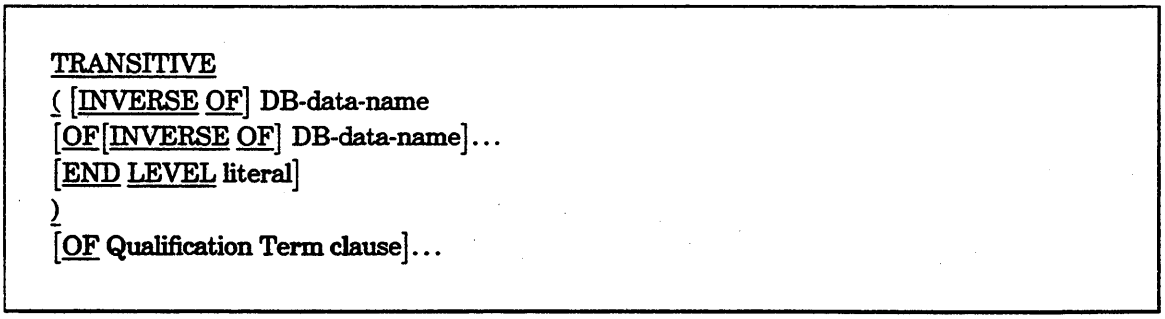
General Format for the Qualification Term Clause



General Format for the Relational Operator Clause



General Format for the Transitive Specification Clause



Explanation of Elements for Expression Formats

Table 8–11 shows the format elements for each of the expression formats. An expression format is not explained if it contains self explanatory elements or if it contains only nested expression formats that require no additional explanation.

Table 8–11. Expression Format Elements

Expression Format	Elements	Explanation of Element
Boolean Expression clause	NOT, AND, and OR Boolean Primary clause Boolean Expression clause	These format elements are Boolean operators.
Boolean Primary clause	TRUE, FALSE Qualification Term clause Transitive Specification clause Boolean Expression clause Condition clause	These format elements are Boolean values. The qualification term in the Boolean Primary clause must be a SIM Boolean item.
Condition clause	ISA, IS IN, and EXISTS Expression clause Relational Operator clause	More information about these operators is provided in "String Expressions" and "Conditional Expressions" earlier in this section.
Expression clause	–, +, *, /, **, CAT, DIV, MOD Primary clause Expression clause	More information about these operators is provided in "Arithmetic Expressions" earlier in this section.

continued

Table 8–11. Expression Format Elements (cont.)

Expression Format	Elements	Explanation of Element
Primary clause	Boolean Expression clause	
	ALL, NO, and SOME	These special constructs are quantifiers.
	Transitive Specification clause	
	Qualification Term clause	
	CURRENT	This special construct can be used to establish a relationship to an entity specified in a selection expression for an active retrieval query. Refer to "Special Constructs" earlier in this section for information on the CURRENT special construct.
	DMFUNCTION	This expression is forwarded to the database system for complete evaluation, so the arguments of the function can contain references to unretrieved values in the database. At run time, COBOL74 arithmetic expressions are evaluated to single values and passed to the database system by content.
	Expression clause	
	Constant	This format element is a number or string, or a program variable. If the constant is a declared variable in the program, the program variable takes precedence over any identically qualified SIM attribute and is used in processing the query. If the constant is not a declared program variable, the SIM database attribute is used. You must qualify the database attribute to uniquely identify it from the program variable if you want to assure correct selection. However, query record items are not considered as program variables in selection and DM expressions and cannot be used in these expressions.

continued

Table 8–11. Expression Format Elements (cont.)

Expression Format	Elements	Explanation of Element
Qualification Term clause	INVERSE OF	The INVERSE OF option invokes the INVERSE function to create a temporary inverse attribute from one class to another.
	Boolean Expression clause	
	DB-data-name	This format element can be any identifier declared in the database.
	WITH	This option applies the Boolean Expression clause as a local selection expression.
	AS	This option is used for role conversion. It qualifies a reference to an entity by role. The object of AS must be in the same hierarchy as the subject.
	CALLED	This special construct assigns a variable to a set of entities for reference by the variable.
Relational Operator clause	IS and relational operators	This clause presents relational operators for a relation condition.
Transitive Specification clause	INVERSE OF DB-data-name	The INVERSE OF option invokes the INVERSE function to create a temporary inverse attribute from one class to another.
	END LEVEL	The END LEVEL option specifies the level at which the search for an entity ends.
	Qualification Term clause	

Selecting and Retrieving Entities

The following statements are used in a program to activate a query and select the entities, and to retrieve and close the query:

- The SELECT statement is used to associate a selected set of entities with the query. It also enables you to assign database attributes to query variables you have defined.

  Selection expressions can be used within the SELECT statement to specify one, some, or all entities from a set.

- The SET statement is used to manipulate the levels of an attribute involving a selection and retrieval in transitive closure.
- The RETRIEVE statement is used to retrieve the query.
- The DISCARD statement terminates an activated query.

There are two uses of the SELECT statement:

- Within transaction state

  When used in transaction state, the SELECT statement locks the selected entities against access by other users. It enables a protected read of the data. Because the query is closed automatically with the END-TRANSACTION statement, all corresponding RETRIEVE statements must be done in transaction state.

- Outside of transaction state

  When the SELECT statement is used outside of transaction state, no entities are locked. Retrieval can then be done either inside or outside of transaction state, but note that no records are locked. This option enables several users to access the entities concurrently, but there is the risk of the database changing while the users are accessing the data.

  If selection is done outside of transaction state and retrieval is inside, the query is still open when the program leaves transaction state. When a selection is made to an open query, an implicit discard is done before the SELECT statement is executed, and an execution time warning is given.

See Also

- For more information on selection expressions, refer to "Using Functions and Expressions" earlier in this section.
- For more information on transaction state, refer to "Using Transactions" earlier in this section.

SELECT Statement

The SELECT statement selects a set of entities from a database and associates it with the query.

General Format

The general format of the SELECT statement is as follows:

```
SELECT query-name-1 FROM data-name-1 [CALLED data-name-2]
[, data-name-3 [CALLED data-name-4]]...
[DISTINCT]
[(mapping-clause-1)]
[ORDER BY [{ASCENDING  }]] [[{BINARY   }]] DM-expression-1
         [{DESCENDING}]    [{ORDERING }]
                           [{COLLATING}]
      [,[[{ASCENDING  }]] [[{BINARY   }]] DM-expression-2]...]
         [{DESCENDING}]    [{ORDERING }]
                           [{COLLATING}]
[WHERE selection-expression-1]
[ON EXCEPTION {imperative-statement-1   }]].
              {conditional-statement-1  }
              {NEXT SENTENCE            }
```

Explanation of Format Elements

query-name-1

This format element identifies the name of the query.

data-name-1 and data-name-3

These format elements identify the database classes.

FROM

This clause specifies the perspective through which SIM associates the attributes. The FROM clause is optional if you have declared the query using the OF phrase to specify a class. The FROM clause is required when you have declared the query using the 01-level clause to specify a query record.

CALLED data-name-2 and data-name-4

This option assigns a reference variable to a specific set of entities. Data-name-2 and data-name-4 are data names that are local to the SELECT statement.

DISTINCT

This option removes any duplicates, and selects only a unique set of data. DISTINCT is valid only for tabular formatting.

mapping-clause-1

There are several options available for the mapping clause. They are explained in "Explanation of Mapping Clause Format Options," which follows this discussion of the SELECT statement.

ORDER BY

This option is used to sort output before it is returned.

ASCENDING and DESCENDING

These options are used in ordering output in the ORDER BY phrase. ASCENDING is the default. If more than one sort key is indicated, the leftmost is the most significant for ordering.

BINARY, ORDERING and COLLATING

These options select the sequence for the retrieved data that is sorted in the ORDER BY option. The BINARY option is the default for the ASERIESNATIVE ccsversion and double-octet strings. The COLLATING option is the default for eight-bit strings that are not native to A Series systems.

The ordering sequence is the arrangement of members of a character set according to a predetermined scheme. Different elements can have the same ordering attribute.

The collating sequence is the arrangement of members of a character set according to the ordering sequence values (OSVs) and the priority sequence values (PSVs). Elements occupy different code positions. Elements that have the same OSV are differentiated by the PSV assigned to the code position.

DM-expression-1 and DM-expression-2

These format elements must be of a type that is ordered and must result in a single value unless tabular formatting is requested.

WHERE selection-expression-1

This option is used for a global selection expression that applies to the entire query.

ON EXCEPTION

The ON EXCEPTION option is described in "Handling SIM Exceptions" later in this section.

See Also

For information on declaring a query, see "Using the Query Declaration to Declare a Query."

Explanation of Mapping Clause Format Options

The following mapping clause options are used to select entities for retrieval with different types of formatting:

- For tabular formatting, do not use embedded SELECT clauses. The compiler requests tabular format if no embedded SELECT clauses are specified; otherwise, the compiler requests hybrid selection.
- For hybrid selection, which combines structured with tabular formats, use the Attribute Map clause for items to be displayed in a table, and use embedded SELECT clauses for items to be displayed in a structure.
- For structured formatting, use embedded SELECT clauses. Structured formatting is a subset of hybrid selection.

When the Attribute Map clause is used with the subset embedded SELECT clause, all attributes must be listed before any embedded SELECT clauses.

You do not have to specify attributes in the order in which they were declared, except when factoring for qualification. For an example of factoring, refer to the examples later in this section.

General Format of the Mapping Clause

The general format for the Mapping clause is as follows:

```
{ attribute-map-clause-1
{ embedded-select-clause-1
{ attribute-map-clause-2 , embedded-select-clause-2 }
```

General Format of the Attribute Map Clause

The format of the Attribute Map clause is as follows:

```
{ , { data-name-5 [= DM-expression-3]                                          } }...
{   { (attribute-map-clause-5) OF { Qualification Term clause                  } }
{                                 { Transitive Specification clause            } }
```

General Format of an Embedded SELECT clause

The format for an embedded SELECT clause is as follows:

```
{ , { SELECT query-name-2                                                                  } }
{   {       FROM { Qualification Term clause [OF Qualification Term clause]                } }
{   {            {      [OF Transitive Specification clause]...                            } }
{   {            { Transitive Specification clause                                         } }
{   {      [(mapping-statement-2)]                                                         } }
```

Explanation of Format Elements

data-name-5

If data-name-5 appears without DM-expression-3, the name must be a COBOL item name declared in the query declaration; the name must also be a database identifier that is one of the following: a data-valued attribute (single-valued or multivalued), an entity-valued attribute, or a database transitive specification. The compiler retrieves each matching attribute in the COBOL item. If the matching attribute is the name of a COBOL group item, it is also a compound attribute name. In this case, there must be a component attribute within the compound with the same name as each corresponding

elementary item within the group. The compiler maps all of the matching components into the elementary items.

If data-name-5 appears with DM-expression-3, data-name-5 must then be a COBOL item name declared in the query declaration and DM-expression-3 must be a DM-expression which can be a single attribute. If data-name-5 is a group item name, then DM-expression-3 must be a database compound attribute with the same name. Similarly, each elementary item in the group must have a corresponding component within the compound that has the same name. These matching components are retrieved into the elementary items. In all other cases, the name in DM-expression-3 does not have to match data-name-5. If DM-expression-3 contains a component of a compound attribute, the component must be qualified with the name of the compound attribute.

DM-expression-3

DM-expression-3 must be a data management expression that could, possibly, be the name of a single attribute.

When the name of an elementary item in the query declaration does not appear in an Attribute Map clause, the compiler assumes an implicit Attribute Map clause without DM-expression-3. For each elementary item, the class must contain an attribute with a matching name. The compiler then maps matching attributes in elementary items.

When the compiler maps an attribute into an elementary item, the rules for a COBOL MOVE statement apply. When a Boolean expression is mapped to its COBOL item, a false value is represented by 0 (zero) and a true value is represented by a nonzero value (not necessarily 1). The COBOL item should be tested to determine its value.

Qualification Term clause

This format element specifies the Qualification Term clause. The format for this clause is described in "Expression Formats" earlier in this section.

FROM

This phrase specifies the perspective through which SIM associates the attributes.

Transitive Specification clause

This clause describes the entity search level for transitive closure. The format for this clause is explained under "Expression Formats" earlier in this section.

query-name-2

This format element identifies the name of the query.

See Also

- For information on the use of the embedded SELECT clause with the TRANSITIVE function, refer to the *InfoExec SIM Programming Guide*.
- See Volume 1 for an explanation of the rules for the COBOL MOVE statement.
- See "Using Expressions" earlier in this section for information on DM expressions.

Examples

The following examples show Mapping clause options.

Example of Option Combinations in the Mapping Clause

The following example uses a combination of options shown in the Mapping clause:

```
MY-AGE = AGE,
MY-NAME = LAST-NAME OF NAME
SELECT Q-2 FROM DEPT-IN
     (MY-DEPT = DEPT-TITLE)
SELECT Q-3 FROM CURRENT-PROJECT
     (MY-PROJECT = PROJECT-TITLE).
```

Example of the Attribute Map Clause

The following example uses an Attribute Map clause:

```
MAN-AGE = AGE OF EMPLOYEE-MANAGER,
     (PROJECT-TITLE, PROJECT-NO) OF CURRENT-PROJECT.
```

Example of an Embedded SELECT Clause

The following example uses an embedded SELECT clause:

```
SELECT Q-2 FROM DEPT-IN
     (MY-DEPT = DEPT-TITLE)
SELECT Q-3 FROM CURRENT-PROJECT
     (MY-PROJECT = PROJECT-TITLE).
```

Example of Tabular Selection

The following example illustrates a tabular selection. Default attributes are obtained from the query record associated with DEPTQ.

```
SELECT DEPTQ FROM DEPARTMENT.
```

The following three examples use the SELECT statement for a query in three distinct versions of the Mapping clause.

Example of a Hybrid Selection Version

```
SELECT PEMPQ FROM PROJECT-EMPLOYEE
    ( PEMP-AGE = AGE,
      SELECT PROJQ FROM CURRENT-PROJECT
          ( P-NO = PROJECT-TITLE,
            MAN = LAST-NAME OF NAME OF PROJECT-MANAGER)
    )
    WHERE FIRST-NAME OF NAME = "John" AND
      LAST-NAME OF NAME = "Smith".
```

Example of a Fully Structured Version

```
SELECT PEMPQ FROM PROJECT-EMPLOYEE
     (PEMP-AGE = AGE,
     SELECT PROJQ FROM CURRENT-PROJECT
          (P-NO = PROJECT-TITLE,
          SELECT MANQ FROM PROJECT-MANAGER
               (PROG = LAST-NAME OF NAME)
          )
     ) WHERE FIRST-NAME OF NAME = "John" AND
          LAST-NAME OF NAME = "Smith".
```

Example of a Fully Tabular Version

```
SELECT PEMPQ FROM PROJECT-EMPLOYEE
     (PEMP-AGE = AGE,
      P-NO = PROJECT-TITLE OF CURRENT-PROJECT,
     MAN = LAST-NAME OF NAME OF PROJECT-MANAGER
          OF CURRENT-PROJECT
     ) WHERE FIRST-NAME OF NAME = "John" AND
          LAST-NAME OF NAME = "Smith".
```

Example of the CALLED Construct

The following example uses the CALLED construct. For more information, see "Special Constructs" earlier in this section.

```
SELECT MANQ FROM MANAGER
    ( MAN-NAME = LAST-NAME OF NAME,
      EMP-NAME1 = LAST-NAME OF NAME OF EMPLOYEES-MANAGING,
      EMP-NAME2 = LAST-NAME OF NAME OF EMPLOYEES-MANAGING
          CALLED OTHEREMP)
    WHERE ( AGE OF EMPLOYEES-MANAGING GREATER 24) AND
          ( AGE OF OTHEREMP LESS 36) AND
          ( DEPT-TITLE OF DEPT-IN OF EMPLOYEES-MANAGING =
            "Accounting") AND
          ( DEPT-TITLE OF DEPT-IN OF OTHEREMP = "Purchasing").
```

Example of Factoring for Qualification

The following example uses factoring for qualification:

```
SELECT PROJECTQ FROM PROJECT
    ( P-NAME = PROJECT-TITLE,
      ( P-MAN = LAST-NAME OF NAME, P-RANK = MANAGER-TITLE)
        OF PROJECT-MANAGER,
      ( S-NAME = LAST-NAME OF NAME, S-NO = PERSON-ID)
        OF STAFF-ASSIGNED OF ASSIGNMENT-HISTORY
        WITH RATING > 5Ø
    )
    WHERE PROJECT-NO = 456.
```

SET Statement

The SET statement alters the value for the level expected in a retrieval involving transitive closure. Refer to the *InfoExec SIM Programming Guide* for detailed information about transitive closure and about using entities in a tabular or structured format.

An entity-valued attribute that refers to the class of which it is an attribute is called a reflexive attribute. You can recursively access a reflexive attribute during a retrieval query, or use a circular path expression; this form of recursion is called transitive closure.

By default, a retrieval traverses the same level, then ends. The SET statement can be used to manipulate level changes during traversal. This use of the SET statement is especially valuable when the attribute is multivalued, with multiple values at each level.

General Format

The general format of the SET statement is as follows:

```
SET query-name-1 LEVEL { UP   }
                       { DOWN }
[ ON EXCEPTION { imperative-statement-1   } ].
               { conditional-statement-1  }
               { NEXT SENTENCE            }
```

Explanation of Format Elements

query-name-1

This format element designates the base of the query tree.

UP and DOWN

The UP phrase adjusts the level up one level from the base of the query tree. The DOWN phrase adjusts the level down one level. Set the appropriate level before a retrieval.

ON EXCEPTION

This option is described under "Handling SIM Exceptions."

Example

The following example uses the SET statement to adjust the level up from the base of the query tree, PQ:

```
SELECT PQ FROM PROJECT
     (PQ-TITLE = PROJECT-TITLE
      SELECT SQ FROM TRANSITIVE (SUB-PPROJECTS)
          (SQ-TITLE = PROJECT-TITLE)
      ) .
.
.
.
SET SQ LEVEL UP.
RETRIEVE SQ.
```

RETRIEVE Statement

The RETRIEVE statement requests information from a database. While the SELECT statement specifies and selects entities to be used, the RETRIEVE statement actually makes the entities available to your program.

Information about the options for a retrieval involving transitive closure is provided in "SET Statement" earlier in this section.

General Format

The general format for the RETRIEVE statement is as follows:

```
RETRIEVE query-name-1
[ ON EXCEPTION { imperative-statement-1 }
               { conditional-statement-1 } ].
               { NEXT SENTENCE           }
```

Explanation of Format Elements

query-name-1

Query-name-1 designates the name of the query. If query-name-1 is declared with a query record that contains an entity reference variable, the RETRIEVE statement must be used within transaction state, or a run-time error occurs.

ON EXCEPTION

This option is described under "Handling SIM Exceptions."

See Also

- For more information, refer to "Using the Query Declaration to Declare a Query" and "Declaring an Entity Reference Variable" earlier in this section.
- For information about transaction statements, see "Using Transactions" earlier in this section.

Example

The following line of code shows the use of the RETRIEVE statement:

```
RETRIEVE PEMPQ.
```

DISCARD Statement

The DISCARD statement discards, or terminates, a currently active query.

General Format

The general format of the DISCARD statement is as follows:

```
DISCARD query-name-1
[ ON EXCEPTION { imperative-statement-1 } ].
               { conditional-statement-1 }
               { NEXT SENTENCE           }
```

Explanation of Format Elements

query-name-1

This format element designates the currently active query.

ON EXCEPTION

This option is described under "Handling SIM Exceptions."

Example

The following example shows the use of the DISCARD statement:

```
DISCARD PROJQ.
```

Updating and Deleting Entities

The following statements are used to update and delete entities:

- The INSERT statement or clause is used to create an entity in a particular class with specified attribute assignments, or to create a new role or roles for an entity.
- The MODIFY statement or clause is used to update existing entities in a particular class using the specified attribute assignments.
- The Compound clause and the ASSIGN, INCLUDE, and EXCLUDE attribute assignment statements are used to add, update, or remove values from attributes.
- The DELETE statement is used to delete entities in a specified class from the database.

The statements and their syntax are explained later in this section.

There are two types of updates. You cannot use both types of updates in the same query. The updates include

- Single-statement update, which is a single statement that is executed when it is encountered
- Multiple-statement update, which enables you to intersperse attribute assignments among other program statements or to place assignments in other procedures or functions

A multiple-statement update uses the verbs START and APPLY before the update verb INSERT or MODIFY. START indicates the type of selection, if any, and associates a query name with the statement. APPLY applies all related multiple-statement update statements after the START statement and before the APPLY statement.

See Also

For more detailed information about the types of updates and their use, refer to the *InfoExec SIM Programming Guide.*

INSERT Statement

An INSERT statement creates an entity in the specified class with specific attribute assignments, or creates a new role or roles for an entity. There are two types of INSERT statements: single-statement update and multiple-statement update.

Single-Statement Update

General Format

The general format of the single-statement update INSERT statement is as follows:

```
INSERT data-name-1
[FROM data-name-2 WHERE selection-expression-1]
{assign-clause   } [,{assign-clause   }] ...
{exclude-clause  }   {exclude-clause  }
{include-clause  }   {include-clause  }
[ ON EXCEPTION {imperative-statement-1  } ].
               {conditional-statement-1 }
               {NEXT SENTENCE           }
```

Explanation of Format Elements

data-name-1 and FROM data-name-2

These format elements identify classes.

If the FROM clause is used, data-name-1 and data-name-2 must have either of the following two relationships. If data-name-1 and data-name-2 do not have either of these two relationships, an INSERT statement cannot be used.

- data-name-1 and data-name-2 must have a common superclass. For example, if the EMPLOYEE class and the EX_EMPLOYEE class both have the superclass PERSON, you can do either of the following:
  - Insert an entity from the EMPLOYEE class into the EX_EMPLOYEE class.
  - Insert an entity from the EX_EMPLOYEE class into the EMPLOYEE class.
- data-name-1 must be a subclass of data-name-2. For example, if the EMPLOYEE class is a subclass of the PERSON class then the following is true:
  - You can insert an entity from the PERSON class into the EMPLOYEE class.
  - You cannot insert an entity from the EMPLOYEE class into the PERSON class.

FROM and WHERE

These options specify an existing entity for which a new role is created.

Clauses

The clauses represent the database attribute assignment clauses. Refer to "Attribute Assignment Statements" later in this section.

ON EXCEPTION

This option is described under "Handling SIM Exceptions."

Examples

The following examples use the INSERT statement.

Example of the INSERT Statement with the ASSIGN and INCLUDE Clauses

The following example shows the INSERT statement with the ASSIGN and INCLUDE clauses used to represent the database attribute assignment clauses:

```
INSERT PROJECT-EMPLOYEE
    ASSIGN ["John" TO FIRST-NAME, "Doe" TO LAST-NAME]
       TO NAME
    ASSIGN 19 TO AGE
    INCLUDE PROJECT WITH PROJECT-TITLE = "Payroll"
       INTO CURRENT-PROJECT.
```

Example of the INSERT Statement with FROM and WHERE Options

The following example uses the INSERT statement with the FROM and WHERE options:

```
INSERT MANAGER FROM PERSON WHERE
    FIRST-NAME OF NAME = "Mary" AND
      LAST-NAME OF NAME = "Doe"
    ASSIGN 1000.00 TO BONUS
    ASSIGN INTERIM-MANAGER TO TEMP-STATUS.
```

Multiple-Statement Update

In a multiple-statement update, the START INSERT statement describes the type of selection, if any, and associates a query name with the statement. The APPLY INSERT statement applies all multiple-statement update INSERT statements between the START and APPLY statements. An example of both statements is provided at the end of the description of the APPLY INSERT statement later in this section.

START INSERT Statement

The START INSERT statement causes the database system to prepare for a multiple-statement update insertion.

General Format

The general format of the START INSERT statement is as follows:

```
START INSERT query-name-1
[FROM data-name-1 WHERE selection-expression-1]
[ ON EXCEPTION { imperative-statement-1 }
               { conditional-statement-1 } ].
               { NEXT SENTENCE           }
```

Explanation of Format Elements

query-name-1

This format element designates the query name to be associated with the statement.

FROM and WHERE

These options specify an existing entity for which a new role is created.

data-name-1

In general, you can use a subclass in place of a class; however, in a START INSERT statement using the FROM option, data-name-1 must be a database class.

ON EXCEPTION

This option is described under "Handling SIM Exceptions."

APPLY INSERT Statement

The APPLY INSERT statement causes the database system to perform the multiple-statement update insertion.

General Format

The general format of the APPLY INSERT statement is as follows:

```
APPLY INSERT query-name-1
[ ON EXCEPTION { imperative-statement-1 }
               { conditional-statement-1 } ].
               { NEXT SENTENCE           }
```

Explanation of Format Elements

query-name-1

This format element associates the query name with the statement.

ON EXCEPTION

This option is described under "Handling SIM Exceptions."

Example

The following example shows a multiple-statement update:

```
START INSERT P-Q.
ASSIGN MY-PROJECT-NO TO PROJECT-NO OF P-Q.
ASSIGN MY-PROJECT-TITLE TO PROJECT-TITLE OF P-Q.
INCLUDE PROJECT WITH PROJECT-NO EQUAL SUBPROJ-NUM AND
        PROJECT-TITLE EQUAL SUBPROJ-NAME
    INTO SUB-PROJECTS OF P-Q.
APPLY INSERT P-Q.
```

MODIFY Statement

A MODIFY statement causes the update of existing entities in the specified class through the accompanying attribute assignments. There are two types of MODIFY statements: single-statement update and multiple-statement update.

Single-Statement Update

General Format

The general format of the single-statement update MODIFY statement is as follows:

```
MODIFY [(LIMIT = arithmetic-expression-1 | NO LIMIT)] data-name-1
{assign-clause | exclude-clause | include-clause}
[, {assign-clause | exclude-clause | include-clause}] ...
[WHERE selection-expression-1]
[ON EXCEPTION {imperative-statement-1 | conditional-statement-1 | NEXT SENTENCE}].
```

Explanation of Format Elements

LIMIT or NO LIMIT

These options limit the number of entities that can be modified. If the limit is exceeded, an exception is produced, and no entities are updated.

arithmetic-expression-1

For details on arithmetic expressions, see "Arithmetic Expressions" earlier in this section.

data-name-1

This format element must be a class.

Clauses

The clauses refer to statements used to assign values to attributes. Refer to "Attribute Assignment Statements" later in this section for more information.

WHERE selection-expression-1

This option is a global selection expression that applies to the entire query. It is required for all attributes except class attributes. It is not allowed for class attributes.

ON EXCEPTION

This option is described under "Handling SIM Exceptions."

Examples

The following examples illustrate the use of the MODIFY statement.

Example of the MODIFY Statement with a NO LIMIT Option

The following example uses the MODIFY statement with the NO LIMIT option:

```
MODIFY (NO LIMIT) PROJECT-EMPLOYEE
    ASSIGN EXEMPT TO STATUS
    WHERE EMPLOYEE-HIRE-DATE LESS DATE-LIMIT.
```

Example of the MODIFY Statement with a LIMIT Option

The following example uses the MODIFY statement with a LIMIT option:

```
MODIFY (LIMIT = MAN-COUNT) MANAGER
     ASSIGN BONUS * 2 TO BONUS
     WHERE MANAGER-TITLE = EXECUTIVE.
```

Example of the MODIFY Statement with a Database Attribute Assignment

The following example uses the MODIFY statement with the INCLUDE, EXCLUDE, and ASSIGN database attribute assignments. The WHERE option is also used.

```
MODIFY PROJECT-EMPLOYEE
    INCLUDE PROJECT WITH PROJECT-TITLE =
        "Supplies Inventory" AND
        PROJECT-NO = 101
        INTO CURRENT-PROJECT
    EXCLUDE PROJECT WITH PROJECT-TITLE = "Office Inventory" AND
        PROJECT-NO = 58
        FROM CURRENT-PROJECT
    ASSIGN MANAGER WITH CURRENT OF MQ = MANAGER
        TO EMPLOYEE-MANAGER
    WHERE FIRST-NAME OF NAME
        OF PROJECT-EMPLOYEE = "John" AND
        LAST-NAME OF NAME OF PROJECT-EMPLOYEE = "Doe".
```

Multiple-Statement Update

A multiple-statement update uses the START MODIFY and APPLY MODIFY statements.

START MODIFY Statement

The START MODIFY statement causes the database system to prepare for a multiple-statement update modification.

General Format

The general format of the START MODIFY statement is as follows:

```
START MODIFY [(LIMIT = arithmetic-expression-1 | NO LIMIT)] query-name-1
[WHERE selection-expression-1]
[ON EXCEPTION imperative-statement-1 | conditional-statement-1 | NEXT SENTENCE].
```

Explanation of Format Elements

LIMIT or NO LIMIT

These options limit the number of entities that can be modified. If the limit is exceeded, an exception is produced.

arithmetic-expression-1

For details on arithmetic expressions, see "Arithmetic Expressions" earlier in this section.

query-name-1

This format element associates the query name with the statement.

WHERE selection-expression-1

This clause is a global selection expression that applies to the entire query. It is required for all attributes except class attributes. It is not allowed for class attributes.

ON EXCEPTION

This option is described under "Handling SIM Exceptions."

APPLY MODIFY Statement

The APPLY MODIFY statement causes the database system to perform the multiple-statement update modification. This update modification includes all multiple-statement update assignments that have occurred since the START MODIFY statement.

General Format

The general format of the APPLY MODIFY statement is as follows:

```
APPLY MODIFY query-name-1
[ ON EXCEPTION { imperative-statement-1   } ].
               { conditional-statement-1  }
               { NEXT SENTENCE            }
```

Explanation of Format Elements

query-name-1

This format element associates the query name with the statement.

ON EXCEPTION

This option is described under "Handling SIM Exceptions."

Examples of the START MODIFY and APPLY MODIFY Statements

The following are examples of the START MODIFY and APPLY MODIFY statements.

Example 1

```
START MODIFY PEMP-Q  WHERE FIRST-NAME OF NAME = "John"
    AND LAST-NAME OF NAME = "Smith".
IF DEPT-CHANGE = "YES" THEN
    ASSIGN DEPARTMENT WITH DEPT-TITLE = MY-DEPT-NAME
        TO DEPT-IN OF PEMP-Q
ELSE
    ASSIGN SENIOR TO TITLE OF PEMP-Q.
APPLY MODIFY PEMP-Q.
```

Example 2

```
START MODIFY MANUPDATE WHERE CURRENT OF MANRETR = MANAGER.
IF GETTING-A-PROMOTION = "YES" THEN
    ASSIGN EMPLOYEE-SALARY * 1.1
          TO EMPLOYEE-SALARY OF MANUPDATE
ELSE
    ASSIGN EMPLOYEE-SALARY * 1.Ø6
          TO EMPLOYEE-SALARY OF MANUPDATE
APPLY MODIFY MANUPDATE.
```

Attribute Assignment Statements

Attribute assignment statements add or remove values from attributes. Database attributes are assigned values within a single-statement update INSERT or MODIFY statement or between multiple-statement update START and APPLY statements. The ASSIGN, EXCLUDE, and INCLUDE attribute assignment statements are used to assign database attribute values as follows:

- As clauses in a single-statement update statement
- As statements in a multiple-statement update statement

The following table explains the purposes of the statements or clauses:

Statement or Clause	Explanation
ASSIGN	Updates single-valued attributes
EXCLUDE	Removes values from either single- or multivalued attributes
INCLUDE	Adds values to multivalued attributes

The Compound clause assigns or adds values to compound attributes. This is an optional clause that is used with the ASSIGN and INCLUDE statements.

The statements and clauses are presented alphabetically on the following pages.

ASSIGN Clause and Statement

The ASSIGN clause or statement is used to update single-valued database attributes.

General Format of the ASSIGN Clause

The format of the ASSIGN clause is as follows:

```
ASSIGN { DM-expression-1   } TO data-name-1
       { compound-clause-1 }
```

General Format of the ASSIGN Statement

The format of the ASSIGN statement is as follows:

```
assign-clause-1 OF query-name-1
```

Explanation of Format Elements

DM-expression-1

If DM-expression-1 has the following form and data-name-1 is entity-valued, then data-name-2 must be the name of a database class:

```
data-name-2 WITH Boolean-expression-clause-1
```

compound-clause-1

For an explanation of compound-clause-1, see "Compound Clause" after this discussion of the ASSIGN statement and clause.

TO data-name-1

This format element must be an immediate single-valued attribute.

assign-clause-1

This format element identifies the assign clause.

query-name-1

This format element associates the query name with the statement.

See Also

For information on expressions, refer to "Using Expressions" earlier in this section.

Examples

The following examples show the use of the ASSIGN clause and the ASSIGN statement with simple attributes:

```
ASSIGN BDATE TO BIRTHDATE OF PEMP.

ASSIGN CURRENT OF MAN-ERV TO EMPLOYEE-MANAGER OF PEMP.
```

The following example uses the ASSIGN clause with a Boolean expression:

```
ASSIGN MANAGER WITH FIRST-NAME OF NAME = "John" AND
     LAST-NAME OF NAME = "Smith"
     TO EMPLOYEE-MANAGER OF PEMP.
```

Compound Clause

The Compound clause is used to assign or add values to compound attributes. It is used only with the ASSIGN or INCLUDE attribute assignment statements and clauses.

General Format

The format of the Compound clause is as follows:

```
[ { DM-expression-1 TO data-name-1                    }
  { EXCLUDE data-name-2                               }
  { compound-clause-1 TO data-name-3                  }
[ , { DM-expression-2 TO data-name-4                  } ] ...]
  { EXCLUDE data-name-5                               }
  { compound-clause-2 TO data-name-6                  }
```

Explanation of Format Elements

data-name-1 through 6

These format elements are components of compound attributes.

DM-expression-1 and DM-expression-2

For information on expressions, see "Using Expressions" earlier in this section.

EXCLUDE

This phrase excludes a part of the compound attribute. For example, in a compound attribute consisting of first, middle, and last names, you could exclude middle names. You can use the EXCLUDE phrase on all data-valued and entity-valued attributes, both single-valued and multivalued. The phrase cannot be used with an INSERT or START INSERT statement.

compound-clause-1 and compound-clause-2

These format elements are compound clauses.

Example of the ASSIGN Statement with a Compound Clause

The following example uses the ASSIGN statement with the Compound clause for compound attributes:

```
ASSIGN
 [ "1234 Main" TO STREET, "Santa Ana" TO CITY ]
    TO CURRENT-RESIDENCE OF PEMPQ.
ASSIGN
 [ "23 Broadway" TO STREET, EXCLUDE STATE ]
    TO CURRENT-RESIDENCE OF MANQ.
```

Example of the INCLUDE Statement with a Compound Clause

The following example uses the INCLUDE statement with the Compound clause for compound attributes:

```
INCLUDE  [["Jimmy" TO FIRST-NAME "Jones"
             TO LAST-NAME] TO NAME OF CHILD
          ["Sally" TO FIRST-NAME "Jones"
             TO LAST-NAME] TO NAME OF CHILD] INTO
             CHILD OF PEMPQ.
```

EXCLUDE Clause or Statement

The EXCLUDE clause or statement is used to remove values from either single-valued or multivalued attributes.

General Format of the EXCLUDE Clause

The general format of the EXCLUDE clause is as follows:

```
EXCLUDE [(LIMIT = arithmetic-expression-1 | NO LIMIT)]
[DM-expression-1] FROM data-name-1
```

General Format of the EXCLUDE Statement

The general format of the EXCLUDE statement is as follows:

```
exclude-clause-1 OF query-name-1
```

Explanation of Format Elements

LIMIT or NO LIMIT

These options limit the number of entities that can be excluded. If neither of these options is specified, the default is one. If the limit is exceeded, an exception is produced.

arithmetic-expression-1

For information on arithmetic expressions, see "Arithmetic Expressions" earlier in this section.

DM-expression-1

If DM-expression-1 has the following form and data-name-1 is entity-valued, then data-name-2 must be the name of an entity-valued attribute (EVA):

```
data-name-2 WITH Boolean-expression-clause-1
```

FROM

This phrase specifies the perspective through which SIM associates the attributes. The omission of the clause containing the DM-expression-1 removes all values from the attribute for the entity (in other words, the attribute value does not exist).

data-name-1

This format element can be an immediate multivalued or single-valued attribute.

exclude-clause-1

This format element designates the use of the EXCLUDE clause in the statement.

query-name-1

This format element associates the query name with the statement.

See Also

For information on expressions, see "Using Expressions" earlier in this section.

Examples

The following are examples of the EXCLUDE clause and statement.

The following example uses the EXCLUDE clause with DM-expression-1. For each project employee with a current project of PROJECT-ERV, the information about the current project is removed.

```
EXCLUDE CURRENT-PROJECT WITH PROJECT = PROJECT-ERV
    FROM CURRENT-PROJECT OF PEMPQ.
```

The following example shows the removal of information about education from the record containing the educational qualifications of each employee (PEQ) with a BS degree:

```
EXCLUDE (NO LIMIT) EDUCATION WITH DEGREE-OBTAINED = BS
    FROM EDUCATION OF PEQ.
```

INCLUDE Clause or Statement

The INCLUDE clause or statement is used to add values to multivalued attributes (MVAs).

General Format of the INCLUDE Clause

The general format of the INCLUDE clause is as follows:

```
INCLUDE
{DM-expression-1 [, DM-expression-2]...     }
{compound-clause-1 [, compound-clause-2]... }
INTO data-name-1
```

General Format of the INCLUDE Statement

The general format of the INCLUDE statement is as follows:

```
include-clause-1 OF query-name-1
```

Explanation of Format Elements

DM-expression-1 and DM-expression-2

If DM-expression-1 and DM-expression-2 have the following form and data-name-1 is entity-valued, then data-name-2 must be the name of a database class:

```
data-name-2 WITH Boolean-expression-clause-1
```

compound-clause-1 and compound-clause-2

For information on compound-clause-1 and compound-clause-2, see "Compound Clause" earlier in this section.

INTO data-name-1

This format element must be an immediate multivalued attribute.

include-clause-1

This format element associates the INCLUDE clause with the statement.

OF query-name-1

This format element associates the query name with the statement.

See Also

Expressions are described under "Using Expressions."

Examples

The following examples use the INCLUDE statement.

The following example illustrates the use of the INCLUDE statement with simple attributes. Each project employee is assigned a current project.

```
INCLUDE PROJECT WITH PROJECT = CURRENT OF PROJQ
      INTO CURRENT-PROJECT OF PEMPQ.
```

The following example uses the INCLUDE statement with compound attributes. Educational qualifications are assigned to each project employee.

```
INCLUDE [PE-DEGREE TO DEGREE-OBTAINED,
         PE-GRAD-DATE TO YEAR-OBTAINED,
         PE-GPA TO GPA]
      INTO EDUCATION OF PEQ.
```

DELETE Statement

The DELETE statement causes the entities in the specified class that meet the specified selection to be deleted from the database.

General Format

The general format of the DELETE statement is as follows:

```
DELETE [(LIMIT = arithmetic-expression-1 | NO LIMIT)] data-name-1
WHERE selection-expression-1
[ ON EXCEPTION imperative-statement-1 | conditional-statement-1 | NEXT SENTENCE ].
```

Explanation of Format Elements

LIMIT or NO LIMIT

These options indicate the number of entities deleted. With the NO LIMIT option, all entities designated by selection expressions will be deleted. If neither of these options is specified, the default is one. An exception is produced if the limit is exceeded.

arithmetic-expression-1

Arithmetic expressions are explained in "Arithmetic Expressions" earlier in this section.

data-name-1

This format element must be a database class.

WHERE selection-expression-1

This clause is a global selection expression that applies to the entire query.

ON EXCEPTION

This option is described under "Handling SIM Exceptions" later in this section.

Examples

The following are examples of the DELETE statement.

This example shows the DELETE statement with a WHERE clause. The name Fred Jones is removed from the list of project employees. Note that his name still exists as an EMPLOYEE and as a PERSON in the database.

```
DELETE PROJECT-EMPLOYEE WHERE
   FIRST-NAME OF NAME = "Fred" AND
   LAST-NAME OF NAME = "Jones".
```

The following example shows the DELETE statement with a WHERE clause that contains a CURRENT function. All managers are removed who fit the condition specified by MANAGER. Note that a deleted manager still exists in the EMPLOYEE and PERSON classes.

```
DELETE MANAGER WHERE CURRENT OF MANQ = MANAGER.
```

The following example shows the DELETE statement with a LIMIT option. The statement deletes all projects with a project number less than 100. If there are more than 5 projects with project numbers less than 100, no projects are removed.

```
DELETE (LIMIT = 5) PROJECT WHERE PROJECT-NO BEFORE 100.
```

The following example shows the DELETE statement with a NO LIMIT option. The statement deletes all managers. The names of the managers still exist in the PERSON and EMPLOYEE classes.

```
DELETE (NO LIMIT) MANAGER WHERE TRUE.
```

Processing SIM Exceptions

When data manipulation statements are executed, an exception condition can occur if the program encounters a fault or does not produce the expected action.

DMSTATE is a database status word that is associated with each COBOL74 program that accesses a SIM database. The value of DMSTATE indicates whether an exception has occurred and specifies the nature of the exception. You can also obtain additional or specific information about exceptions by using the CALL SYSTEM statement.

The following pages explain the DMSTATE database status word, the CALL SYSTEM statement, and methods for handling exceptions.

For information on exception category values and mnemonics, refer to the *InfoExec SIM Programming Guide*.

DMSTATE Statement

The value of the DMSTATE database status word is set by the system at the completion of each data management statement. This value indicates whether an exception has occurred and the nature of the exception.

If you do not specify a field for the DMSTATE database status word, a value of TRUE is returned if an error has occurred.

General Format for the DMSTATE Statement

The general format for accessing the DMSTATE database status word follows:

```
[ { DMEXCEPTION     } ]
[ { DMSUBEXCEPTION  } ] OF DMSTATE
[ { DMMOREEXCEPTION } ]
[ { DMUPDATECOUNT   } ]
```

General Format for Accessing the DMEXCEPTION Category Mnemonics

The general format for accessing DMEXCEPTION category mnemonics is as follows:

```
VALUE category-mnemonic
```

Explanation of Format Elements

DMEXCEPTION

This option yields a numeric value identifying a major error category involved in the exception.

DMEXCEPTION VALUE category-mnemonic

DMEXCEPTION can be compared to major error category mnemonics with the VALUE phrase.

DMSUBEXCEPTION

This option yields a numeric value that identifies the subcategory of the major error category involved in the exception.

DMMOREEXCEPTION

This option yields a TRUE value if there are multiple errors.

DMUPDATECOUNT

This option yields a numeric value that identifies the number of entities updated. When no entities are updated, the value of DMUPDATECOUNT is 0 (zero).

CALL SYSTEM Statement

When an exception occurs, you can obtain text that describes the current exception, as well as additional information about the specific exception and about multiple exceptions. To do this, use the system names DMEXCEPTIONMSG, DMNEXTEXCEPTION, and DMEXCEPTIONINFO in the CALL SYSTEM statement.

General Format

The general format of the CALL SYSTEM statement is as follows:

```
CALL SYSTEM { DMEXCEPTIONMSG GIVING data-name-1 }
            { DMNEXTEXCEPTION                   }
            { DMEXCEPTIONINFO                   } .
```

Explanation of Format Elements

DMEXCEPTIONMSG

This system name causes the system text associated with the current exception to be moved into data-name-1.

data-name-1

This format element is the required parameter for the DMEXCEPTIONMSG system name and must be a PIC X DISPLAY item of at least 156 characters. The message returned is formatted in two 78-character lines.

DMNEXTEXCEPTION

This system name causes the next exception in a multiple error list to become the current exception.

DMEXCEPTIONINFO

This system name updates the DMEXCEPTIONINFO structure with additional information concerning the present exception. (The DMEXCEPTIONINFO structure is built into the program by the compiler.) You access the structure as a group item and access the subfields of the structure for specific information. For a list of the fields of the structure and an explanation of the information they contain, refer to the *InfoExec SIM Programming Guide*.

Example of Exception Handling in SIM

The following example shows code that can be used for handling exceptions:

```
RETRIEVE MAN-QUERY.
IF DMSTATE
    CALL SYSTEM DMEXCEPTIONMSG GIVING OUT-TEXT
    DISPLAY OUT-TEXT
    CALL SYSTEM DMEXCEPTIONINFO
    IF DMMOREEXCEPTIONS OF DMSTATE
        CALL SYSTEM DMNEXTEXCEPTION
        CALL SYSTEM DMEXCEPTIONMSG GIVING OUT-TEXT
        DISPLAY OUT-TEXT.
```

Example of the CALL SYSTEM Statement with the DMEXCEPTIONINFO Structure

The following example shows code that can be used to access the DMEXCEPTIONINFO structure and fields in COBOL74:

```
MOVE DMEXCEPTIONINFO
    TO WS-A.
DISPLAY DMSTRUCTURENAME
    OF DMEXCEPTIONINFO.
```

Handling SIM Exceptions

Any of the following methods can be used in a program to handle exceptions:

- Specifying the ON EXCEPTION option with the statement.
- Calling the DMERROR Use procedure.
- Using neither of the above. In this case, the program terminates if there is an exception.

The ON EXCEPTION option and the DMERROR Use procedure are described in the following paragraphs. For general information on the USE statement, refer to Volume 1.

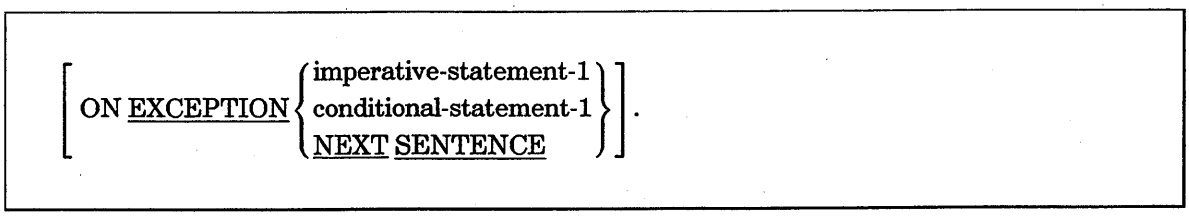
ON EXCEPTION Option

The ON EXCEPTION option can be placed after particular data management statements to handle exceptions. These statements include the following:

ABORT-TRANSACTION	INSERT
APPLY INSERT	MODIFY
APPLY MODIFY	OPEN
BEGIN-TRANSACTION	RETRIEVE
CANCEL TRANSACTION POINT	SAVE TRANSACTION POINT
CLOSE	SELECT
DELETE	SET
DISCARD	START INSERT
END-TRANSACTION	START MODIFY

General Format

The general format of the ON EXCEPTION option is as follows:



Explanation of Format Elements

imperative-statement-1, conditional-statement-1, or NEXT SENTENCE

These format elements are executed if an exception occurs. Refer to Volume 1 for information about these statements.

Example

The following is an example of the ON EXCEPTION option:

```
RETRIEVE DEPT-Q
   ON EXCEPTION
      GO TO PARA-EXIT.
```

DMERROR Use Procedure

The DECLARATIVES portion of the PROCEDURE DIVISION provides the ability to specify a DMERROR Use procedure.

The DMERROR Use procedure is called each time an exception occurs during the execution of a data management statement, unless an ON EXCEPTION option is associated with the statement. Control is returned from the DMERROR Use procedure to the statement following the data management statement that encountered the exception.

The DMERROR Use procedure can appear by itself or in any order with other Use procedures in the DECLARATIVES SECTION. Only one DMERROR Use procedure can be declared in a COBOL74 program. The DMERROR Use procedure cannot contain GO TO statements that reference labels outside the procedure.

If both a DMERROR Use procedure and an ON EXCEPTION option are used, the ON EXCEPTION option takes precedence, and the DMERROR Use procedure is not executed.

Example

The following example shows the declaration of the DMERROR Use procedure:

```
DECLARATIVES.
DMERR-SECT SECTION.
     USE ON DMERROR.
DMERR-PARA.
     IF DMEXCEPTION OF DMSTATE = VALUE DMCOMPLETE
     THEN ...
END DECLARATIVES.
```

SIM Sample Programs

Included are three sample programs that exemplify features of the SIM program interface. The programs reflect an organization in which projects have been assigned to various employees.

SIM Program to Update Project Assignments for Employees

In the sample program in Example 8–1, you can add or drop a project for employees in the database. If you choose to drop a project, the program completes all related assignments by recalculating the ratings of project employees. The program then updates the overall ratings for those employees.

The code in this sample program shows how to use single-statement and multiple-statement updates, tabular retrieval, and the AVERAGE and CURRENT functions.

```
000100 ID DIVISION.
000200 ENVIRONMENT DIVISION.
000300 CONFIGURATION SECTION.
000400 SPECIAL-NAMES.  RESERVE SEMANTIC.
000500 DATA DIVISION.
000600 DATA-BASE SECTION.
000600 DB  PROJEMPDB = "ORGANIZATION"  VALUE OF DBKIND IS SEMANTIC.
000700 QD  PEMP-Q  OF PROJECT-EMPLOYEE.
000800 QD  ASSI-Q.
000900 01  ASSI-Q-REC.
001000     02   ASSI-START-DATE      PIC X(10).
001100
001200 WORKING-STORAGE SECTION.
001300 01  PROJECT-INDICATOR         PIC X(4).
001400     88     PROJ-ADD                        VALUE "ADD".
001500     88     PROJ-DROP                       VALUE "DROP".
001600 01  PROJ-NUM                  PIC 9(11)    BINARY.
001700 01  SS-NUM                    PIC 9(11)    BINARY.
001800 01  INPUT-RATING              PIC 9V9.
001900
002000 PROCEDURE DIVISION.
002100 BEGIN.
002200     OPEN UPDATE PROJEMPDB.
002300
002400*************************************************************************
002500* Input the project indicator, project number, and social               *
002600* security number.                                                      *
002700*************************************************************************
002800     BEGIN-TRANSACTION.
002900     START MODIFY PEMP-Q WHERE PERSON-ID  EQUAL SS-NUM.
003000     IF PROJ-ADD THEN
003100         INCLUDE PROJECT WITH PROJECT-NO EQUAL PROJ-NUM
003200           INTO CURRENT PROJECT OF PEMP-Q
003300     ELSE
003400         IF PROJ-DROP THEN
003500           SELECT ASSI-Q FROM ASSIGNMENT
003600               ( ASSI-START-DATE = START-DATE )
003700             WHERE
003800               PERSON-ID  OF STAFF-ASSIGNED = SS-NUM AND
003900                 PROJECT-NO OF PROJECT-OF = PROJ-NUM AND
004000                 NOT RATING EXISTS
004100           PERFORM PROCESS-AN-ASSIGNMENT THRU
004200                   PROCESS-AN-ASSIGNMENT-EXIT UNTIL DMSTATE
004300           ASSIGN DMFUNCTION AVG (RATING OF ASSIGNMENT-RECORD )
004400             TO OVERALL-RATING OF PEMP-Q
004500           EXCLUDE CURRENT-PROJECT WITH PROJECT-NO EQUAL PROJ-NUM
004600             FROM CURRENT-PROJECT OF PEMP-Q.
004700     APPLY MODIFY PEMP-Q.
004800     END-TRANSACTION.
004900     CLOSE PROJEMPDB.
005000     STOP RUN.
```

Example 8–1. Updating Project Assignments for Employees

```
005100
005200 PROCESS-AN-ASSIGNMENT.
005300     RETRIEVE ASSI-Q
005400       ON EXCEPTION GO TO PROCESS-AN-ASSIGNMENT-EXIT.
005500
005600*************************************************************************
005700* Display the start date and enter the input rating.                    *
005800*************************************************************************
005900     MODIFY ASSIGNMENT
006000       ASSIGN INPUT-RATING TO RATING
006100         WHERE ASSIGNMENT EQUAL CURRENT OF ASSI-Q.
006200
006300 PROCESS-AN-ASSIGNMENT-EXIT.
006400     EXIT.
```

Example 8–1. Updating Project Assignments for Employees (cont.)

SIM Program to Archive Assignments

Example 8–2 shows a sample program that cleans up the database by removing assignments that were completed at least five years ago. The assignments are stored on tape.

The sample program shows hybrid retrieval including program formats, and extended attributes in both tabular and structured form.

```
000100 ID DIVISION.
000200 ENVIRONMENT DIVISION.
000300 CONFIGURATION SECTION.
000400 SPECIAL-NAMES.  RESERVE SEMANTIC.
000500 DATA DIVISION.
000600 DATA-BASE SECTION.
000700 DB PROJEMPDB= "ORGANIZATION"  VALUE OF DBKIND IS SEMANTIC.
000800
000900 QD  PEQ.
001000 01  PEQ-REC.
001100     02  PEQ-NAME                PIC X(20).
001200     02  PERSON-ID               PIC 9(11)   BINARY.
001300     02  DEPT                    PIC X(20).
001400 QD  AQ.
001500 AQ-REC.
001600     02  AQ-START-DATE           PIC X(10).
001700     02  AQ-END-DATE             PIC X(10).
001800     02  AQ-EST-PERSON-HOURS     PIC 9V9.
001900     02  AQ-RATING               PIC 9(2)V9.
002000 WORKING-STORAGE SECTION.
002100 01  DEADLINE                    PIC X(8)    VALUE "01/01/85".
002200 01  OUT-TEXT                    PIC X(156).
002300
002400 PROCEDURE DIVISION.
002500 BEGIN.
```

Example 8–2. Archiving Project Assignments

```
002600     OPEN UPDATE PROJEMPDB.
002700     BEGIN-TRANSACTION.
002800     SELECT PEQ FROM PROJECT-EMPLOYEE
002900*************************************************************
003000* PERSON-ID  does not need to be specified in the mapping list  *
003100* because the name of field is the same as the name of the      *
003200* immediate attribute.                                          *
003300*************************************************************
003400     ( PEQ-NAME = LAST-NAME OF NAME,
003500
003600*************************************************************
003700* DEPT is retrieved in tabular form with the rest of the        *
003800* immediate attributes.                                         *
003900*************************************************************
004000       DEPT = DEPT-TITLE OF DEPT-IN,
004100       SELECT AQ FROM ASSIGNMENT-RECORD
004200*************************************************************
004300* The assignment attributes are retrieved as a substructure of  *
004400* PROJECT-EMPLOYEE.                                             *
004500*************************************************************
004600       ( AQ-START-DATE = START-DATE,
004700         AQ-END-DATE = END-DATE,
004800         AQ-EST-PERSON-HOURS = EST-PERSON-HOURS,
004900         AQ-RATING = RATING ) )
005000     WHERE SOME ( END-DATE OF ASSIGNMENT-RECORD ) BEFORE DEADLINE.
005100     PERFORM DO-EMPLOYEE THRU DO-EMPLOYEE-EXIT UNTIL DMSTATE.
005200     END-TRANSACTION.
005300     CLOSE PROJEMPDB.
005400     STOP RUN.
005500
005600 DO-EMPLOYEE.
005700     RETRIEVE PEQ
005800       ON EXCEPTION
005900           GO TO DO-EMPLOYEE-EXIT.
006000*************************************************************
006100* Write the employee information on tape.                       *
006200*************************************************************
006300     PERFORM DO-ASSIGNMENT THRU DO-ASSIGNMENT-EXIT UNTIL DMSTATE.
006400
006500 DO-EMPLOYEE-EXIT.
006600     EXIT.
006700
006800 DO-ASSIGNMENT.
006900     RETRIEVE AQ
007000       ON EXCEPTION
007100         IF DMEXCEPTION OF DMSTATE NOT = VALUE DMCOMPLETE THEN
007200            CALL SYSTEM DMEXCEPTIONMSG GIVING OUT-TEXT
007300            DISPLAY OUT-TEXT
007400            GO TO DO-ASSIGNMENT-EXIT
007500         ELSE
007600            GO TO DO-ASSIGNMENT-EXIT.
```

Example 8–2. Archiving Project Assignments (cont.)

```
007700*************************************************************
007800* Write the assignment information to tape.                 *
007900*************************************************************
008000     DELETE ASSIGNMENT WHERE ASSIGNMENT EQUAL CURRENT OF AQ.
008100
008200 DO-ASSIGNMENT-EXIT.
008300     EXIT.
```

Example 8–2. Archiving Project Assignments (cont.)

SIM Program to List Subprojects

The sample program in Example 8–3 lists the subprojects for a specific project. If a subproject includes additional subprojects, those are also included in the list.

The sample program shows how to use transitive closure and the related SET statements.

```
000100 ID DIVISION.
000200 ENVIRONMENT DIVISION.
000300 CONFIGURATION SECTION.
000400 SPECIAL-NAMES.  RESERVE SEMANTIC.
000500 DATA DIVISION.
000600 DATA-BASE SECTION.
000700 DB  PROJEMPDB = "ORGANIZATION"  VALUE OF DBKIND IS SEMANTIC.
000800 QD  PQ.
000900 01  PQ-REC.
001000     02  PQ-TITLE      PIC X(30).
001100 QD  SQ.
001200 01  SQ-REC.
001300     02  SQ-TITLE      PIC X(30).
001400 WORKING-STORAGE SECTION.
001500 01  MASTER-PROJECT    PIC X(30).
001600 01  KOUNTER           PIC 9(11)  BINARY VALUE 0.
001700 01  DONE-STATUS       PIC X(3).
001800     88  NOT-DONE                      VALUE "NO".
001900     88  ALL-DONE                      VALUE "YES".
002000
002100 PROCEDURE DIVISION.
002200 MAIN.
002300     OPEN PROJEMPDB.
002400*************************************************************
002500* Enter the MASTER-PROJECT.                                 *
002600*************************************************************
002700     SELECT PQ FROM PROJECT
002800       ( PQ-TITLE = PROJECT-TITLE
002900         SELECT SQ FROM TRANSITIVE( SUB-PROJECTS )
003000           ( SQ-TITLE = PROJECT-TITLE ))
003100       WHERE PROJECT-TITLE OF PROJECT EQUAL MASTER-PROJECT.
003200     RETRIEVE PQ
003300       ON EXCEPTION
```

Example 8–3. Listing Subprojects

```
003400          DISPLAY "NO SUCH PROJECT"
003500        GO TO MAIN-END.
003600      MOVE "NO" TO DONE-STATUS.
003700      PERFORM GET-SUBPROJECT UNTIL ALL-DONE.
003800
003900 MAIN-END.
004000      CLOSE PROJEMPDB.
004100      STOP RUN.
004200
004300 GET-SUBPROJECT.
004400 RETRIEVE SQ.
004500*************************************************************************
004600* Display SQ-TITLE.                                                     *
004700* Check the next level of subprojects.                                  *
004800*************************************************************************
004900      SET SQ LEVEL UP.
005000      ADD 1 TO KOUNTER.
005100      RETRIEVE SQ
005200        ON EXCEPTION
005300          IF DMEXCEPTION OF DMSTATE = VALUE DMCOMPLETE THEN
005400            PERFORM BACK-UP-LEVEL UNTIL ALL-DONE OR NOT DMSTATE
005500            GO TO GET-SUBPROJECT-EXIT
005600          ELSE
005700            MOVE "YES" TO DONE-STATUS
005800            GO TO GET-SUBPROJECT-EXIT.
005900
006000 GET-SUBPROJECT-EXIT.
006100 EXIT.
006200 BACK-UP-LEVEL.
006300*************************************************************************
006400* Check the previous level for more subprojects.                        *
006500*************************************************************************
006600      SET SQ LEVEL DOWN.
006700      SUBTRACT 1 FROM KOUNTER.
006800      IF KOUNTER EQUAL Ø THEN
006900*************************************************************************
007000* All subprojects are exhausted.                                        *
007100*************************************************************************
007200        MOVE "YES" TO DONE-STATUS
007300      ELSE
007400        RETRIEVE SQ
007500          ON EXCEPTION
007600            IF DMEXCEPTION OF DMSTATE NOT = VALUE DMCOMPLETE
007700              THEN MOVE "YES" TO DONE-STATUS.
```

Example 8–3. Listing Subprojects (cont.)

Appendix A
Reserved Words

The following is a list of the COBOL74 reserved words in alphabetical order. For general information about COBOL74 reserved words, see Volume 1.

A

ABORT-TRANSACTION
ACCEPT
ACCESS
ADD
ADVANCING
AFTER
ALARM
ALL
ALLOW
ALPHABETIC
ALSO
ALTER
ALTERNATE
AND
ARE
AREA
AREAS
ASCENDING
ASSIGN
AT
ATTACH
AUDIT
AUTHOR

B

BEFORE
BEGIN-TRANSACTION
BEGINNING
BINARY
BLANK
BLOCK
BOTTOM
BY

C

CALL
CANCEL
CAUSE
CD
CF
CH
CHANGE
CHARACTER
CHARACTERS
CLOCK-UNITS
CLOSE
CMP
COBOL
CODE
CODE-SET
COLLATING
COLUMN
COMMA
COMMANDKEYS
COMMUNICATION
COMP
COMPUTATIONAL
COMPUTE
CONFIGURATION
CONTAINS
CONTENT
CONTINUE
CONTROL
CONTROL-POINT
CONTROLS
COPY
COPY-NUMBER
CORR
CORRESPONDING
COUNT
CP
CREATE
CRUNCH
CURRENCY
CURRENT

DATA
DATE-COMPILED
DB
DEBUG-ITEM
DEBUG-SUB-1
DEBUGGING
DELETE
DEPENDING
DETACH
DISABLE
DISPLAY
DMERROR
DMTERMINATE
DS

D

DATA-BASE
DATE-WRITTEN
DE
DEBUG-LINE
DEBUG-SUB-2
DECIMAL-POINT
DELIMITED
DESCENDING
DETAIL
DISALLOW
DIVIDE
DMSTATUS
DOUBLE
DUPLICATES

DATE
DAY
DEBUG-CONTENTS
DEBUG-NAME
DEBUG-SUB-3
DECLARATIVES
DELIMITER
DESTINATION
DICTIONARY
DISK
DIVISION
DMSTRUCTURE
DOWN
DYNAMIC

E

EGI
ENABLE
END-TRANSACTION
EOP
ESI
EXCEPTION
EXTEND

ELSE
END
ENDING
EQUAL
EVENT
EXECUTE
EXTERNAL

EMI
END-OF-PAGE
ENVIRONMENT
ERROR
EVERY
EXIT

F

FALSE
FILE
FINAL
FOOTING
FORM-KEY
FUNCTION

FD
FILE-CONTROL
FIND
FOR
FREE

FIELD
FILLER
FIRST
FORM
FROM

G

GENERATE
GREATER

GIVING
GROUP

GO

H

HEADING
HIGH-VALUES

HERE

HIGH-VALUE

I

I-O
I-O-CONTROL
ID
IDENTIFICATION
IF
IN
INDEX
INDEXED
INDICATE
INITIAL
INITIALIZE
INITIATE
INPUT
INPUT-OUTPUT
INQUIRY
INSERT
INSPECT
INSTALLATION
INTERRUPT
INTO
INVALID
INVOKE
IS

J-K

JUST
JUSTIFIED
KEY

L

LABEL
LAST
LD
LEADING
LEFT
LENGTH
LESS
LIMIT
LIMITS
LINAGE
LINAGE-COUNTER
LINE
LINE-COUNTER
LINES
LINKAGE
LOCAL
LOCAL-STORAGE
LOCK
LOCKED
LOW-VALUE
LOW-VALUES
LOWER-BOUND
LOWER-BOUNDS

M

MEMORY
MERGE
MESSAGE
MID-TRANSACTION
MODE
MODULES
MOVE
MULTIPLE
MULTIPLY

N

NATIVE
NEGATIVE
NEXT
NO
NO-AUDIT
NONE
NOT
NULL
NUMBER
NUMERIC

O

OBJECT-COMPUTER
OC
OCCURS
ODT-INPUT-PRESENT
OF
OFF
OFFSET
OMITTED
ON
OPEN
OPTIONAL
OR
ORGANIZATION
OUTPUT
OVERFLOW
OWN

P

PAGE
PAGE-COUNTER
PC
PERFORM
PF
PH
PIC
PICTURE
PLUS
POINT
POINTER
PORT
POSITION
POSITIVE
PRINTING
PRIOR
PROCEDURE
PROCEDURES
PROCEED
PROCESS
PROGRAM
PROGRAM-ID
PURGE

Q

QUEUE
QUOTE
QUOTES

R

RANDOM
RD
READ
READ-OK
READ-WRITE
REAL
RECEIVE
RECEIVED
RECORD
RECORDS
RECREATE
REDEFINES
REEL
REF
REFERENCE
REFERENCES
RELATIVE
RELEASE
REMAINDER
REMOVAL
REMOVE
RENAMES
REPLACING
REPORT
REPORTING
REPORTS
RERUN
RESERVE
RESET
RETURN
REVERSED
REWIND
REWRITE
RF
RH
RIBBON
RIGHT
ROUNDED
RUN

S

SAME
SAVE
SD
SEARCH
SECTION
SECURE
SECURITY
SEEK
SEGMENT
SEGMENT-LIMIT
SELECT
SEND
SENTENCE
SEPARATE
SEQUENCE
SEQUENTIAL
SET
SIGN
SINGLE
SIZE
SORT
SORT-MERGE
SOURCE
SOURCE-COMPUTER
SPACE
SPACES
SPECIAL-NAMES
STACK
STANDARD
STANDARD-1
START
STATUS
STOP

continued

continued

	S	
STORE	STRING	SUB-QUEUE-1
SUB-QUEUE-2	SUB-QUEUE-3	SUBTRACT
SUM	SYMBOLIC	SYNC
SYNCHRONIZED	SYSTEM	SYSTEMERROR
	T	
TABLE	TAG-KEY	TAG-SEARCH
TALLYING	TAPE	TASK
TB	TERMINAL	TERMINATE
TEXT	THAN	THEN
THROUGH	THRU	TIME
TIMER	TIMES	TO
TODAYS-DATE	TODAYS-NAME	TOP
TRACTORS	TRAILING	TRANSACTION
TRANSPORT	TRUE	TYPE
	U	
UNIT	UNLOCK	UNSTRING
UNTIL	UP	UPDATE
UPON	USAGE	USE
USING		
	V	
VA	VALUE	VALUES
VARYING	VIA	
	W	
WAIT	WHEN	WHERE
WITH	WORDS	WORKING-STORAGE
WRITE	WRITE-OK	
	Z	
ZERO	ZEROES	ZEROS

Appendix B
User-Defined Words

A user-defined word is a COBOL74 word that must be supplied by the user to satisfy the format of a clause or statement. Each character of a user-defined word is selected from the set of characters A through Z, 0 through 9, and the hyphen (-). The hyphen cannot appear as the first or last character of a word.

A list of the words that you can define is shown below. For detailed information about user-defined words, refer to Volume 1.

alphabet-name
COMS-header-name
data-name
file-name
form-name
index-name
library-name
paragraph-name
record-name
routine-name
segment-number
cd-name
condition-name
family-name
formlibrary-name
group-list-name
level-number
mnemonic-name
program-name
report-name
section-name
text-name

Glossary

The glossary contains terms that appear in this volume and are important in understanding the program interfaces. For definitions of general Unisys Standard COBOL74 terms, refer to Volume 1. For definitions of product-specific terms, refer to the product programming guides.

A

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
In Data Management System II (DMSII), 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.

address
(1) The identification of a location in storage (memory). (2) A sequence of bits, a character, or a group of characters that identifies a network station or a group of stations, a user, or an application. (3) The location of a device in the system configuration. (4) The identification of the location of a disk sector.

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.

agenda
In the Communications Management System (COMS), an entity used for message routing that consists of a processing-item list and a destination. An agenda can be applied to messages that are received or sent by application programs.

aggregate functions
The functions applied to a set of entities to produce a single scalar result.

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.

alias identifier
An alternative label or name. For instance, a data item or entity name is invoked and given an alias identifier that is used to refer to the item or entity throughout the rest of the program. An alias identifier is useful when, for example, an entity name in a program is imported from a system or a language that has different restrictions on name formation.

alphanumeric character
Any character in the computer's character set.

ancestor
(1) The parent of a particular task, or the parent of any ancestor of the task. (2) In embedded data sets in the Data Management System II (DMSII) environment, the owner of a record, the owner of the owner, and so forth.

apply
In the Semantic Information Manager (SIM), to update data in the database. The changed data is not made permanent until an END-TRANSACTION statement is executed.

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 operator
A single character or a fixed 2-character combination belonging to the following set: + (addition), – (subtraction), * (multiplication), / (division), or ** (exponentiation).

ascending key
A key that has values on which data are ordered, starting with the lowest value of the key up to the highest value of the key, in accordance with the rules for comparing data items.

attribute
(1) The information that describes a characteristic of an entity. (2) In the Semantic Information Manager (SIM), a characteristic of the entities of a class or of the class itself. A SIM attribute can be either data-valued or entity-valued.

audited database
In Data Management System II (DMSII) and in the InfoExec environment, 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
In Data Management System II (DMSII), 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
In Data Management System II (DMSII), to undo changes made against a database and to roll back the progress of one or more transactions to a previously consistent state.

batch mode
(1) An execution mode in which a group of commands or other input is transmitted and processed by the computer with no user interaction. (2) In the Communications Management System (COMS), an execution mode in which a program running under COMS can do batch-type updates to a database shared by other transaction processors. (3) *Contrast with* interactive mode.

BCD
See binary-coded decimal.

BDMSCOBOL74
See Burroughs Data Management System COBOL74.

binary
A characteristic or condition for which there are two alternatives. A binary number system uses a base of 2 and the digits 0 and 1.

binary-coded decimal (BCD)
The decimal notation in which the decimal digits are represented by a binary numeral.

Binder
A program that enables separately compiled subprograms to be joined with a host object code file to produce a single object code file.

binding
(1) The process of combining one or more separately compiled subprogram object code files with a host object code file to produce a single object code file. This process is performed by the Binder program. (2) The process by which distinct occurrences of a name in a query are made to refer to the same instance of a reference variable during execution of the query.

bit
The most basic unit of computer information. The word *bit* is a contraction of *binary digit*. A bit can have one of two values: binary 0 (sometimes referred to as OFF) and binary 1 (sometimes referred to as ON).

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.

BNA

The network architecture used on A Series, B 1000, and V Series systems as well as CP9500 and CP 2000 communications processors to connect multiple, independent, compatible computer systems into a network for distributed processing and resource sharing.

Boolean

Pertaining to variables, data items, and attributes having a value of TRUE or FALSE.

Burroughs Data Management System COBOL74 (BDMSCOBOL74)

A Unisys language based on COBOL74 that contains extensions for accessing Data Management System II (DMSII) databases.

byte

On Unisys A Series systems, a measurable group of 8 consecutive bits having a single usage. In data communications, a byte is also referred to as a character or an octet.

C

CANDE

See Command and Edit.

character

(1) The actual or coded representation of a digit, letter, or special symbol in display form. (2) In data communications, 8 contiguous bits (1 byte).

character position

The amount of physical storage required to store a single standard data format character whose usage is described as DISPLAY. Further characteristics of the physical storage are defined by the implementor.

checksum

In Data Management System II (DMSII), a value used to detect certain classes of I/O errors. A checksum is computed for each database file block by applying an equivalence operator to each word in the block. When the block is physically written, the checksum value is stored in a checksum word appended to the end of the block. When the block is read, the checksum is recomputed and the result is compared to the stored value. A checksum error occurs if the two values are not equal.

class

In the Semantic Information Manager (SIM), a collection of entities of the same basic type.

class attribute

In the Semantic Information Manager (SIM), an attribute that describes a class as a whole, rather than one that describes the entities of a class.

clause

An ordered set of consecutive character-strings that specifies an attribute of an entry.

COBOL74
A version of the COBOL language that is compatible with the American National Standard X3.23-1974.

CODASYL
See Conference for Data Systems Language.

column
A character position in a print line. The column numbers are incremented by 1, starting from 1 at the leftmost character position of the print line and extending to the rightmost position of the print line.

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.

commit
In a Semantic Information Manager (SIM), to record a transaction permanently in a database and make the results visible to the other users of the database. This action generally occurs with an END-TRANSACTION statement.

communication device
A mechanism (hardware, or hardware and software) capable of sending data to a queue and/or receiving data from a queue. This mechanism can be a computer or a peripheral device. One or more programs containing communication description entries and residing within the same computer define one or more of these mechanisms.

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.

compiler
A computer program that translates instructions written in a source language, such as COBOL or ALGOL, into machine-executable object code.

compound attribute
In the Semantic Information Manager (SIM), a data-valued attribute that consists of elements, called components, that can be accessed individually. The attribute PHONE_NUMBER, for example, can be a compound attribute that contains the components AREA_CODE, PREFIX, and SUFFIX.

COMS
See Communications Management System.

COMS header
In Communications Management System (COMS), the part of the communication structure that contains routing and descriptive information about the message. There is an input header for input messages and an output header for output messages.

COMS interface
In a program interface for the Communications Management System (COMS), the interface that directs input and output and provides an optional conversation area for descriptive information.

concatenation
The linking of strings or items.

condition
(1) 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. (2) An expression used to limit the attribute values that are to appear in a report, or to limit the values that are to be updated.

Conference for Data Systems Language (CODASYL)
A committee composed of Department of Defense representatives, and computer users and manufacturers, who define standards in hardware and software tools for database management.

control file
In Data Management System II (DMSII), a file containing data file coordination information, audit control information, and dynamic database parameter values.

control item
(1) In Data Management System II (DMSII), a count item, population item, or record-type item. (2) In the transaction processing system (TPS), a system-defined item contained in a transaction record. A control item is maintained by TPS and is read-only in all programs written in Burroughs Data Management System COBOL (BDMSCOBOL), Burroughs Data Management System COBOL74 (BDMSCOBOL74), and Burroughs Data Management System ALGOL (BDMSALGOL). The initial value of a control item is assigned when a transaction record is created.

conversation area
In Communications Management System (COMS), the user data space in the header of a message. The conversation area is user defined and can contain information passed by a program or processing item. When used with a direct-window interface, this area contains the telephone number to be dialed.

count item
In Data Management System II (DMSII), a control item that contains a system-maintained count of the number of counted links that refer to a record.

current record
(1) The record that is available in the record area associated with a file. (2) In Data Management System II (DMSII), 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.

current record pointer
A conceptual entity used to select the next record.

D

DASDL
See Data and Structure Definition Language.

Data and Structure Definition Language (DASDL)
In Data Management System II (DMSII), 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 communications interface (DCI) library
A library that serves as the direct programmatic interface to the Communications Management System (COMS). Application programs must communicate with COMS through the DCI library to use agendas, processing items, routing by trancode, and synchronized recovery.

data dictionary
A repository of information about the definition, structure, and usage of data. The data dictionary does not contain the actual data.

data item
(1) In Data Management System II (DMSII), a field in a database record or transaction format that contains a particular type of information. (2) In COBOL, a character or a set of contiguous characters (excluding, in either case, literals) defined as a unit of data by the COBOL program.

data management
The operating system function of placing and retrieving data in storage and protecting its 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
In Data Management System II (DMSII), 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 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.

data-description entry
An entry in the DATA DIVISION that is composed of a level number followed by a data-name, if required, and a set of data clauses, as required.

data-valued attribute (DVA)
In the Semantic Information Manager (SIM), an attribute that contains data values for the entities of a class. *See also* entity-valued attribute.

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 equation
In Data Management System II (DMSII), 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 status
In Data Management System II (DMSII), the description of the success or failure of the most recent DMSII statement. The database status can access the DMSTATUS and DMSTRUCTURE statements. *See also* major category and subcategory.

DB
See database.

DBMS
See database management system.

DCI library
See data communications interface (DCI) library.

DCIENTRYPOINT
An entry point of the data communications interface (DCI) library. A compiler automatically generates code calling this entry point whenever an application program executes an ENABLE, RECEIVE, or SEND statement.

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.

default value
The value automatically given to a variable when no other value has been assigned.

descending key
A key that has values on which data are ordered, starting with the highest value of the key down to the lowest value of the key, in accordance with the rules for comparing data items.

description file
In Data Management System II (DMSII), 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.

designator
In Communication Management System (COMS), a binary number that is part of an internal code used in the table structure. By using designators in programs that run under COMS, the programmer can control messages symbolically rather than by communicating directly with entities in the data communications environment.

destination
(1) A device to which output is sent. (2) In COBOL, the symbolic identification of the receiver of a transmission from a queue.

direct window
In the Communications Management System (COMS), a type of window that enables the user to route messages directly to COMS, while using all the COMS capabilities for preprocessing and postprocessing of messages.

directory
A list of file names organized into a hierarchy according to similarities in their names. File names are grouped in a directory if their first name constants (and associated usercodes) are identical. These groups are divided into subdirectories consisting of those file names whose first two name constants are identical, and so on.

discontinue
(1) To terminate a referenced task. (2) To cause a process to terminate abnormally. A process can be discontinued by operator commands, by statements in related processes, or by the system software.

disjoint
In Data Management System II (DMSII), pertaining to a data set, set, or subset when it is not contained in another data set. *Contrast with* embedded.

DM
See data management.

DMSII
See Data Management System II.

DVA
See data-valued attribute.

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.

EGI
See end-of-group indicator.

elementary item
A data item described as not being further logically subdivided.

embedded
In Data Management System II (DMSII), 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.

EMI
See end-of-message indicator.

enabled
Referring to a station that is being polled (invited to transmit in a certain order) and that can communicate with the system.

end of job (EOJ)
(1) The termination of processing of a job. (2) In the Communications Management System (COMS), the control code that signals the receiver that a job has completed.

end of task (EOT)
The termination of processing of a task.

end-of-group indicator (EGI)
An option indicator that specifies the end of a group of data in a data communications message.

end-of-message indicator (EMI)
An option indicator that specifies the end of a data communications message.

end-of-segment indicator (ESI)
In data communications, an option indicator that specifies the end of a segment of data in a message.

entity
(1) An item about which information is stored. An entity can be tangible or intangible, and is further defined by attributes, which are the characteristics of the entity. (2) In the Communications Management System (COMS), a category of items within the configuration file. (3) Any object defined in the Advanced Data Dictionary System (ADDS). To ADDS, an entity can be a Screen Design Facility (SDF) field, form, or formlibrary; an attribute or class in a Semantic Information Manager (SIM) database; a data set, group, or item in a Data Management System II (DMSII) database; or the entire SIM or DMSII database. Note that the definitions that are stored in ADDS–objects and their relationships–are themselves known as entities. (4) In the Screen Design Facility (SDF), a field, form, or formlibrary about which information is stored.

entity reference variable
In Semantic Information Manager (SIM) programs, a variable that refers explicitly to an entity.

entity-valued attribute (EVA)
In the Semantic Information Manager (SIM), an attribute that links an entity to one or more other entities of the same or another class. Entity-valued attributes establish relationships between entities. *See also* data-valued attribute.

entry point
A procedure or function that is a library object.

EOJ
See end of job.

EOT
See end of task.

ESI
See end-of-segment indicator.

EVA
See entity-valued attribute.

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.

expression
A combination of operands and operators that results in the generation of one or more values.

extended attribute
In the Semantic Information Manager (SIM), an attribute referenced by means of an entity-valued attribute (EVA). An extended attribute of one class can be an immediate attribute of another class.

extension
A change in a language that enables it to perform an activity not previously supported by the language.

F

FD
See File Description (FD) entry.

field
(1) An area on a screen or form in which data is displayed or entered. The delimiters of the field can be visible or invisible to the terminal operator. (2) A consecutive group of bits within a word or a component of a record that represents a logical piece of data.

file
A named group of related records.

file attribute
An element that describes a characteristic of a file and provides information the system needs to handle the file. Examples of file attributes are the file title, record size, number of areas, and date of creation. For disk files, permanent file attribute values are stored in the disk file header.

File Description (FD) entry
An entry in the FILE SECTION of the DATA DIVISION that is composed of the level-indicator FD, a file-name, and a set of file clauses as required.

form
(1) A special screen containing prompts requesting information and empty form fields in which the requested information can be entered. (2) In the Screen Design Facility (SDF) and SDF Plus, an entity consisting of form attributes, a form image, associated fields, and field attributes. In SDF Plus, a form also contains processing logic. (3) In the Advanced Data Dictionary System (ADDS), the form image and related field attributes that the programmer formats and creates for applications.

form key
In the Screen Design Facility (SDF), a unique binary number that the compiler assigns to an SDF form. The form key is used when SDF is run with the Communications Management System (COMS).

form record
In the Screen Design Facility Plus (SDF Plus), an element of a form record library that represents records of data. Form records describe the format of messages used to output data from, or input data to, an SDF Plus form. *Synonym for* message type.

form record library
In the Screen Design Facility Plus (SDF Plus), a collection of form records and transaction types, and the interrelationship between them.

form record number
In the Screen Design Facility Plus (SDF Plus), the unique number by which SDF Plus internally references a form record. *Synonym for* message type number.

format
The specific arrangement of a set of data.

formlibrary
In the Screen Design Facility (SDF), an entity that contains one or more forms after generation and is invoked by an application program to format a form image and process the form for use in the program.

function
(1) An assigned purpose, activity, or significance. (2) A subroutine that returns a value.

G

global data item
In Data Management System II (DMSII), 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 to the entire database.

global selection expression
An expression that applies to the entire query.

group item
In Data Management System II (DMSII), 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.

header
A sequence of characters preceding the text of a message, containing routing or other communications-related information.

hybrid selection
In Semantic Information Manager (SIM) application programs, the process of requesting information using a combination of tabular selection and structured selection.

I

ID
See identifier.

I-O-CONTROL
The name of an ENVIRONMENT DIVISION paragraph in which the following are specified: object program requirements for specific I/O techniques, rerun points, sharing of same areas by several data files, and multiple file storage on a single I/O device.

I/O
Input/output. An operation in which the system reads data from or writes data to a file on a peripheral device such as a disk drive.

identifier (ID)
(1) A data-name followed by the required combination of qualifiers, subscripts, and indexes necessary to make unique reference to a data item. (2) One node of a file name.

immediate attribute
In the Semantic Information Manager (SIM), an attribute that is directly associated with a particular class.

indexed file
A file whose records are accessed by a key, which is a field in each record. An entry containing the key value and physical address of each record is stored in an index associated with the file. The index entries are ordered by key value. Access to an indexed file is either sequential or random.

InfoExec
Information Executive. The name of a family of Unisys products that define, maintain, retrieve, and update databases.

initial value
In the transaction processing system (TPS), the value assigned to an item in a newly created transaction record. An initial value can be declared for each item in the Transaction Formatting Language (TFL) declaration for the item; otherwise, TFL assigns the item a default initial value.

input procedure
In sorting, a group of statements executed before each record is released to be sorted.

integer
(1) A whole number. (2) In COBOL, a numeric literal or a numeric data item that does not include any character positions to the right of the assumed decimal point.

interactive mode
An execution mode in which each command or item of data is validated and executed at the time it is entered at a terminal or workstation, allowing the user to see immediate results and correct errors as they are made. *Contrast with* batch mode.

interface
(1) A common boundary at which independent systems or diverse groups interact. (2) To interact or coordinate smoothly. (3) A set of conventions for passing information.

InterPro
Interactive Productivity. A family of Unisys software facilities used to create new products and enhance existing products.

K

key
(1) A field used to locate or identify a record in an indexed file. (2) In COBOL, a data item that identifies the location of a record, or a group of data items that identifies the ordering of data.

key condition
In a selection expression, a condition that specifies the values used to locate specific records in a data set that is referenced by a specific set or subset. *See also* selection expression.

keyword
In programming languages, a reserved word that must be present when the format in which the word appears is used in a source program.

L

language interface
(1) The means of allowing a programming language to interact. (2) The protocols and extensions developed for a programming language and implemented in the compiler.

level-indicator
Two alphabetic characters (DB, CD, FD, LD, QD, RD, and SD) that identify a specific type of file or a position in a hierarchy.

library
A collection of one or more named routines or library objects that are stored in a file and can be accessed by other programs.

line
A row of text in a printout.

line number
An integer that denotes the vertical position of a report line on a page.

link
In Communications Management System (COMS), to join the application program to COMS.

link item
In Data Management System II (DMSII), 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.

local selection expression
In Semantic Information Manager (SIM) programs, a selection expression that applies only to a specific entity-valued attribute (EVA).

lock
To prevent access to particular data in the database by other users when one user is accessing it.

logical database
In Data Management System II (DMSII), 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 it are listed.

logical operator
An operator that corresponds to the logical (Boolean) operation of AND, OR, or NOT.

logical station number (LSN)
A unique number assigned to each station in a network and each pseudostation allocated by a message control system (MCS). Each station has an LSN assigned according to the order in which the stations are defined.

LSN
See logical station number.

M

major category
In Data Management System II (DMSII), a numeric value that occurs at the end of each data management statement to identify the type of error.

manual subset
In Data Management System II (DMSII), 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.

mapping
(1) A transformation from one set to another set. (2) A correspondence. (3) A description of the way in which different record types of a database are associated with one another.

master
In Data Management System II (DMSII), a data set or record that contains one or more embedded data sets or records. In DMSII, *synonym for* parent, and owner.

master record
See master.

MCS
See message control system.

member
In Data Management System II (DMSII), a record of a data set.

message
(1) Any combination of characters and symbols designed to communicate information from an originator to one or more destinations. (2) In data communications, any information-containing data unit, in an ordered format, sent by means of a communications process to a named network entity or interface. A message contains the information (text portion) and controls for routing and handling (header portion). (3) In COBOL, data associated with an end-of-message indicator or an end-of-group indicator.

message area
In the Communications Management System (COMS), an area of the communication structure in which the message is contained.

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.

message count
The number of complete messages in the designated message queue.

message key
In the Screen Design Facility (SDF), an alphanumeric transaction code that identifies the form for processing in an application program.

message segment
A subdivision of a message. A segment is normally associated with an end-of-segment indicator.

message type
Synonym for form record.

message type number
Synonym for form record number.

MFI
See module function index.

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.

module function index (MFI)
An integer value that represents transaction code or group of transaction codes that are used to route forms or or messages.

multiple-statement update
In Semantic Information Manager (SIM) application programs, an update query in which attribute statements are interspersed among other program statements. An update query can also be a single-statement update.

multivalued attribute (MVA)
In the Semantic Information Manager (SIM), an attribute that assumes several values for each entity. *Contrast with* single-valued attribute.

MVA
See multivalued attribute.

N

network support processor (NSP)
A data communications subsystem processor that controls the interface between a host system and the data communications peripherals. The NSP executes the code generated by the Network Definition Language II (NDLII) compiler for line control and editor procedures. An NSP can also control line support processors (LSPs).

next executable statement
The statement to which control is transferred after execution of the current statement is complete.

nonnumeric literal
A character string bounded by quotation marks ("). The string can include any character in the character set of the computer. To represent a single quotation mark character within a nonnumeric literal, two contiguous quotation marks must be used.

NSP
See network support processor.

null value
The value contained in an item that does not contain valid information.

numeric literal
A literal composed of one or more numeric characters that can also contain either a decimal point, an algebraic sign, or both. The decimal point must not be the rightmost character. The algebraic sign, if present, must be the leftmost character.

O

object time
In COBOL, the time during which an object program is executed. *Synonym for* run time, execution time.

operational sign
An algebraic sign, associated with a numeric data item or a numeric literal, that indicates whether the value is positive or negative.

ordered
Pertaining to an item maintained in a user-specified sequence.

P

partitioned structure
(1) In Data And Structure Definition Language (DASDL), a structure that can be declared and assigned one or more structure numbers corresponding to a data set, set, or subset. (2) In Data Management System II (DMSII), the structure used for the structure number function, DMSTRUCTURE, to analyze the results of exception conditions.

path
In Data Management System II (DMSII), a specific location within the logical ordering of a data set, set, subset, or access.

pattern matching
A string expression that generates a set of strings from a regular expression. A string expression can be tested for membership in the set by using an operator.

perspective class
The class from which a query is directed. Any additional classes that are involved in the query are viewed in relation to the perspective class.

phrase
An ordered set of one or more consecutive COBOL character-strings that forms a portion of a COBOL procedural statement or of a COBOL clause.

population
For disjoint data sets in Data Management System II (DMSII), 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.

postprocessing
The processing done to a message by processing items after an application program sends out the message.

preprocessing
The processing that the Agenda Processor performs on a message before an application program receives the message.

procedure
A paragraph or group of logically successive paragraphs, or a section or group of logically successive sections, within the PROCEDURE DIVISION.

processing item
A procedure, contained in a processing-item library, used for processing a message.

processing-item library
In the Communications Management System (COMS), a user-written ALGOL library containing a set of procedures called processing items. A processing-item library can be called only by the COMS Agenda Processor library to preprocess and postprocess messages as they are received and sent by programs.

product interface
The protocols that exist within a product to allow it to interact with other, specific products or programming languages.

program interface
The means used by a programming language to manipulate a product or products and produce the desired output. Such means can include protocols and extensions or syntax specifically developed and implemented for the activity and language.

program record description
A record description of the fields in a form. Nonimage fields exist in the program record without appearing on the form image. The program record description also can include fields used to programmatically control the display of the form image.

program tracking
See tracking.

programmatic control
In the Screen Design Facility (SDF), an option to generate extra data items, or flags, into a program record description. These data items can be set in the program or by the formlibrary to manipulate the form.

protected read
In Data Management System II (DMSII) and Semantic Information Manager (SIM), data protected from changes by other users during a transaction.

Q

qualification
(1) In Data Management System II (DMSII), 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. (2) In the Semantic Information Manager (SIM), the process of particularizing an attribute to a specific class, or a class to a specific database.

qualifier
Any of the following items: a data-name used in a reference together with another data-name at a lower level in the same hierarchy; a section-name used in a reference

together with a paragraph-name specified in that section; or a library-name used in a reference together with a text-name associated with that library.

quantifier
A Boolean operator that limits the variables of a condition, such as ALL, SOME, or NO.

query
A request to a database to retrieve, insert, update, or delete data.

query record
In the COBOL74 program interface for the Semantic Information Manager (SIM), a normal record (data-item) into which data is placed when a retrieval query is executed.

query record description
In Semantic Information Manager (SIM) application programs, that part of the query declaration that contains the names and descriptions of variables to be associated with database attributes.

query statement
In Semantic Information Manager (SIM) application programs, a basic programming statement that updates or retrieves entities.

query variable
In Semantic Information Manager (SIM) application programs, a variable that represents the query statement.

queue
A data structure used for storing objects; the objects are removed in the same order they are stored.

R

RDS
See restart data set.

read
The process of acquiring or interpreting data from an outside medium.

read-only access
The access to data that allows the data to be read but not changed.

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 area
A storage area allocated for the purpose of processing the record described in a record-description entry in the FILE SECTION.

record collection
In the Advanced Data Dictionary System (ADDS), a unique structure that consists of record structures and items that are grouped together under one name and invoked in COBOL74 and Pascal programs.

recovery
(1) 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. (2) In the Communications Management System (COMS), reconstruction of a database after a system failure.

recursive retrieval
In the Semantic Information Manager (SIM), recursive access to a reflexive attribute or a circular path expression during a retrieval query. *Synonym for* transitive function.

reflexive attribute
In the Semantic Information Manager (SIM), an entity-valued attribute (EVA) that refers to another entity in the same class. For example, SPOUSE can be a reflexive attribute of the class Person.

relation condition
A proposition, for which a truth value can be determined, in which the value of an arithmetic expression or data item has a specific relationship to the value of another arithmetic expression or data item.

relational operator
A reserved word, a relational character, a group of consecutive reserved words, or a group of consecutive reserved words and relational characters used in the construction of a relation condition.

relationship
An association between entities or data structure components.

remap
In Data Management System II (DMSII), 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.

reserved word
A word that has special meaning within a programming language and that generally cannot be redefined or redeclared by the programmer.

restart
To return to a particular point in a program and resume operation from that point.

restart data set (RDS)
In Data Management System II (DMSII), a data set containing restart records that application programs can access to recover database information after a system failure.

restart record
In Data Management System II (DMSII), a record containing user-defined information that enables a user program to restart in response to a particular condition.

retrieval query
A query used to select and retrieve data from a database.

rollback
The recovery of a database or transaction base to a consistent state at an earlier point in time.

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 unit
A set of one or more object programs that functions, at object time, as a unit to provide problem solutions.

S

scalar
(1) A quantity specified by a number on an appropriate scale. (2) A device that yields an output equal to the input multiplied by a constant.

Screen Design Facility (SDF)
The InterPro product used for creating forms for online, transaction-based application systems.

Screen Design Facility Plus (SDF Plus)
A Unisys product used for creating user interface systems (UISs) for online, transaction-based application systems.

SDF
See Screen Design Facility.

SDF Plus
See Screen Design Facility Plus.

segmented output
The output sent in separate segments or parts. In COBOL74, the use of the WITH option for the SEND statement can provide either nonsegmented or segmented output.

selection expression
(1) In Data Management System II (DMSII), 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. (2) An expression used to identify the set of entities upon which a query is to operate. The expression can be either global or local. (3) *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.

service function
An integer procedure of the Communications Management System (COMS) library that enables the user to access subroutines that can do the following: translate a designator to a name that represents a COMS entity; translate a name that represents a COMS entity to a designator; or obtain additional information about the name or designator passed to the service function.

set
In Data Management System II (DMSII), 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 sequence and are normally used to optimize certain types of retrievals of the data set records.

SIM
See Semantic Information Manager.

single-statement update
In Semantic Information Manager (SIM) application programs, an update query with one statement. An update query can also be a multiple-statement update.

single-valued attribute (SVA)
In the Semantic Information Manager (SIM), an attribute that assumes only one value for an entity. *Contrast with* multivalued attribute.

source
The symbolic identification of the originator of a transmission to a queue.

source program
A program coded in a language that must be translated into machine language before execution. The translator program is usually a compiler.

standard data format
The concept used to describe the characteristics of data in a DATA DIVISION. The data characteristics are expressed in a format oriented to the appearance of the data on a printed page, rather than a format oriented to the manner in which data are stored internally in the computer or on a particular external medium.

state
The condition of one or all the units or elements of a computer system.

statement
A syntactically valid combination of words and symbols written in the PROCEDURE DIVISION and beginning with a verb.

status
In the Advanced Data Dictionary System (ADDS), a feature that identifies the development cycle of entities.

string
A connected sequence or group of characters.

structure
(1) In Data Management System II (DMSII), a data set, set, subset, access, or remap. (2) In the Advanced Data Dictionary System (ADDS), a hierarchy of entities.

subcategory
In Data Management System II (DMSII), a numeric value that describes the type of error identified in a major category.

subfile
A logical, hierarchical division of a file.

subformat
See transaction subformat.

subrole
In the Semantic Information Manager (SIM), one of the 12 possible data types available for a data-valued attribute (DVA). (The others are integer, real, number, Boolean, symbolic, date, time, character, string, Kanji, and user-defined.) The attribute of type subrole is used to define subclasses, and it is a read-only attribute. For example, the class Person can have a subrole attribute called PROFESSION that defines Manager and Employee as subclasses of the class Person.

subscript
(1) A number that is an index into an array. (2) In COBOL, an integer whose value identifies a particular element in a table.

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 the data set records. A subset might or might not be automatically maintained by Data Management System II (DMSII).

support library
A library that is associated with a function name. User programs can access a support library by way of its function name instead of its object code file title. The operator uses the SL (support library) system command to link function names with libraries.

SVA
See single-valued attribute.

symbolic functions
In Semantic Information Manager (SIM), functions that operate on SIM symbolic data types, which define values that are identifiers.

synchronized recovery
In the Communications Management System (COMS), a function that resubmits incomplete transactions to the database after a transaction-state abort, system crash, or rollback occurs. This COMS function is called synchronized recovery because it reprocesses transactions in the same order that they were originally processed by multiple programs running asynchronously, even if the transactions were conflicting.

syncpoint
In Data Management System II (DMSII), a point in time when no program is in a transaction state.

synonym
In the Advanced Data Dictionary System (ADDS), a different name that can be given to an entity to handle differences in host language identifier rules. The synonym retains all the primary attributes of the target entity.

syntax
The rules or grammar of a language.

system library
A library that is part of the system software and is accorded special privileges by the operating system. Two examples of system libraries are GENERALSUPPORT and PRINTSUPPORT.

SYSTEM/BINDER
See Binder.

T

table
A one-dimensional or multidimensional structure in which like data items are stored. Each data item can be uniquely identified and accessed by means of its location in the table; identification and access procedures vary according to the language or product.

tabular selection
In Semantic Information Manager (SIM) application programs, the process of requesting information to produce output in a tablelike form.

TADS
See Test and Debug System.

terminal
(1) An I/O device designed to receive or send source data in a network. (2) In COBOL, the originator of a transmission to a queue or the receiver of a transmission from a queue.

Test and Debug System (TADS)
A Unisys interactive tool for testing and debugging programs and libraries. TADS enables the programmer to monitor and control the execution of the software under testing and examine the data at any given point during program execution.

text
In data communications, the part of a message containing information that has an ultimate purpose and destination beyond the data communications subsystem.

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.

TPS
See transaction processing system.

tracking
In the Advanced Data Dictionary System (ADDS), a feature of the data dictionary that enables the data management module (DMM) to keep track of the entities that are used by application source programs.

trancode
See transaction code.

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 base
In the transaction processing system (TPS), the software and files that constitute a TPS interface to a database.

transaction code (trancode)
(1) A sequence of characters included in a message that indicates the agenda to apply to a message during preprocessing or postprocessing. (2) In the Communications Management System (COMS), a code that can appear in a transaction-initiating message header, indicating the processing that is to be carried out. This code is used to route the message to the appropriate host program.

transaction compile-time function
In the transaction processing system (TPS), a function that provides access to certain properties of transaction record formats that are constant at compile time.

Transaction Formatting Language (TFL)
The Unisys language used to write source files that are compiled to produce description files for transaction bases.

transaction journal
In the transaction processing system (TPS), a collection of one control file and any number of data files. The transaction journal stores information about transactions.

transaction library
In the transaction processing system (TPS), a collection of procedures accessed by user-written programs to process or tank transactions and to read the transactions back from a transaction journal. The procedures are accessed through a set of entry points supplied by the transaction library, which is tailored for a particular transaction base during compilation.

transaction number
In the Screen Design Facility Plus (SDF Plus), the unique number by which SDF Plus internally references a transaction type.

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 record
In the transaction processing system (TPS), a structured variable that contains user-defined data items and system-defined control items for individual transactions. The user-defined items are similar to the data items in a Data Management System II (DMSII) data set record or a COBOL 01-level variable. A transaction record can be passed as a parameter to and from a procedure. It can also be read from and written to a transaction journal.

transaction record format
In the Transaction Formatting Language (TFL), a construct that defines the format of a single transaction record, including the data items and group items that can be contained in the record.

transaction state
In Data Management System II (DMSII), the period in a user-language program between a begin transaction operation and an end transaction operation.

transaction subformat
In the transaction processing system (TPS), the variable part of a transaction record format.

transaction type
In the Screen Design Facility Plus (SDF Plus), a group of records in the dictionary that describes the format of a transaction. A transaction type contains a pair of message types: a request message type and a response message type.

transitive closure
See transitive function.

transitive function
In the Semantic Information Manager (SIM), recursive access to a reflexive attribute or a circular path expression during a retrieval query. Transitive function is also called transitive closure. *Synonym for* recursive retrieval.

truth value
The representation of the evaluation results of a condition in terms of one of two values: TRUE or FALSE.

two-phase transaction
A transaction in which the first execution phase locks records without freeing any, the second and final execution phase of the transaction frees records without locking any, and no records are retrieved without locking them.

U

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.

update library
In the transaction processing system (TPS), a collection of user-written processing routines that serve as an interface between the transaction library and a Data Management System II (DMSII) database. These processing routines can be written in any of the DMSII user languages: Burroughs Data Management System ALGOL (BDMSALGOL), Burroughs Data Management System COBOL74 (BDMSCOBOL74), or Burroughs Data Management System COBOL (BDMSCOBOL). The update library is the only user-written module in TPS that contains the database declaration and all the code that performs data management statements against the database.

user work area
In Data Management System II (DMSII), 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.

user-defined word
A word that must be supplied by the user to satisfy the format of a clause or statement.

usercode
An identification code used to establish user identity and control security, and to provide for segregation of files. Usercodes can be applied to every task, job, session, and file on the system. A valid usercode is identified by an entry in the USERDATAFILE.

V

variable

(1) An object in a program whose value can be changed during program execution. (2) In the Screen Design Facility Plus (SDF Plus), a component of a form that stores data entered in the fields of the form image or the return value for a menu or a function key. A variable is also referred to as a display variable.

variable format

In Data Management System II (DMSII), 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.

verb

A word that expresses an action to be taken by a COBOL compiler or object program.

version

In the Advanced Data Dictionary System (ADDS), an optional entry field that allows variations of an entity.

virtual station (VS)

In data communications, a station declared in Network Definition Language II (NDLII) for both hosts connected to the same line. Terminal data conveyed between hosts during terminal transfer uses the device address field of a virtual station.

virtual terminal (VT)

(1) *See also* virtual station. (2) In the Screen Design Facility Plus (SDF Plus), the standardized presentation of logical terminal characteristics to an application program.

VS

See virtual station.

VT

See virtual terminal.

W

WFL

See Work Flow Language.

window

In the Communications Management System (COMS) architecture, the concept that enables a number of program environments to be operated independently and simultaneously at one station. One of the program environments can be viewed while the others continue to operate.

word

(1) 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. (2) In COBOL,

a character-string of not more than 30 characters that forms a user-defined word, a system-name, or a reserved word.

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.

write
(1) The process of transferring information to an output medium. (2) To record data in a storage device or location, or in a register.

0

01-level description entry
A record description that consists of a set of data-description entries that describe the characteristics of a particular record. Levels indicate the organization of elementary and group items; records start at level-number 01.

7

77-level description entry
A data-description entry that describes a noncontiguous data item with the level-number 77.

Bibliography

A Series COBOL ANSI-74 Programming Reference Manual, Volume 1: Basic Implementation (8600 0296). Unisys Corporation.

A Series COBOL ANSI-74 Test and Debug System (TADS) Programming Guide (1169901). Unisys Corporation.

A Series Communications Management System (COMS) Capabilities Manual (8600 0627). Unisys Corporation.

A Series Communications Management System (COMS) Configuration Guide (8600 0312). Unisys Corporation.

A Series Communications Management System (COMS) Migration Guide (8600 1567). Unisys Corporation.

A Series Communications Management System (COMS) Operations Guide (8600 0833). Unisys Corporation.

A Series Communications Management System (COMS) Programming Guide (8600 0650). Unisys Corporation.

A Series Data Base Data Transfer (DBT) Utility User's Guide (1180585). Unisys Corporation.

A Series Data Management Software Installation Guide (8600 1658). Unisys Corporation.

A Series DMSII Application Program Interfaces Programming Guide (5044225). Unisys Corporation.

A Series DMSII Data and Structure Definition Language (DASDL) Programming Reference Manual (8600 0213). Unisys Corporation.

A Series DMSII Interpretive Interface Programming Reference Manual (8600 0155). Unisys Corporation.

A Series DMSII Transaction Processing System (TPS) Programming Guide (1164043). Unisys Corporation.

A Series DMSII Utilities Operations Guide (8600 0759). Unisys Corporation.

A Series InfoExec Advanced Data Dictionary System (ADDS) Operations Guide (8600 0197). Unisys Corporation.

A Series InfoExec Capabilities Manual (8600 0254). Unisys Corporation.

A Series InfoExec Interactive Query Facility (IQF) Operations Guide (8600 0767). Unisys Corporation.

A Series InfoExec Semantic Information Manager (SIM) Object Manipulation Language (OML) Programming Guide (8600 0163). Unisys Corporation.

A Series InfoExec Semantic Information Manager (SIM) Programming Guide (8600 1666). Unisys Corporation.

A Series InfoExec Semantic Information Manager (SIM) Technical Overview (8600 1674). Unisys Corporation.

A Series Screen Design Facility (SDF) Capabilities Manual (1180437). Unisys Corporation.

A Series Screen Design Facility (SDF) Operations and Programming Guide (1185295). Unisys Corporation.

A Series Screen Design Facility Plus (SDF Plus) Capabilities Manual (8600 0270). Unisys Corporation.

A Series Screen Design Facility Plus (SDF Plus) Installation and Operations Guide (8600 0262). Unisys Corporation.

A Series Screen Design Facility Plus (SDF Plus) Technical Overview (8600 0072). Unisys Corporation.

A Series System Software Utilities Operations Reference Manual (8600 0460). Unisys Corporation.

A Series Systems Functional Overview (8600 0353). Unisys Corporation.

A Series Task Attributes Programming Reference Manual (8600 0502). Unisys Corporation.

A Series Work Flow Language (WFL) Programming Reference Manual (8600 1047). Unisys Corporation.

Workstations InfoExec Workstation Query Facility (WQF) Operations Guide (1185279). Unisys Corporation.

Index

A

B

C

D

E

F

G

H

I

K

L

M

N

P

Q

R

S

T

U

V

W

Y

UNISYS

Help Us To Help You

Publication Title

Form Number

Unisys Corporation is interested in your comments and suggestions reguarding this manual. We will use them to improve the quality of your Product Information. Please check type of suggestion:

☐ Addition ☐ Deletion ☐ Revision ☐ Error

Comments:

Name Telephone number ( )

Title Company

Address

City State Zip code

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS MAIL PERMIT NO. 817 DETROIT, MI

POSTAGE WILL BE PAID BY ADDRESSEE

UNISYS CORPORATION
ATTN: PUBLICATIONS
25725 JERONIMO ROAD
MISSION VIEJO, CA 92691-9826

Fold Here

Tape

Please Do Not Staple

Tape

Cut along dotted line

86000130-000

END-TRANSACTION Statement

The END-TRANSACTION statement commits all updates applied within a transaction, and takes a program out of transaction state.

General Format

The general format of an END-TRANSACTION statement is as follows:

```
END-TRANSACTION [COMS-header-name-1 [USING data-name-1]]
[ ON EXCEPTION { imperative-statement-1   } ] .
               { conditional-statement-1  }
               { NEXT SENTENCE            }
```

Explanation of Format Elements

COMS-header-name-1

This option specifies the COMS output header. This call is made before the exception-handling procedure is executed.

COMS-header-name-1 is used only with COMS. COMS-header-name-1 causes the COBOL74 program to call the DCIENTRYPOINT of a DCI library when an exception condition is detected. This program call enables a program interfacing with COMS to support synchronized transactions and recovery.

USING data-name-1

The USING option enables the message area indicated by data-name-1 to be passed to the DCIENTRYPOINT when the call is made on the DCI library.

ON EXCEPTION

The ON EXCEPTION option is described under "Handling SIM Exceptions" later in this section.

See Also

Refer to Section 3, "Using the COMS Program Interface," for more information on COMS.