



Application Development Solutions

COBOL ANSI-74 Programming Reference Manual

Volume 2
Product Interfaces

ClearPath MCP 13.1

April 2011

8600 0130-305

NO WARRANTIES OF ANY NATURE ARE EXTENDED BY THIS DOCUMENT. Any product or related information described herein is only furnished pursuant and subject to the terms and conditions of a duly executed agreement to purchase or lease equipment or to license software. The only warranties made by Unisys, if any, with respect to the products described in this document are set forth in such 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, 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.

Notice to U.S. Government End Users: This is commercial computer software or hardware documentation developed at private expense. Use, reproduction, or disclosure by the Government is subject to the terms of Unisys standard commercial license for the products, and where applicable, the restricted/limited rights provisions of the contract data rights clauses.

Contents

Section 1. Introduction to COBOL74 Program Interfaces

Documentation Updates	1-3
What's New?	1-3
Using Program Interfaces for Specific Products	1-3
Using Language Extensions for Specific Products	1-4
ADDS Extensions	1-4
Transaction Server Extensions	1-5
Enterprise Database Server Extensions	1-5
Enterprise Database Server TPS Extensions	1-7
SDF Extensions	1-8
SDF Plus Extensions	1-9

Section 2. Using the ADDS Program Interface

Accessing Entities with a Specific Status	2-2
Identifying Specific Entities	2-3
VERSION Clause	2-3
DIRECTORY Clause	2-4
Assigning Alias Identifiers	2-4
Identifying a Dictionary	2-6
Selecting a File	2-8
Invoking File Descriptions	2-10
File Description Entries	2-10
INVOKE ALL Option	2-12
Invoking Data Descriptions	2-13
ADDS Programming Examples	2-15
Invoking Entities and Program Tracking	2-15
Using ADDS with SDF	2-21

Section 3. Using the Transaction Server Program Interface

Preparing the Communication Structure	3-2
Declaring a Message Area	3-3
Declaring a Transaction Server Interface	3-3
Using Transaction Server Headers	3-4
Declaring Transaction Server Headers	3-4
Mapping Transaction Server Data Types to COBOL74	3-6
Using Transaction Server Input Header Fields	3-6
Using Transaction Server Output Header Fields	3-9
Preparing to Receive and Send Messages	3-11

Linking an Application Program to Transaction Server	3-11
Initializing an Interface Link	3-13
Using the DCI Library	3-14
Using Communication Statements	3-18
ACCEPT MESSAGE COUNT Statement	3-18
DISABLE Statement	3-19
ENABLE Statement	3-20
RECEIVE Statement	3-22
SEND Statement	3-24
Using Service Functions	3-30
Using Transaction Server Designators	3-31
Identifying Information with Service Function Mnemonics	3-31
Calling Service Functions	3-33
Passing Parameters to Service Functions	3-35
Transaction Server Sample Program	3-51
Transaction Server Sample Program with an Enterprise Database Server Database	3-51

Section 4. Using the Enterprise Database Server Program Interface

Using Database Items	4-2
Naming Database Components	4-2
Using Set and Data Set Names	4-2
Referencing Database Items	4-4
Declaring a Database	4-7
Invoking Data Sets	4-10
Using a Database Equation Operation	4-18
Using Selection Expressions	4-20
Using Data Management Attributes	4-23
Count Attribute	4-23
Record Type Attribute	4-24
Population Attribute	4-25
Using the DATADICTINFO Option	4-27
Manipulating Data in a Database	4-28
ABORT-TRANSACTION Statement	4-28
ASSIGN Statement	4-29
BEGIN-TRANSACTION Statement	4-32
CANCEL TRANSACTION POINT Statement	4-35
CLOSE Statement	4-36
COMPUTE Statement	4-38
CREATE Statement	4-39
DELETE Statement	4-41
DMTERMINATE Statement	4-44
END-TRANSACTION Statement	4-44
FIND Statement	4-47
FREE Statement	4-49
GENERATE Statement	4-51
IF Statement	4-53

INSERT Statement	4-56
LOCK/MODIFY Statement.....	4-58
OPEN Statement	4-60
RECREATE Statement	4-62
REMOVE Statement	4-64
SAVE TRANSACTION POINT Statement	4-66
SECURE Statement	4-67
SET Statement	4-68
STORE Statement	4-71
Processing Enterprise Database Server Exceptions	4-74
DMSTATUS Database Status Word	4-74
DMSTRUCTURE Structure Number Function	4-76
Enterprise Database Server Exceptions	4-77
Enterprise Database Server Sample Program	4-80

Section 5. Using the Enterprise Database Server TPS Program Interface

Using the Transaction Formatting Language (TFL)	5-2
Declaring a Transaction Base	5-3
Creating Transaction Records.....	5-6
Declaration of Transaction Record Variables	5-7
Creation of Transaction Record Formats	5-8
Using Transaction Records.....	5-10
Transaction Record Variables as Parameters	5-10
Transaction Record Variable Assignment.....	5-11
Accessing Transaction Record Items	5-12
Subscripts	5-13
Data Item Qualification	5-15
Inquiring about Transaction Record Control Items	5-16
Using Transaction Compile-Time Functions.....	5-18
Using Transaction Library Entry Points.....	5-20
Parameter Passing with COBOL74 Constructs	5-20
TPS Entry Points in COBOL74	5-22
Using the Update Library.....	5-23
Update Library Programming Conventions	5-23
Declaration of the Transaction Use Procedures	5-24
Parameter Passing to an Update Library	5-25
Transaction Processing Statements	5-26
TPS Programming Examples	5-31
Enterprise Database Server TPS Complete Programming Example.....	5-31

Section 6. Using the SDF Program Interface

Identifying the Dictionary	6-2
Invoking Data Descriptions.....	6-3
Reading Forms	6-6
Writing Forms	6-9
Using the FORM-KEY Function	6-11
Manipulating Programmatic Controls	6-12

Programmatic Flag Setting	6-12
Flag Group Generation	6-15
Using SDF with Transaction Server	6-17
SDF Sample Programs	6-19
SDF Program Using the READ Statement	6-19
SDF Program Using the WRITE and READ Statements	6-21
SDF Program Using Programmatic Controls	6-23
SDF Program Using Message Keys	6-27

Section 7. Using the SDF Plus Program Interface

Understanding Interface Elements	7-2
Form Record Libraries	7-2
Form Records	7-2
Form Record Numbers	7-3
Transaction Types	7-3
Transaction Numbers	7-3
Identifying the Dictionary	7-5
Invoking Data Descriptions	7-6
Selecting a Global Remote File	7-10
READ FORM Statement	7-10
WRITE FORM Statement	7-13
Using the Form Record Number Attribute	7-23
Using the Transaction Number Attribute	7-25
Using SDF Plus with the Transaction Server	7-26
Using Transaction Server Input/Output Headers	7-26
Sending and Receiving Messages	7-27
Sending Transaction Errors	7-28
Sending Text Messages	7-29
SDF Plus Sample Programs	7-30
SDF Plus Program with a Remote File Interface	7-30
SDF Plus Program with a Transaction Server Interface	7-34

Section 8. Open Distributed Transaction Processing for ClearPath MCP

Appendix A. User-Defined Words

Index	1
--------------------	----------

Tables

3-1.	Transaction Server Data Types and COBOL74 Usage	3-6
3-2.	Input Header Fields	3-8
3-3.	Output Header Fields	3-10
3-4.	Transmission Indicators for Identifier-1.....	3-27
3-5.	Service Function Mnemonics.....	3-31
5-1.	TFL Item Interpretations.....	5-2
5-2.	Using Compile-Time Functions	5-19
5-3.	TPS Entry Points	5-22
6-1.	Programmatic Flag Suffixes and Settings.....	6-13
6-2.	Programmatic Flag Information	6-13

Examples

2-1.	ADDs Sample Program to Invoke Entities.....	2-15
2-2.	ADDs Sample Program with SDF	2-21
3-1.	Transaction Server Sample Program with an Enterprise Database Server Database	3-52
4-1.	Enterprise Database Server Sample Program	4-81
5-1.	DASDL Description for Banking Transaction Program	5-31
5-2.	TFL Description for Banking Transaction Program.....	5-32
5-3.	Update Library for Banking Transaction Program	5-35
5-4.	Banking Transaction Program	5-40
5-5.	Declaring the Transaction Base and Entry Points.....	5-51
6-1.	SDF Sample Program with a READ Statement	6-20
6-2.	SDF Sample Program with READ and WRITE Statements	6-21
6-3.	SDF Sample Program with Programmatic Controls	6-23
6-4.	Sample Program with Message keys	6-27
7-1.	SDF Plus Program with a Remote File Interface.....	7-30
7-2.	Using SDF Plus with a Transaction Server Interface	7-34

Examples

Section 1

Introduction to COBOL74 Program Interfaces

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

- Advanced Data Dictionary System (ADDS)
- Transaction Server
- Enterprise Database Server
- Enterprise Database Server transaction processing system (TPS)
- Screen Design Facility (SDF)
- Screen Design Facility Plus (SDF Plus)
- Open Distributed Transaction Processing for ClearPath MCP

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

Volume 2 briefly describes the Open Distributed Transaction Processing for ClearPath MCP and provides references to the appropriate documentation.

This implementation of standard COBOL74 is based on, and compatible with, the American National Standard, X3.23-1974.

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.

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 Reference Manual

Throughout this manual, *Volume 1* refers to the *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. For the usage of 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.

Acronyms are 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.

Documentation Updates

This document contains all the information that was available at the time of publication. Changes identified after release of this document are included in problem list entry (PLE) 18781336. To obtain a copy of the PLE, contact your Unisys representative or access the current PLE from the Unisys Product Support Web site:

<http://www.support.unisys.com/all/ple/18781336>

Note: If you are not logged into the Product Support site, you will be asked to do so.

What's New?

Updated with miscellaneous minor changes and clarifications.

New or Revised Information	Location
Modified the STORE statement.	Section 4: STORE statement
Improved the example.	Section 4: Example 4-1

Using Program Interfaces for Specific Products

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.

COBOL74 program interfaces for products are easy to use, flexible, and efficient. The interfaces are the COBOL74 extensions to ANSI-74 COBOL and the files provided on the release media for the Open Distributed Transaction Processing for ClearPath MCP interface. 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)
- Transaction Server
- Enterprise Database Server
- Enterprise Database Server Transaction Processing System (TPS)
- Screen Design Facility (SDF)
- Open Distributed Transaction Processing for ClearPath MCP interface

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 standard COBOL74 comprise two groups.

- Extensions used only with specific products
- Extensions for general use

The extensions used only with specific products are explained in this volume. The extensions for general use 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.

Transaction Server Extensions

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

Extension	Explanation
ACCEPT MESSAGE COUNT statement	Makes available the number of messages in a program application queue.
Transaction Server header declaration	Provides information about the message for input or output.
DISABLE statement	Closes a Transaction Server direct window to a station, or disconnects a station reached through a modem or a Communications Processor 2000.
ENABLE statement	Initializes the interface between Transaction Server and a program, and opens a Transaction Server direct window to a station not currently attached to Transaction Server.
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 Transaction Server service function routines and enables a mnemonic parameter to be passed to obtain a numeric result. The VALUE clause appears in the CALL statement.

Enterprise Database Server Extensions

The following extensions have been developed for the Enterprise Database Server interface. Detailed information on each extension is provided in Section 4, "Using the Enterprise Database Server 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.

Extension	Explanation
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.
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.

Extension	Explanation
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.

Enterprise Database Server TPS Extensions

The following extensions have been developed for the Enterprise Database Server TPS interface. More information on the extensions is provided in Section 5, "Using the Enterprise Database Server 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 Enterprise Database Server 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.

Extension	Explanation
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 Transaction Server 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 Transaction Server.
FROM DICTIONARY clause	Invokes a formlibrary from the dictionary.
READ FORM statement	Reads specific and self-identifying forms.
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.

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. 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 DICTONARY 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 DICTONARY 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 Enterprise Database Server databases.

For information on Enterprise Database Server, see Section 4, "Using the Enterprise Database Server 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 DICTONARY 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 DICTONARY 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 DICTONARY compiler control option is as follows.

$$\underline{\text{DICTONARY}} = \left\{ \begin{array}{c} \underline{\text{PRODUCTION}} \\ \underline{\text{TEST}} \end{array} \right\}$$

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 DICTONARY compiler control option used to invoke entities that are defined in ADDS with a PRODUCTION status:

```
$SET DICTONARY = 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:

```
01 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, it is recommended 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.

**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:

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

Identifying a Dictionary

You can use the DICTONARY 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 DICTONARY statement, the compiler uses DATADICTONARY by default.

Note: A program can invoke only one dictionary. Therefore, if a program accesses SDF forms and another dictionary, then both must be in the same dictionary.

General Format

The general format of the DICTONARY 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

DICTONARY

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.

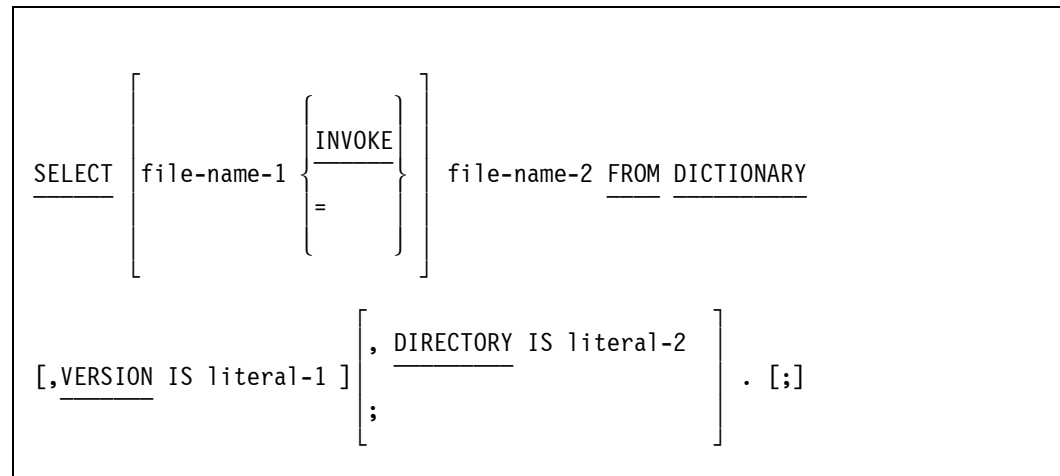
```
SPECIAL-NAMES.  
  DICTIONARY IS "DATADictionary"  
    PROGRAM-NAME IS "EXAMPLE-PROGRAM"  
    PROGRAM-VERSION IS 1  
    PROGRAM-DIRECTORY IS "JOHNDoe".
```

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.



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.

; (semicolon)

This option, the semicolon following the ending period, controls the behavior of compiler control records (CCRs) and format listings. The optional semicolon must be separated from the ending period of the SELECT statement by at least one space.

If a CCR immediately follows a SELECT . . . FROM DICTIONARY statement, the compiler control option changes might occur before the compiler processes the information invoked from the dictionary. You can avoid this situation by using the optional semicolon after the ending period. The semicolon ensures that the compiler processes the invoked information before the compiler control option changes.

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.

; (semicolon)

This option, the semicolon following the ending period, controls the behavior of compiler control records (CCRs) and format listings. The semicolon must be separated from the ending period of the FD or SD statement referencing a file from the dictionary by at least one space.

If a CCR immediately follows a FD or SD statement invoked from the dictionary, the compiler control option changes might occur before the compiler processes the information invoked from the dictionary. You can avoid this situation by using the optional semicolon after the ending period. The semicolon ensures that the compiler processes the invoked information before the compiler control option changes.

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 descriptions, SDF forms, 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 forms, 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.
```

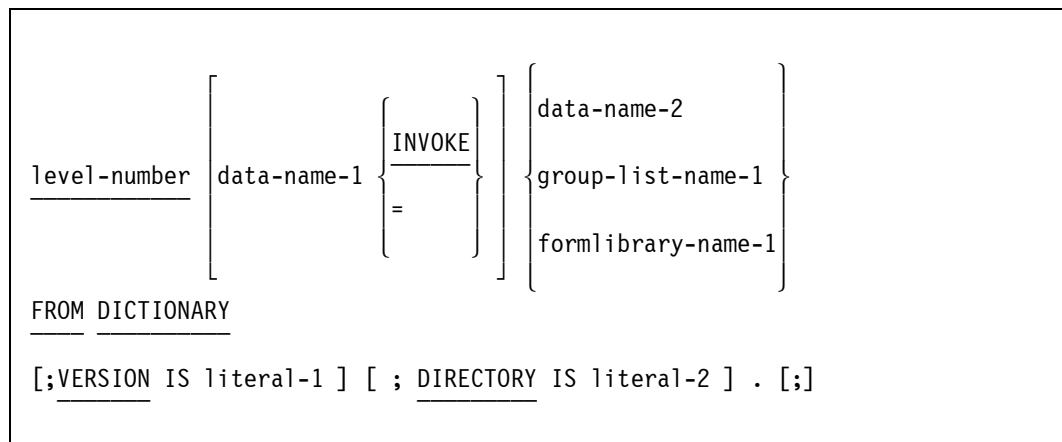

Invoking Data Descriptions

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 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 record. 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 record is stored in the data dictionary.

; (semicolon)

This option, the semicolon following the ending period, controls the behavior of compiler control records (CCRs) and format listings. The optional semicolon must be separated from the ending period of the data invocation by at least one space.

If a CCR immediately follows a data item invoked from the dictionary, the compiler control option changes might occur before the compiler processes the information invoked from the dictionary. You can avoid this situation by using the optional semicolon after the ending period. The semicolon ensures that the compiler processes the invoked information before the compiler control option changes.

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.
01 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.
```

Example 2–1. ADDS Sample Program to Invoke Entities

```
*****
*                                                                 *
```

```
*   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 "".
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*--    .
```

Example 2-1. ADDS Sample Program to Invoke Entities

```

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).
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

```

Example 2-1. ADDS Sample Program to Invoke Entities

```

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.
      * 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.

```

Example 2-1. ADDS Sample Program to Invoke Entities

```

000140 01 ADDS-RL-GRP-1.
000150 02 ADDS-RLG1-1 PIC S9(10)V99.
000160 02 ADDS-RLG2-2 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
004725 ON ERROR STOP RUN.
004800 READ FORM REMOTE-FILE USING ADDS-FORM-1
004825 ON ERROR STOP RUN.

```

Example 2-1. ADDS Sample Program to Invoke Entities

```
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

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.

```

001000 IDENTIFICATION DIVISION.
007000
008000 ENVIRONMENT DIVISION.
009000 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.               *
*****
011100
011200 SPECIAL-NAMES.
011300     DICTIONARY IS "DATADictionary"
011400     PROGRAM-NAME IS "SAMPLE2"
011425     PROGRAM-VERSION IS 1
011450     PROGRAM-DIRECTORY IS "USER1".
*****
*   The following lines declare the remote file.      *
*                                                     *
*****
011500
012000 INPUT-OUTPUT SECTION.
013000 FILE-CONTROL.
014000     SELECT REMFILE
014500     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.   *
*                                                     *
*****

```

Example 2–2. ADDS Sample Program with SDF

```

015000
016000 DATA DIVISION.
017000 FILE SECTION.
018000 FD REMFILE.
000100*--DICTIONARY DIRECTORY : *.
000200*--  BLOCK CONTAINS 2500 CHARACTERS
000300*--  RECORD CONTAINS 2500 CHARACTERS
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
D000110*--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

```

Example 2-2. ADDS Sample Program with SDF

```

*****
*
* The following line opens the remote file.
*
*****
026000 OPEN I-O REMFILE.
026100
*****
* 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

Section 3

Using the Transaction Server Program Interface

The Transaction Server is a general message control system (MCS) that controls online environments. Transaction Server supports a network of users and handles a high volume of transactions from programs, stations, and remote files.

The program interface for the Transaction Server enables you to create online, interactive, and batch programs that take advantage of the features that Transaction Server offers for transaction processing through direct windows. The program interface enables your program to communicate with the Transaction Server, using the following Transaction Server 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 the Transaction Server, and dynamic communication by modem
- Synchronized recovery

Transaction Server can be used with the Advanced Data Dictionary System (ADDs), Enterprise Database Server, Screen Design Facility (SDF), and Screen Design Facility Plus (SDF Plus). 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 Transaction Server extensions to communicate with the Transaction Server through direct windows. For an alphabetized list of the extensions developed for Transaction Server, refer to "Transaction Server 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 Transaction Server headers when receiving and sending messages.
- Attaching to and detaching from stations dynamically.

- Preparing to receive and send messages. This preparation includes linking to the Transaction Server 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 Transaction Server programming concepts, are provided in the *Transaction Server Programming Guide*.

When Enterprise Database Server is run with Transaction Server, the following Enterprise Database Server statements are used to enable a program interfacing with Transaction Server to support synchronized transactions and recovery:

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

Transaction Server can also use the Enterprise Database Server TPS MID-TRANSACTION statement.

See Also

- Information about the syntax and use of Enterprise Database Server statements is provided in Section 4, "Using the Enterprise Database Server Program Interface".
- Information about the use of the Enterprise Database Server TPS *MID-TRANSACTION* statement is provided in Section 5, "Using the Enterprise Database Server TPS 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 Transaction Server 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 a 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, Transaction Server 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 Transaction Server message area. The declaration occurs in the DATA DIVISION.

```
IDENTIFICATION DIVISION.  
PROGRAM-ID. COMSMMSG.  
DATA DIVISION.  
WORKING-STORAGE SECTION.  
01 COMS-MESSAGE-AREA.  
    02 COMS-MESSAGE          PIC X(2500).  
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 Transaction Server, see Section 6, "Using the SDF Program Interface," and the sample programs in the *Transaction Server Programming Guide*.

Declaring a Transaction Server Interface

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

Transaction Server headers offer the following advantages:

- You do not need to know the memory structure because Transaction Server handles the location of fields in the headers.
- Transaction Server 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 Transaction Server occur.
- Transaction Server headers can be referenced by bound-in procedures.

Using Transaction Server Headers

There are two types of Transaction Server 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 Transaction Server headers begins with a description of the declaration for Transaction Server headers and an explanation of how the Transaction Server 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 *Transaction Server Programming Guide*.

Declaring Transaction Server 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 the Transaction Server headers in your program. Examples are also provided.

General Format

The general format for declaring input and output Transaction Server 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 the Transaction Server 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 the Transaction Server. 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 the Transaction Server; 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 Enterprise Database Server Program Interface."
- For more information on the CONVERSATION AREA option and the fields of the input and output headers see "Using Transaction Server Input Header Fields" and "Using Transaction Server 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 the Transaction Server input and output headers. An example of declarations for headers within the context of a complete program is provided in Example 3–1, "Transaction Server Sample Program with an Enterprise Database Server Database."

```
COMMUNICATION SECTION.  
INPUT HEADER COMS-IN  
  PROGRAMDESG IS COMS-IN-PROGRAM.  
CONVERSATION AREA.  
  02 CA.  
    05 CA-1 PIC X(20).  
    05 CA-2 PIC X(30).  
OUTPUT HEADER COMS-OUT.
```

Mapping Transaction Server Data Types to COBOL74

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

Table 3–1. Transaction Server Data Types and COBOL74 Usage

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

Transaction Server data types include a Transaction Server designator data type that is used only for specific fields and with service functions. Transaction Server 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 Transaction Server designators used with service functions is provided in “Using Transaction Server 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 Transaction Server type TIME(6), you must redefine the type as a PIC X(6) DISPLAY item that you test against.

Using Transaction Server 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 the Transaction Server; 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 the Transaction Server or COBOL74. The structure of the Transaction Server input headers is obtained from the Transaction Server at compilation.

The Transaction Server 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 Transaction Server entity. See "Using Service Functions" later in this section for more information.

The Transaction Server 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 *Transaction Server Programming Guide*.
- For information about data types, see "Mapping Transaction Server Data Types into COBOL74" and "Using Transaction Server Designators" in this section.

Table 3–2 lists the fields of the input header, showing the Transaction Server and COBOL74 field names, the Transaction Server 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, "Transaction Server Sample Program with an Enterprise Database Server Database," (lines 017002 through 017062) at the end of this section.

Table 3–2. Input Header Fields

Transaction Server Field Name	COBOL74 Field Name	Transaction Server Data Type	COBOL74 Usage
Agenda Designator	AGENDA	Designator	Real
Continuator Data Status	CONTDATASTATUS	Mnemonic	Binary
Continuator Data Offset	CONTDATAOFFSET	Integer	Binary
Continuator Data Length	CONTDATALENGTH	Integer	Binary
Continuator Entry Number	CONTENTRYNUM	Integer	Binary
Function Index	FUNCTIONINDEX	Mnemonic	Binary
Function Status	FUNCTIONSTATUS	Mnemonic	Binary
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 Transaction Server 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 the Transaction Server; it is an optional user-defined field that is associated with the output header. All other fields in the header are defined by either the Transaction Server or COBOL74. The structure of the Transaction Server output headers is obtained from the Transaction Server during compilation.

Transaction Server 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 the Transaction Server entities to designators.

Transaction Server 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 *Transaction Server Programming Guide*.
- For information on data types, refer to "Mapping Transaction Server Data Types to COBOL74" earlier in this section and to "Using Transaction Server 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 *Transaction Server 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 the Transaction Server and COBOL74 field names, the Transaction Server data types, and the COBOL74 usages. The fields are listed alphabetically.

Table 3–3. Output Header Fields

Transaction Server Field Name	COBOL74 Field Name	Transaction Server Data Type	COBOL74 Usage
Agenda Designator	AGENDA	Designator	Real
Casual Output	CASUALOUTPUT	Boolean	DMS Boolean
Continuator Mode	CONTMODE	Mnemonic	Binary
Continuator Data Status	CONTDATASTATUS	Mnemonic	Binary
Continuator Data Offset	CONTDATAOFFSET	Integer	Binary
Continuator Data Length	CONTDATALENGTH	Integer	Binary
Continuator Entry Number	CONTENTRYNUM	Integer	Binary
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
Map Alias Name	MAPALIAS	Display	Display
Next Input Agenda Designator	NEXTINPUTAGENDA	Designator	Real
Retain Transaction Mode	RETAINTRANSACTIONMODE	Boolean	DMS Boolean
Set Next Input Agenda	SETNEXTINPUTAGENDA	Boolean	DMS Boolean
Send wait	SENDWAIT	Boolean	DMS Boolean
Send wait timelimit	SENDWAITTIMELIMIT	Integer	Binary
Status Value	STATUSVALUE	Mnemonic	Binary
Text Length	TEXTLENGTH	Integer	Binary
Transparent	TRANSPARENT	Boolean	DMS Boolean
User-Defined Conversation Area	User defined	User defined	User defined

Requesting Delivery Confirmation on Output

Delivery confirmation is available for all stations. This Transaction Server 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, Transaction Server 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 *Transaction Server 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 the Transaction Server and initialize the program. The following information explains how to perform these tasks.

Linking an Application Program to Transaction Server

To prepare for receiving or sending messages, you must link your application program to the Transaction Server. Linkage of the program to the Transaction Server is achieved through the data communications interface (DCI) library, which contains the programmatic interface with the Transaction Server. The DCI library is linked to the Transaction Server by specifying COMSSUPPORT as a function name and by designating library access by function. Because Transaction Server 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 the Transaction Server. If you want to test versions of the Transaction Server with the currently installed version of the Transaction Server, 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. Transaction Server 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 *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 Transaction Server program by initiator. This method of linking to the Transaction Server is the preferred method for Transaction Server programs. All other programs should be linked by function.

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

Example of Linking by Title

A Transaction Server application program can link to the Transaction Server by title. This method is not recommended. Linking by title might be used, for example, when linkage is needed to a test version of the Transaction Server 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 the Transaction Server 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 the Transaction Server.*

The following example shows how to link your program to the Transaction Server 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 the Transaction Server 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 the Transaction Server:

```
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 DCIENTRYPPOINT. The DCIENTRYPPOINT library object is an untyped procedure.

The DCIENTRYPPOINT 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 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. DCIENTRYPPOINT.
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
*
01 THE-MESSAGE
   03 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 *Transaction Server 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.

<pre><u>ACCEPT</u> <u>COMS-header-name-1</u> <u>MESSAGE</u> <u>COUNT</u></pre>
--

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 the Transaction Server maintains for the program.

COMS-header-name-1

This format element is the name of the Transaction Server input header.

DISABLE Statement

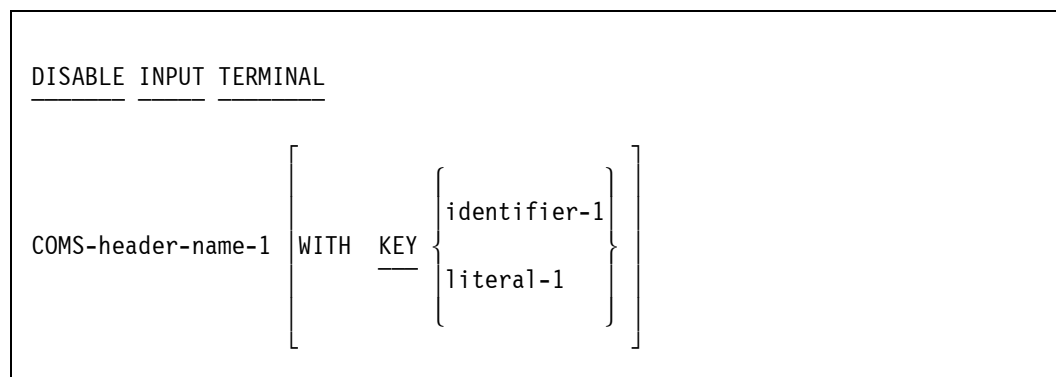
The DISABLE statement closes a direct window to a station or disconnects a station that was connected by a Protocol Specific Handler (PSH).

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

If the logical disconnection specified by the DISABLE statement has already occurred or is denied by the Transaction Server, the data item in the Status Value field is updated. See "Using Transaction Server 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.



The KEY option is required only if an identifier-1 or a literal-1 is specified.

The DISABLE statement is used for dynamically detaching a station.

Explanation of Format Elements

DISABLE INPUT

This phrase logically disconnects the Transaction Server 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 Transaction Server.

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 Transaction Server input header.

WITH KEY literal-1 or identifier-1

This format element must be alphanumeric. The KEY options are literals defined for use with the Transaction Server. KEY options are described in the *Transaction Server 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 the Transaction Server. You can check the status of a station attachment by querying values in the fields of the Transaction Server input header.

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

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

The formats for the ENABLE statement are as follows.

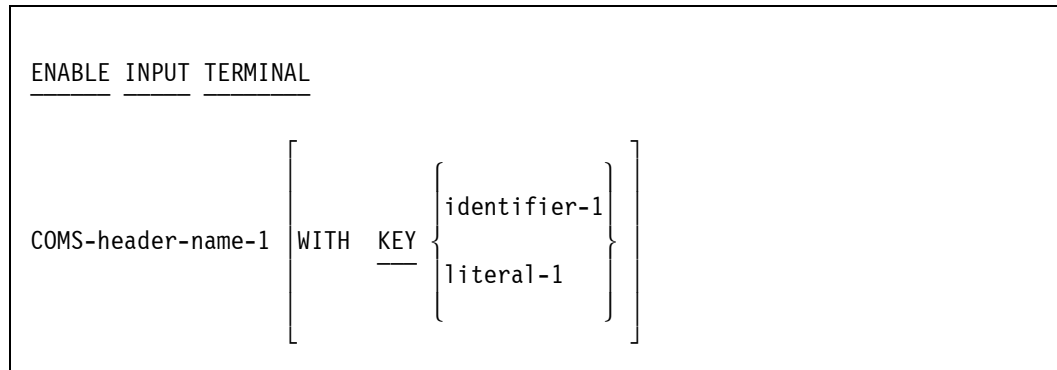
Format 1

<u>ENABLE</u>	<u>INPUT</u>	
COMS-header-name-1	WITH	<u>KEY</u> $\left\{ \begin{array}{l} \text{identifier-1} \\ \text{literal-1} \end{array} \right\}$

The KEY option of either an identifier-1 or a literal-1 is required.

Format 1 is used to establish communication between the application program and the Transaction Server.

Format 2



The KEY option is required only if an identifier-1 or a literal-1 is specified.

Format 2 is used for dynamically attaching a station.

Explanation of Format Elements

ENABLE INPUT

This phrase provides a logical connection between the Transaction Server 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 the Transaction Server.

TERMINAL

This option dynamically opens a direct window to a station that is not attached to the Transaction Server.

COMS-header-name-1

This format element specifies the name of a Transaction Server output header.

WITH KEY literal-1 or identifier-1

This format element must be alphanumeric. The KEY values provided by the Transaction Server are described in the *Transaction Server 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, “Transaction Server Sample Program with an Enterprise Database Server Database,” later in this section. For more information on the use of the ENABLE statement to establish communication with the Transaction Server 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.
```

```
MOVE "SERVER=SOME-SVR SOCKET = 12345" 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 the Transaction Server. 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 the Transaction Server 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  
]
```

Explanation of Format Elements

COMS-header-name-1

This format element specifies the name of a Transaction Server 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.

Always declare identifier-1 as an 01-level record. You can also use the RECEIVE statement to accept the information into an 01-level record, and then use the MOVE statement to move the 01-level record into the desired array element.

NO DATA imperative-statement-1

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

If the Transaction Server 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.

Note: When a RECEIVE statement successfully receives data, the execution continues with the next statement. If the NO DATA option is present and no data is received, then the imperative statement associated with the NO DATA option is executed. If the NO DATA option is not specified for the RECEIVE statement, then the execution does not proceed until data is present.

Considerations for Use

The data items identified by the Transaction Server input header are appropriately updated by the Transaction Server 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 the Transaction Server 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, "Transaction Server Sample Program with an Enterprise Database Server Database," later in this section.

```
01 MSG-IN-TEXT          PIC X(1920).
:
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 the Transaction Server. Before you can send or receive messages, however, you must link to the Transaction Server 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 the Transaction Server. See the *Transaction Server 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

```
SEND COMS-header-name-1 [FROM identifier-1] {  
    WITH identifier-2  
    WITH ESI  
    WITH EMI  
    WITH EGI  
}  
  
[ {  
    {  
        {  
            BEFORE  
            AFTER  
        } ADVANCING {  
            {  
                identifier-3  
                integer  
            } {  
                LINE  
                LINES  
            }  
            {  
                mnemonic-name  
                PAGE  
            }  
        }  
    }  
}]
```

Explanation of Format Elements

COMS-header-name-1

This format element specifies the name of a Transaction Server 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 the Transaction Server.

When you execute a SEND statement, Transaction Server 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), then 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 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). Transaction Server 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.

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.

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

A single execution of the SEND statement with segmented output never releases to the Transaction Server 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, Transaction Server ignores the specified or implied vertical positioning.

On a device where vertical positioning is possible and the ADVANCING option is not specified, the Transaction Server 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 Transaction Server 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 *Transaction Server 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.  
01 MESSAGE-1 PIC X(100).  
01 MESSAGE-2 PIC X(100).  
01 MESSAGE-3 PIC X(100).  
  
COMMUNICATION SECTION.  
INPUT HEADER INHDR.  
OUTPUT HEADER OUTHDR.  
PROCEDURE DIVISION.  
    MOVE OUTPUT-SIZE1 TO TEXTLENGTH OF OUTHDR.  
    MOVE 0 TO STATUSVALUE OF OUTHDR.  
    SEND OUTHDR FROM MESSAGE-1 WITH ESI.  
  
    MOVE OUTPUT-SIZE2 TO TEXTLENGTH OF OUTHDR.  
    MOVE 0 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 0 TO STATUSVALUE OF OUTHDR.  
    SEND OUTHDR FROM MESSAGE-3 WITH EGI.
```


Using Service Functions

This discussion of service functions includes explanations of the following:

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

Transaction Server uses the following service functions:

- CONVERT_TIMESTAMP
- GET_BUFFER_DESIGNATOR
- GET_DESIGNATOR_ARRAY_USING_DESIGNATOR
- GET_DESIGNATOR_USING_DESIGNATOR
- GET_DESIGNATOR_USING_NAME
- GET_ERRORTTEXT_USING_NUMBER
- GET_INTEGER_ARRAY_USING_DESIGNATOR
- GET_INTEGER_USING_DESIGNATOR
- GET_INPUT_QUEUE_STATUS
- GET_NAME_USING_DESIGNATOR
- GET_REAL_ARRAY
- GET_STRING_USING_DESIGNATOR
- GET_WINDOW_NAME_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 the Transaction Server.

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

Using Transaction Server Designators

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

See Also

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

Identifying Information with Service Function Mnemonics

The Transaction Server 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
Host name	HOST-NAME
Installation data link by another Transaction Server entity	INSTALLATION-DATA-LINK
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

Table 3–5. Service Function Mnemonics

Data Item	Mnemonic
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
Station list	STATION-LIST
Station: logical station number (LSN)	LSN
Station: security designator	SECURITY
Station: session number	SESSION
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

Table 3–5. Service Function Mnemonics

Data Item	Mnemonic
Usercode	USERCODE
XATMI buffer: length	LENGTH
XATMI buffer: subtype	SUBTYPE
XATMI buffer: type	TYPE
XATMI service designator	SERVICE

Calling Service Functions

You can call the Transaction Server 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 Transaction Server 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 the Transaction Server.

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 *Transaction Server 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 the Transaction Server 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 Transaction Server 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 *Transaction Server 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 Transaction Server timestamp TIME(6) to the date or time in EBCDIC. For information on the Transaction Server timestamp TIME(6) type and COBOL74 usage, refer to "Mapping Transaction Server 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.
01 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_BUFFER_DESIGNATOR Service Function

The GET-BUFFER-DESIGNATOR service function converts the XATMI buffer type and subtype variables into an XATMI buffer designator and the length of such an XATMI buffer expressed in bytes.

The input parameters are

- An XATMI buffer type
- An XATMI buffer subtype

The output parameters are

- A designator
- An integer representing the length of the XATMI buffer in bytes.

Example

In the following example of the GET_BUFFER_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.

```
.  
.   
WORKING-STORAGE SECTION.  
.  
01 WS-TYPE                      PIC X(8).  
01 WS-SUBTYPE                   PIC X(16).  
77 WS-DESG                      REAL.  
77 WS-BUFFER-LENGTH            PIC S9(11) USAGE BINARY.  
77 SF-RSLT                     PIC S9(11) USAGE BINARY.  
.  
.  
PROCEDURE DIVISION  
.  
.  
    CALL "GET-BUFFER-DESIGNATOR OF DCILIBRARY"  
        USING WS-TYPE  
              WS-SUBTYPE  
              WS-DESG  
              WS-BUFFER-LENGTH  
        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 the Transaction Server.

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.
01 WS-DESG-TABLE                      REAL.
   05 WS-D-TABLE                      REAL OCCURS 10 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, call	DEVICE
Call	SERVICE

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_ERRORTEXT_USING_NUMBER Service Function

The GET_ERRORTEXT_USING_NUMBER service function converts an XATMI function error code into text representing an error message.

The input parameter is an integer representing an XATMI function result value.

The output parameters are the following:

- An integer representing the length of the text string returned
- A text string representing an error message

If the error text is available in multiple languages, the language attribute of the calling program determines the language in which the error message is returned.

Example

In the following example of the GET_ERRORTEXT_USING_NUMBER 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.

```
.  
.   
WORKING-STORAGE SECTION.  
.  
77 WS-ERROR                      PIC S9(11) USAGE BINARY.  
77 WS-TEXT-LENGTH                PIC S9(11) USAGE BINARY.  
01 WS-ERROR-TEXT                 PIC X(90).  
77 SF-RSLT                       PIC S9(11) USAGE BINARY.  
.  
.  
PROCEDURE DIVISION  
.  
.  
    CALL "GET-ERRORTEXT-USING-NUMBER OF DCILIBRARY"  
        USING WS-ERROR  
              WS-TEXT-LENGTH  
              WS-ERROR-TEXT  
        GIVING SF-RSLT.
```

GET_DESIGNATOR_USING_NAME Service Function

The GET_DESIGNATOR_USING_NAME service function converts the Transaction Server 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 the Transaction Server. 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.
01 WS-INT-TABLE                 USAGE BINARY.
   03 WS-INT-TABLE-DETAIL       PIC S9(11) OCCURS 10 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, VIRTTERM, SCREENSZ, SESSION
Window	MAXIMUM_USER_COUNT and CURRENT_USER_COUNT
Program	QUEUE_DEPTH, MESSAGE_COUNT, LAST_RESPONSE, and AGGREGATE_RESPONSE
XATMI buffer	LENGTH

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_INPUT_QUEUE_STATUS Service Function

You can use this service function to find out which programs have a buffer memory in use that is greater than 0 (zero). The declaration in the Transaction Server is as follows:

```
INTEGER PROCEDURE GET_INPUT_QUEUE_STATUS (VERSION, INPQ_INFO);
```

The input parameter is VERSION, which is an integer that must equal 0 or 1. If the value is not 0 or 1, then the Transaction Server returns an error result of 1 (Failed). INPQ_INFO is a real array declared in your program. If the array is not large enough to hold all the input queue information, then the Transaction Server returns an error result of 3 (Short array).

The output results in the GET_INPUT_QUEUE_STATUS service function populating the INPQ_INFO array as follows.

Word no.	Content
1	Contains the number of queues that are using memory, which is greater than 0. Use this value as a counter to step through the queues starting at the eleventh word.
2	Contains the number of messages queued for all the programs.
3	Represents the buffer memory that the Transaction Server is using. This word has the same value as the fourth line that STATUS COMS returns.
4	Represents the buffer memory in the available pool. This word has the same value as the fifth line that STATUS COMS returns.
5	Represents the total input delay. This word has the same value as the seventh line that STATUS COMS returns.
6-10	Reserved
11	Queue number
12	Program designator (might be used to get program name)
13	Count of queued messages
14	Memory in use for the program
15	Memory limit for the program (specified in Utility)
16	1 if the queue input is being written to disk
17	1 if program is protected, 0 if not
18	1 if program is disabled with the NOW option
19	1 if the program is enabled, 0 if disabled

GET_INPUT_QUEUE_STATUS returns nine words of information for each queue in the following format:

Program designator (can be used to get program name)

1 if the queue input is being written to disk

1 if program is protected, 0 if not

1 if program is disabled with the NOW option

1 if the program is enabled, 0 if disabled

GET_NAME_USING_DESIGNATOR Service Function

The GET_NAME_USING_DESIGNATOR service function converts a Transaction Server designator to the Transaction Server 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.  
01 WS-NAME                PIC X(30).  
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.
01 WS-REAL-TABLE                REAL.
   05 WS-R-TABLE                REAL OCCURS 500 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 string. The allowable mnemonics are HOST-NAME; INSTALLATION_STRING_1 through INSTALLATION_STRING_4; INSTALLATION_HEX_1 and INSTALLATION_HEX_2; LANGUAGE; CONVENTION; TYPE; and SUBTYPE.
- 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 TYPE and SUBTYPE mnemonics provide a way of determining the type and subtype of the XATMI buffer.

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.
01 WS-STRING              PIC X(30).
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.
```

GET_WINDOW_NAME_USING_DESIGNATOR Service Function

The GET_WINDOW_NAME_USING_DESIGNATOR service function returns the name of the current window for a station.

The input parameter is the station designator.

The output parameter is the current window for the station. If there is no window for the station, then NONE is returned. The output parameter is blank and filled to the right.

RESET_PROGRAM_STATISTICS Service Function

The RESET_PROGRAM_STATISTICS service function resets the processed messages and response time statistics for a selected program.

The input parameter is the program name. The parameter must be terminated by a space if the program name is less than 17 characters.

Example

In the following example of the RESET_PROGRAM_STATISTICS 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 PROGRAM-NAME          PIC X(17).  
77 SF-RSLT               PIC S9(11)  USAGE BINARY.  
  
CALL "RESET_PROGRAM_STATISTICS OF DCILIBRARY"  
  USING PROGRAM-NAME  
  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. It is recommended that the size of the table is 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.  
.   
.   
.   
01  STATION-HASH                      REAL.  
    02  STATION-DESIGNATOR            REAL OCCURS 100.  
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 0
        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.

```

Transaction Server Sample Program

The sample program tracks sailboat races. It updates a database by using features of the Transaction Server direct-window interface. The program, SAILOLPROG, uses an Enterprise Database Server database.

Transaction Server Sample Program with an Enterprise Database Server 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).

Transaction Server Features Used in the Sample Program

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

- Declared Transaction Server input and output headers
- Trancodes
- Synchronized recovery

The SAILOLPROG program runs in a Transaction Server environment that has been configured to include an Enterprise Database Server database called SAILDB and the following Transaction Server 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 the Transaction Server to allow the program to run.

See Also

- Information on synchronized recovery when the Transaction Server is used with the Enterprise Database Server provided in the explanation of the CLOSE statement under "CLOSE Statement" in Section 4, "Using the Enterprise Database Server Program Interface."
- Information on TADS is provided in the *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 the Transaction Server. The Enterprise Database Server 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 the Transaction Server 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).
```

Example 3–1. Transaction Server Sample Program with an Enterprise Database Server Database

```

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
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;

```

Example 3-1. Transaction Server Sample Program with an Enterprise Database Server Database


```

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.
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.

```

Example 3-1. Transaction Server Sample Program with an Enterprise Database Server Database

```
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
```

Example 3-1. Transaction Server Sample Program with an Enterprise Database Server Database

```

022200      ELSE
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.
023050*      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.

```

Example 3-1. Transaction Server Sample Program with an Enterprise Database Server Database

```
025900 200-CREATE-RACE-EXIT.
026000     EXIT.
026100
026200*****
026300 300-ADD-ENTRY  SECTION.
026400*****
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
```

Example 3–1. Transaction Server Sample Program with an Enterprise Database Server Database

```

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.
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.

```

Example 3-1. Transaction Server Sample Program with an Enterprise Database Server Database

```
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 0             TO COMS-OUT-DESTINATION.
035000     MOVE 0             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 = 0 OR 92
035400       NEXT SENTENCE
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.
```

Example 3-1. Transaction Server Sample Program with an Enterprise Database Server Database

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

Example 3–1. Transaction Server Sample Program with an Enterprise Database Server Database

Section 4

Using the Enterprise Database Server Program Interface

Enterprise Database Server 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 Enterprise Database Server application program interface. The Enterprise Database Server 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 Enterprise Database Server, refer to Section 1, "Introduction to COBOL74 Program Interfaces." Refer to the Enterprise Database Server Application Programming Guide for information on general programming considerations and concepts.

The Enterprise Database Server program interface can be used with the Advanced Data Dictionary System (ADDS), Transaction Server, and Enterprise Database Server transaction processing system (TPS). 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 Enterprise Database Server, see the *InfoExec ADDS Operations Guide*.
- For additional information on using Transaction Server with Enterprise Database Server, see the *Transaction Server 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 does not issue a warning. If qualification is insufficient, the compiler usually uses the last item declared or invoked in the program that has the same name as the variable. To avoid unexpected results, do one or both of the following when invoking the Enterprise Database Server 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:  
01 Q.  
03 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.  
01 D1.
```

```
01 DA=D1.
```

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

The compiler and generated code file ignore the qualification of an item if it is used in a FIND/LOCK statement and the same item is also used in another data set. For example,

```
FIND S1 AT FIELD1 OF D2=10
```

In the preceding statement, FIELD1 is a field in the S1 set of D1 and is also a field in D2. The qualification "OF D2" of the key item FIELD1 in the FIND statement can be ignored under certain circumstances. In such cases, the key item, FIELD1, is associated with the S1 set of the D1 data set. The FIND statement described previously is interpreted as

```
FIND S1 AT FIELD1 OF D1=10
```

The ambiguity in the preceding statement does not occur if the key items are invoked from a logical database where their names are unique.

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.

Note: Do not consider control items to be part of the group item associated with the data set for any COBOL statement. MOVE statements using the data set record as a field do not include the control information. The possibility exists that some other statements (for example, MOVE CORRESPONDING, if the appropriate corresponding data items exist in both declarations) might inadvertently include some control information. Unisys strongly recommends that you refer to control items using only syntax that explicitly includes them by name, never as a part of a record in the database.

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:

```
01 D
   02 A
   02 B
   02 C
       03 E1
       03 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.

```
01 D DATA SET (#1).
   02 T RECORDTYPE.
   02 CT COUNT.
   02 A PIC X(6) DISPLAY.
   02 B PIC 9(6) COMP.
   02 C PIC 9(6) COMP.
   02 L REFERENCE TO E.
   02 E DATA SET (#2).
       03 E1...
       03 E2...
   02 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.

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

Example of Creating Valid and Invalid Enterprise Database Server 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 procedures or program must match the database reference in your COBOL74 program, and the components being bound together must be compiled against the same database description file. Failure to observe these restrictions might result in syntax errors when using BINDER to combine the separately compiled components into a single 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 *Data and Structure Definition Language (DASDL) Programming 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.
```

```
01 DATASET-1.
```

```
01 DATASET-2.
```

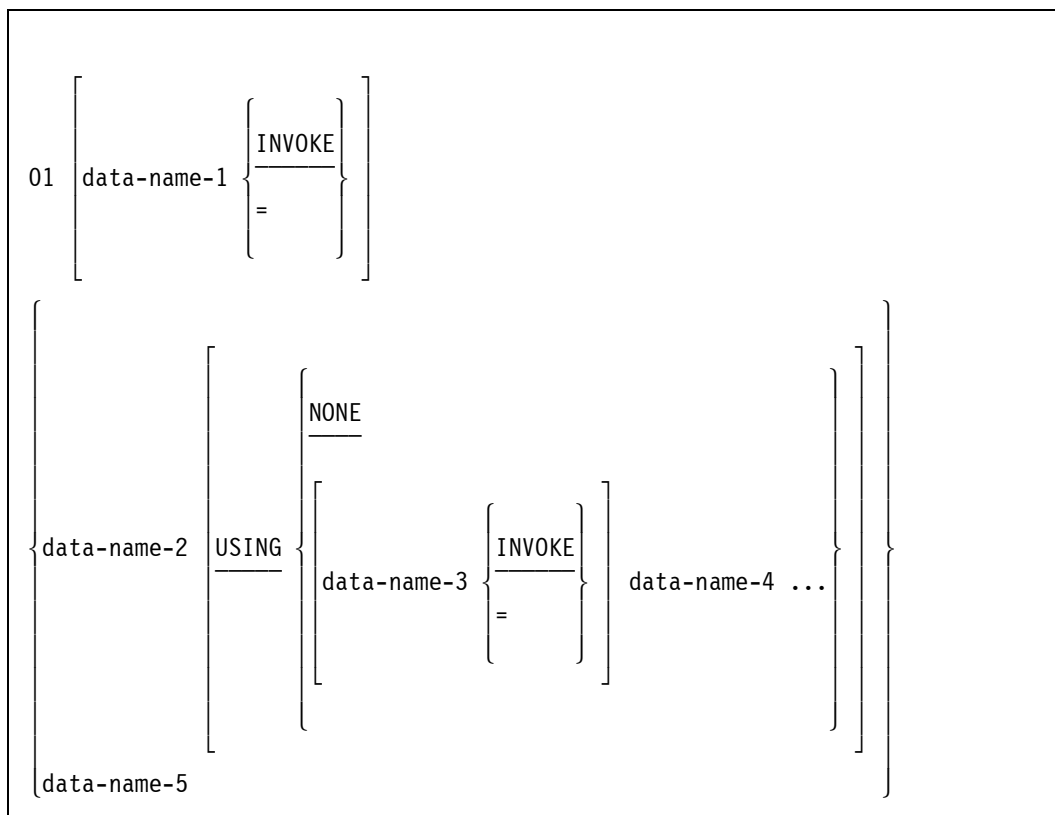
Invoking Data Sets

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.



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.

```
01 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.

```
01 D.
```

```
01 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.

```
01 D.
```

```
01 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.

```
01 D USING S1.
```

```
01 X=D USING S1.
```

```
01 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.

```
01 X=D USING S1.
```

```
01 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.

```
01 D.
```

```
01 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:

```
01 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:

```
01 F.  
01 E.  
01 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.
* 01 D1 STANDARD DATA SET(#2).
*      S1 SET(#4,AUTO) OF D1 KEYS ARE A,B,C.
* 02 A REAL.
* 02 B PIC 9(5) COMP.
* 02 C PIC X(10) DISPLAY.
* 01 D2 STANDARD DATA SET(#5).
*      S2 SET(#9,AUTO) OF D2 KEYS ARE X,Y,Z.
* 02 X FIELD SIZE IS 08 BITS.
* 02 Y PIC 99 COMP.
* 02 Z REAL.
* 02 E STANDARD DATA SET(#6).
*      SE SET(#7,AUTO) OF E KEY IS V1.
* 03 V1 REAL.
* 03 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 (10);
    FIRST ALPHA (10);
  );
  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 *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 an Enterprise Database Server Host Program to Call a Separately Compiled Procedure That References the Same Database

The following example shows the COBOL74 program declarations for the Enterprise Database Server 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 Access routines 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.

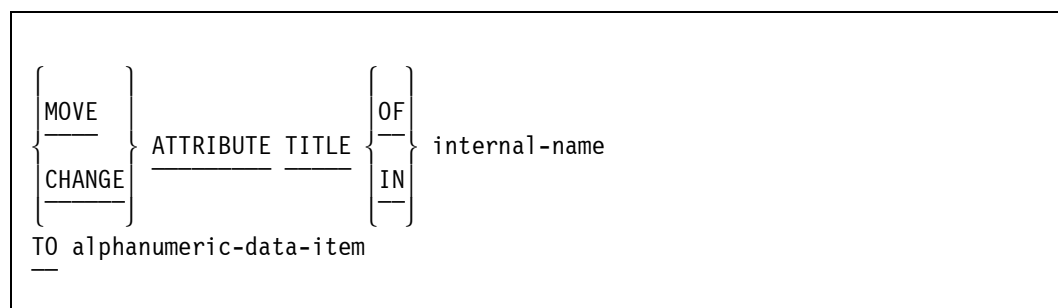
A database equation is like a file equation. A WFL equation overrides the specification of a database title in a language declaration, and run-time modification of a database title overrides both WFL equations and user language specifications. However, a database equation differs from a file equation 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.

Note: *When the database is accessed, the system verifies that the description of the database matches the description applied at the time the program was compiled. If this verification is unsuccessful, the program receives an exception on the access or terminates. Use of a database title equation is thus limited to cases in which the description file for the database that was originally named in the program matches that of the database referenced in the run-time database equation statement.*

General Format

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



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, “Enterprise Database Server Sample Program,” shows the MOVE statement within the context of a complete program.

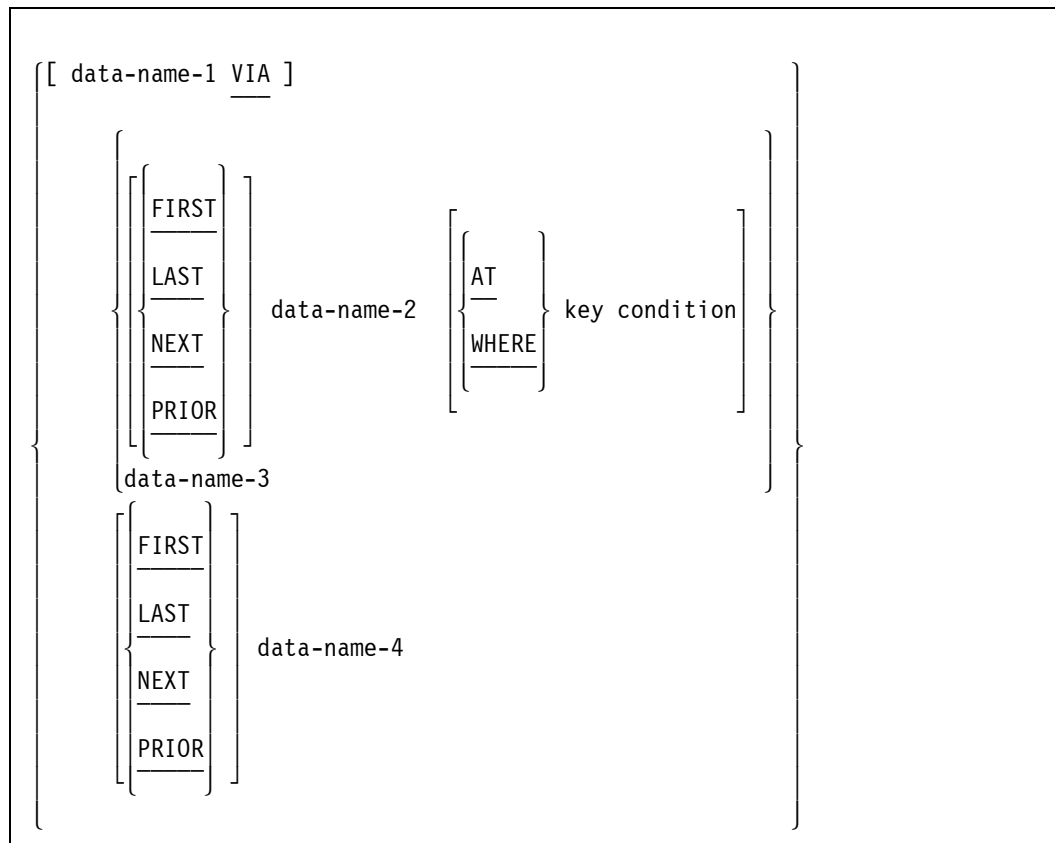
```
IDENTIFICATION DIVISION.  
PROGRAM-ID. DBEQUATE.  
.  
.  
.  
DATA-BASE SECTION.  
  
    DB MYDB ALL  
    VALUE OF TITLE IS "LIVEDB".  
.  
.  
.  
OPEN UPDATE MYDB.  
.  
.  
.  
CLOSE MYDB.  
CHANGE ATTRIBUTE TITLE OF MYDB  
    TO "(UC)TESTDB ON TESTPACK".  
OPEN UPDATE MYDB.  
.  
.  
.  
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, the Enterprise Database Server 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, the Enterprise Database Server 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, the Enterprise Database Server 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, the Enterprise Database Server 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. The Enterprise Database Server 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 format element selects the particular record to be located in the selection expression. A key condition is an expression that ultimately evaluates to a TRUE or FALSE value.

A key condition is a proper subset of a conditional expression with the following limitations:

- The Enterprise Database Server data-item must precede the relational operator.
- You cannot use the abbreviated form of a relation condition in a key condition.

A key condition is composed of the parts shown in the following table. The parts of the key condition must be present in the order listed.

Order	Part	Purpose
1	Enterprise Database Server data-item	Acts as the key in the key condition.
2	Relational operator	Indicates the relation condition.
3	Data-item or arithmetic expression	Acts as the target value against which the Enterprise Database Server data-item is to be compared.

You can repeat the key condition construction one or more times to create a more complex condition. Use left and right parentheses to indicate precedence, if desired or required.

data-name-3

This format element specifies a link item defined in DASDL. The Enterprise Database Server 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. The Enterprise Database Server 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 it 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". Note that in the

following example, "A = 50 AND (B = 50 OR C < 90)" and "A = "MYNAME"" are key conditions.

```
FIND S AT A = 50 AND (B = 50 OR C < 90).
```

```
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.

The Enterprise Database Server 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) = 0 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.

<code>data-name-1 (record-name-1)</code>
--

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.

<code>data-name-1 (data-name-2)</code>
--

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 Enterprise Database Server 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.

<pre>[<u>DATADICTINFO</u>]</pre>

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

Transaction Server -header-name-1

This option is used only with Transaction Server and specifies a Transaction Server header. Transaction Server -header-name-1 calls the DCENTRYPOINT of a data communications interface (DCI) library when a program detects an exception condition. This feature enables a program interfacing with Transaction Server 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 “Enterprise Database Server Exceptions” later in this section for information on the clause.

See Also

Refer to Section 3, “Using the Transaction Server Program Interface,” for more information on the Transaction Server.

Examples

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

```
ABORT-TRANSACTION VENDOR-RESTART.
```

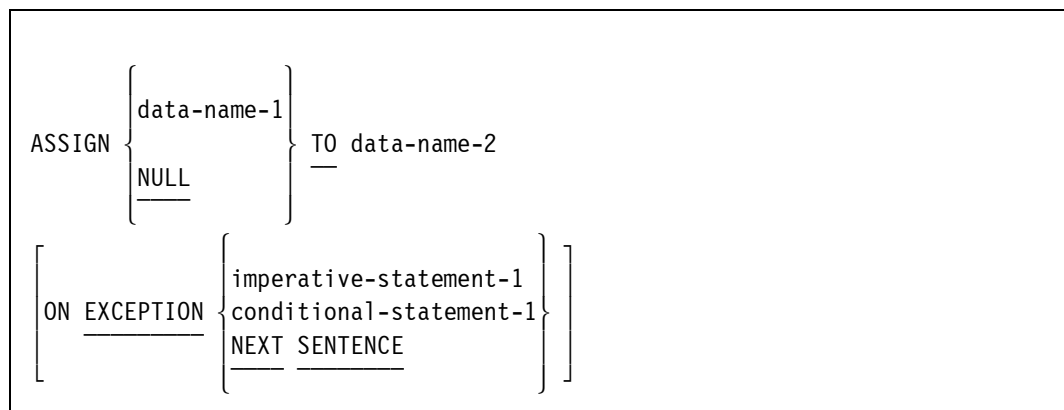
```
ABORT-TRANSACTION OUTHDR MY-RESTART.
```

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 the Enterprise Database Server 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 “Enterprise Database Server 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 Enterprise Database Server 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, the Enterprise Database Server 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.  
ENVIRONMENT DIVISION.  
INPUT-OUTPUT SECTION.  
FILE CONTROL.  
    SELECT TAPE-FILE ASSIGN TO TAPE.  
DATA DIVISION.  
FILE SECTION.  
FD TAPE-FILE.  
01 TAPE-REC.  
    02 X PIC XXX.  
    02 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 [ USING identifier-1 ] ]  
  
{  
  AUDIT  
  NO-AUDIT  
} data-name-1  
  
[  
  ON EXCEPTION {  
    imperative-statement-1  
    conditional-statement-1  
    NEXT SENTENCE  
  }  
]
```

Explanation of Format Elements

Transaction Server -header-name-1

This option is used only with the Transaction Server and specifies the input header. You can use Transaction Server -header-name-1 to call the DCIENTRYPPOINT of a DCI library when your program detects an exception condition. This feature enables a program interfacing with the Transaction Server 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 DCIENTRYPPOINT.

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 “Enterprise Database Server 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.  
ENVIRONMENT DIVISION.  
INPUT-OUTPUT SECTION.  
FILE CONTROL.  
    SELECT TAPE-FILE ASSIGN TO TAPE.  
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, “Enterprise Database Server 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, the Enterprise Database Server 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 “Enterprise Database Server 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

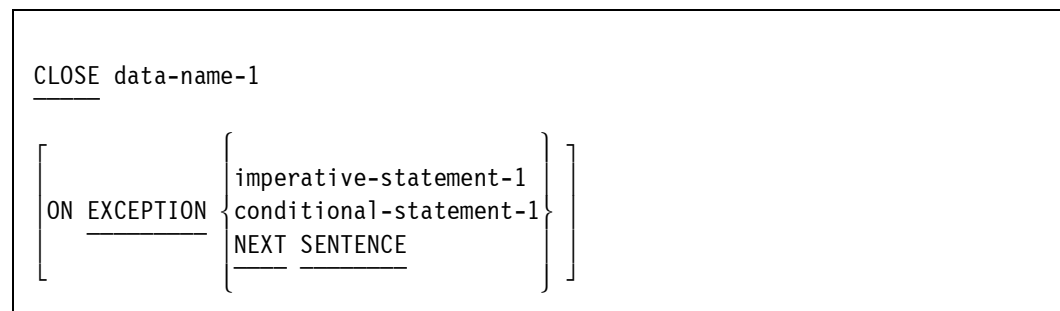
The CLOSE statement closes a database when your program requires no further access to it. The CLOSE statement is optional because the system closes any open database when the program terminates. A successfully closed database causes a syncpoint in the audit trail.

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.



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 "Enterprise Database Server 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 the Transaction Server for synchronized recovery, it is recommended that you do not use the ON EXCEPTION option. If the Enterprise Database Server 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, to close a database when you are using the Transaction Server with the Enterprise Database Server 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.
```

The following code checks for the ABORT condition, which is not caught by the ON EXCEPTION clause if it happens during the CLOSE operation.

```
IF DMSTATUS(DMCATEGORY) = 16
  OPEN MYDB
  GO TO ABORTED.
```

Later in this section, Example 4–1, “Enterprise Database Server 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.

<u>COMPUTE</u>	data-name-1	=	<table><tr><td>condition</td></tr><tr><td><u>TRUE</u></td></tr><tr><td><u>FALSE</u></td></tr></table>	condition	<u>TRUE</u>	<u>FALSE</u>
condition						
<u>TRUE</u>						
<u>FALSE</u>						

Explanation of Format Elements

data-name-1

This format element specifies the current record of a data set.

condition

If you specify a condition, Enterprise Database Server 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, “Enterprise Database Server 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 *Data and Structure Definition Language (DASDL) Programming 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 "Enterprise Database Server 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. If a database is an audited database, a STORE statement must be preceded by a BEGIN-TRANSACTION statement and followed by an END-TRANSACTION statement. If you do not want to store the record, you can nullify the CREATE statement with a subsequent FREE statement or with 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;  
RS RESTART DATA SET (RS-NUM NUMBER(3););
```

The following example uses the CREATE statement:

```
IDENTIFICATION DIVISION.  
PROGRAM-ID. EXAMPLE.  
ENVIRONMENT DIVISION.  
INPUT-OUTPUT SECTION.  
FILE CONTROL.  
  SELECT TAPE-FILE ASSIGN TO TAPE.  
DATA DIVISION.  
FILE SECTION.  
FD TAPE-FILE.  
01 TAPE-REC.  
  02 X PIC X(10).  
  02 Y PIC 9.  
  02 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 NO-AUDIT RS.
STORE D.
END-TRANSACTION AUDIT RS.
GO TO START-PRG.
```

Later in this section, Example 4–1, “Enterprise Database Server 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 “Enterprise Database Server 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.
```

```
ENVIRONMENT DIVISION.  
INPUT-OUTPUT SECTION.  
FILE CONTROL.  
    SELECT TAPE-FILE ASSIGN TO TAPE.  
DATA DIVISION.  
FILE SECTION.  
FD TAPE-FILE.  
01 TAPE-REC.  
    02 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, “Enterprise Database Server 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
  NO-AUDIT
} data-name-1 [ SYNC ]

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

Explanation of Format Elements

Transaction Server -header-name-1

This option specifies the Transaction Server output header. For information on Transaction Server, refer to Section 3, "Using the Transaction Server Program Interface."

The Transaction Server -header-name-1 option is used only with the Transaction Server. You can specify Transaction Server -header-name-1 to call the DCIENTRYPPOINT of a DCI library whenever you execute the statement. This feature enables a program interfacing with the Transaction Server 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 DCIENTRYPPOINT.

identifier-1

This format element specifies the message area passed to the DCIENTRYPPOINT.

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 "Enterprise Database Server 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.
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE CONTROL.
  SELECT TAPE-FILE ASSIGN TO TAPE.
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 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, “Enterprise Database Server 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 *Enterprise Database Server Application Programming Guide*.

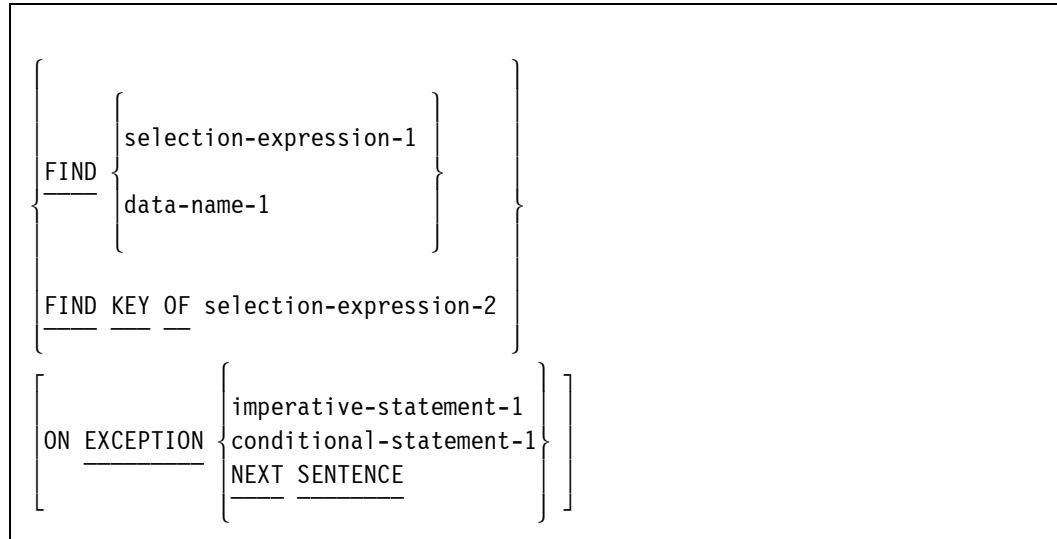
The FIND statement performs the following steps:

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.



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, the Enterprise Database Server 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 “Enterprise Database Server 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 = 1019
ON EXCEPTION
  MOVE 0 TO POP-OVR-65 (1019).
```

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, “Enterprise Database Server 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 *Data and Structure Definition Language (DASDL) Programming Reference Manual* and the *Enterprise Database Server 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, the Enterprise Database Server 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.

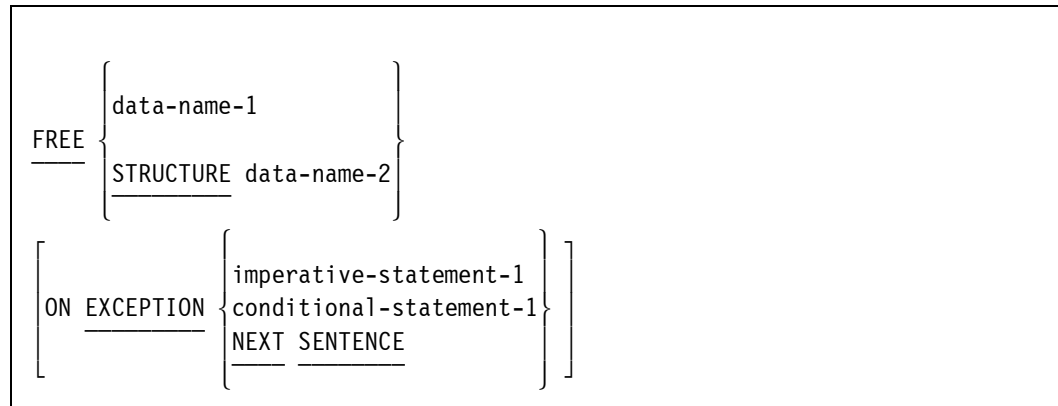
FIND, LOCK, and MODIFY statements using sets or subsets free the locked record only if a new record is retrieved. Other constructs that can free data set records are

- 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 “Enterprise Database Server 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

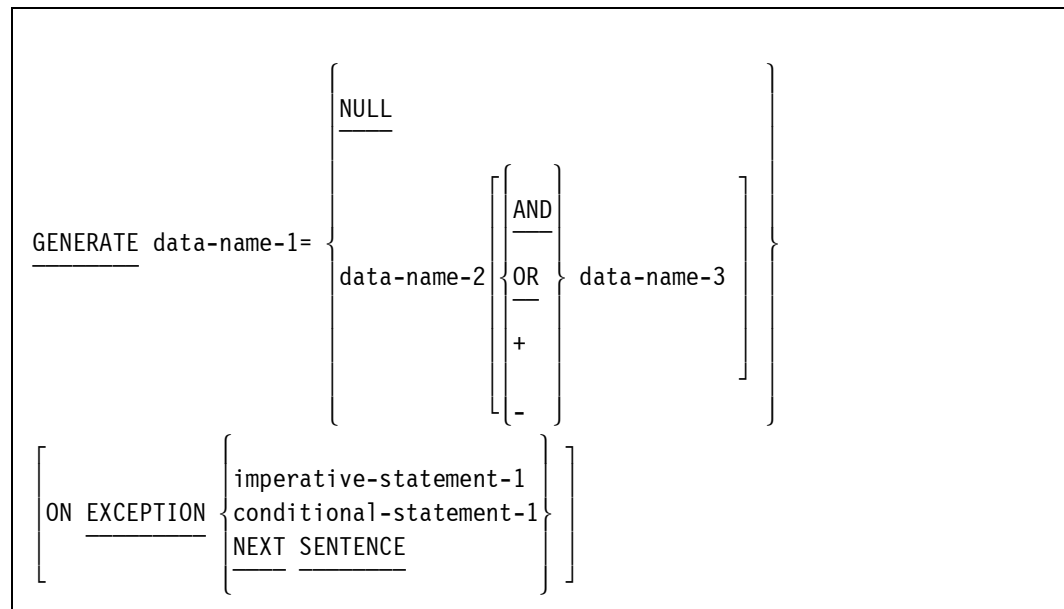
The GENERATE statement creates an entire subset in one operation. All subsets must be disjoint bit vectors. The GENERATE statement performs the following steps:

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: It is recommended 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 "Enterprise Database Server 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.  
ENVIRONMENT DIVISION.  
INPUT-OUTPUT SECTION.  
FILE CONTROL.  
    SELECT TAPE-FILE ASSIGN TO TAPE.  
DATA DIVISION.  
FILE SECTION.  
FD  TAPE-FILE.  
01  TAPE-REC.  
    02  S PIC XXX.  
    02  T PIC 9.  
    02  U PIC 999.  
    02  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, “Enterprise Database Server Sample Program,” shows the GENERATE statement within the context of a complete program.

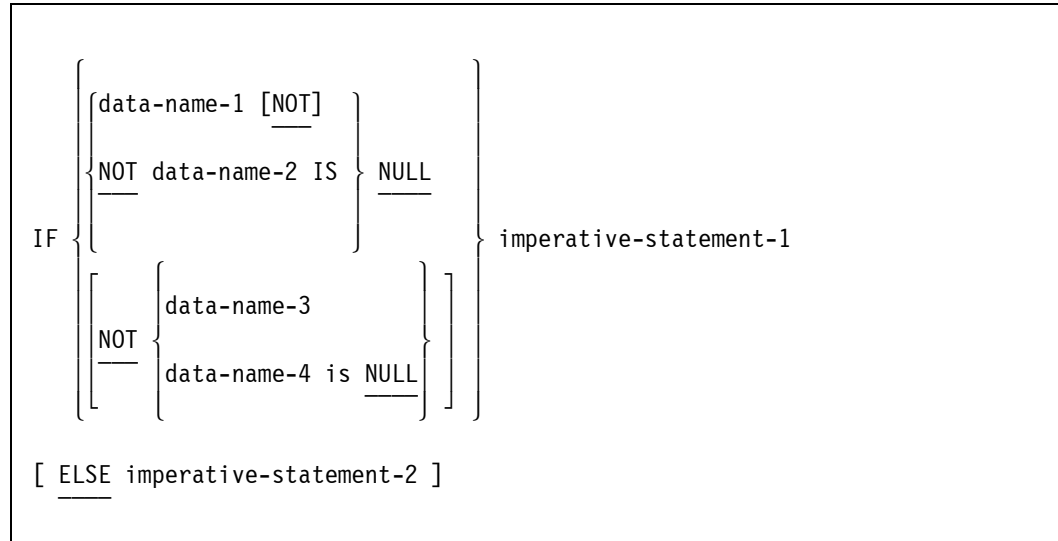
IF Statement

The IF statement tests an item to determine if it contains a NULL value.

Note: It is recommended 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.



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.
- Use of a DMVERB against the structure into which the link item points would cause a VERSIONERROR.

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, “Enterprise Database Server 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. If INDEPENDENTTRANS is set in DASDL, unconditionally locks the current record of the specified data set
2. Inserts the current record of the specified data set into the specified subset
3. 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 “Enterprise Database Server 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.
- INDEPENDENTTRANS is set in DASDL, you attempted to insert the specified record, and a deadlock occurred during the locking step.

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, “Enterprise Database Server 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.

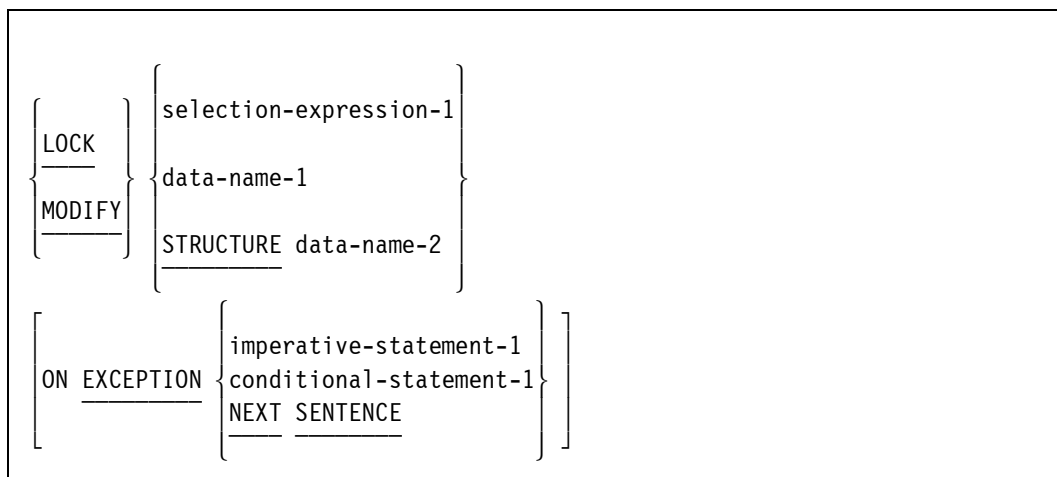
No other user can lock or secure the record once it is locked; therefore, the record must be freed when locking is no longer required. You can free a record explicitly by using a FREE statement or free a record implicitly by executing a subsequent LOCK, FIND, DELETE, CREATE, or RECREATE statement on 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.



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 in this section.

data-name-2

Data-name-2 must be a data set.

ON EXCEPTION

This option handles exceptions. Refer to “Enterprise Database Server 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 *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 ONON EXCEPTION option:

```
LOCK FIRST EMP AT DEPT-NO = 1019
ON EXCEPTION
  MOVE 0 TO POP-EMP (1019).
```

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, “Enterprise Database Server Sample Program,” shows the LOCK statement within the context of a complete program.

OPEN Statement

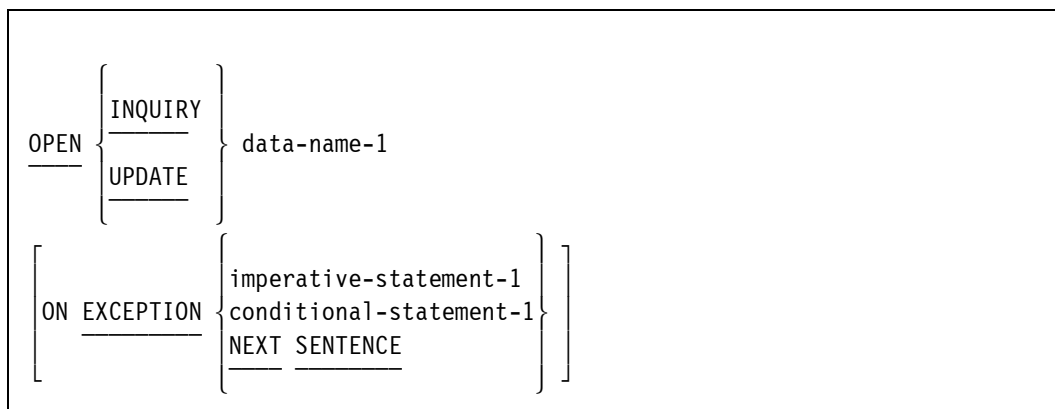
The OPEN statement opens a database for subsequent access and specifies the access mode. You must execute an OPEN statement before the database is first accessed; otherwise, the program terminates at run time with an invalid operator error.

The OPEN statement performs the following steps:

1. Opens an existing database. If files required for invoked structures are not in the system directory, the Enterprise Database Server 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

The Enterprise Database Server does not open any audit files if the OPENINQUIRY 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 "Enterprise Database Server 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, “Enterprise Database Server 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 *Enterprise Database Server 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 "Enterprise Database Server 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, "Enterprise Database Server 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. If INDEPENDENTTRANS is set in DASDL, locks the previously found record and then removes the record from the specified subset
4. Removes the record from the specified subset

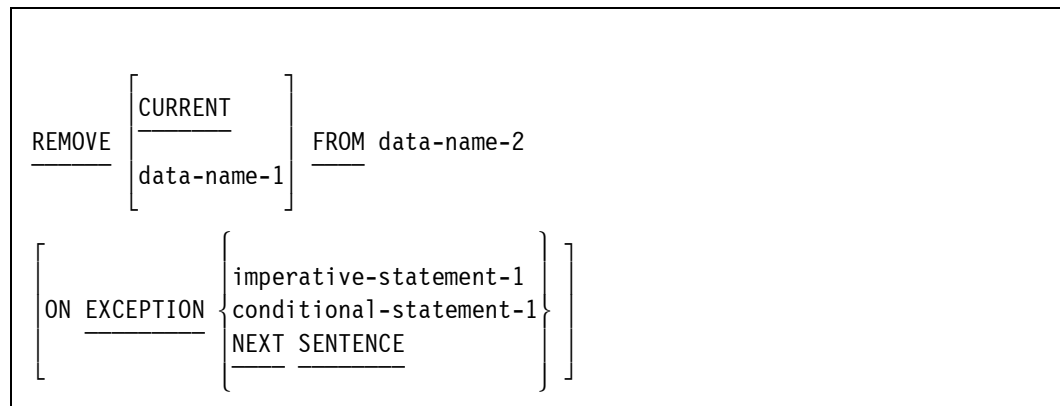
If the program returns an exception after step 2, the current path is invalid.

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.



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 "Enterprise Database Server 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.
- INDEPENDENTTRANS is set in DASDL, you attempted to remove the specified record, and a deadlock occurred during the locking step.

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, “Enterprise Database Server 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]

$$\left[\text{ON } \underline{\text{EXCEPTION}} \left\{ \begin{array}{l} \text{imperative-statement-1} \\ \text{conditional-statement-1} \\ \underline{\text{NEXT SENTENCE}} \end{array} \right\} \right]$$

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 "Enterprise Database Server Exceptions."

Example

The following line of code shows an example of the SAVE TRANSACTION POINT statement:

```
SAVE TRANSACTION POINT MY-RESTART 3.
```

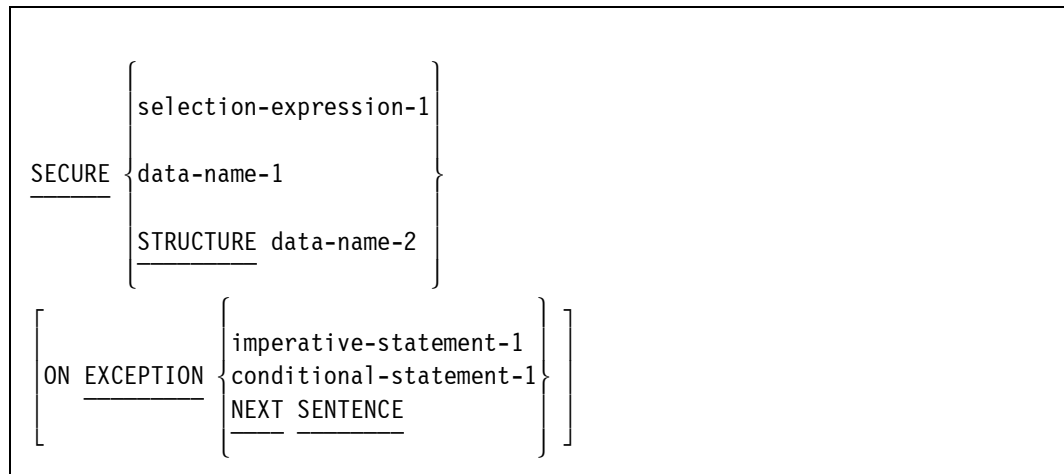
SECURE Statement

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.



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 "Enterprise Database Server 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.

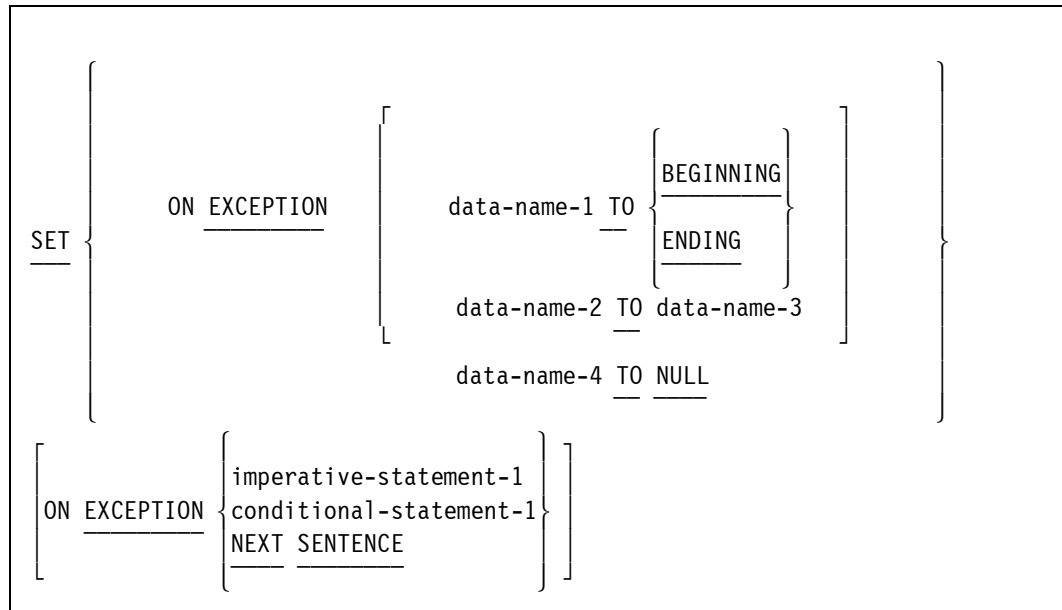
The SET statement performs the following steps:

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 "Enterprise Database Server 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, "Enterprise Database Server 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, the STORE statement performs the following steps:

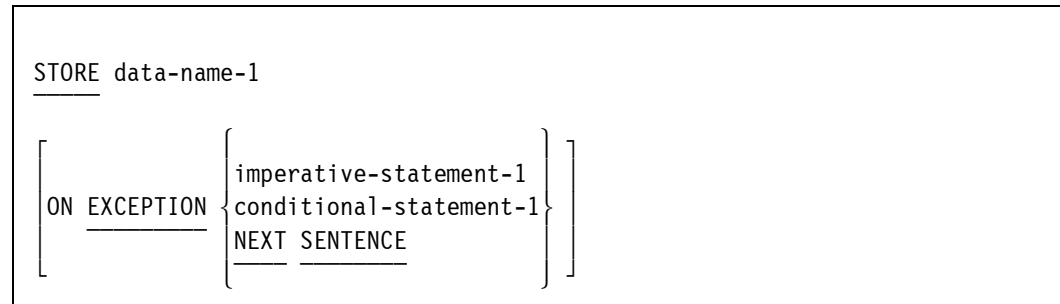
- Ensure the validity of the data in the user work area if you have specified a VERIFY condition in DASDL.
- Test the record for validity before the STORE statement inserts the record into each set in the data set. For example, the STORE statement can test the record to determine whether or not duplicate 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.



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 "Enterprise Database Server 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, "Enterprise Database Server Sample Program," shows the STORE statement within the context of a complete program.

Processing Enterprise Database Server 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.

Applications that utilize the Open Distributed Transaction Processing (DTP) system might encounter an AUDITERROR 1 exception while performing some database operations. This can occur when the program uses multiple databases and enters into a global transaction, goes into transaction state for one database, and attempts to SECURE or LOCK a record in another database. Note that a FIND on another database is acceptable.

This error indicates that the program is already in a transaction state. The error occurs because the DTP software coordinates record ownership through its own transactions by using its RX-GLOBAL-TR data set, as the transaction context migrates amongst participating application programs. The error is the result of attempting to be in transaction state for two databases at the same time.

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.

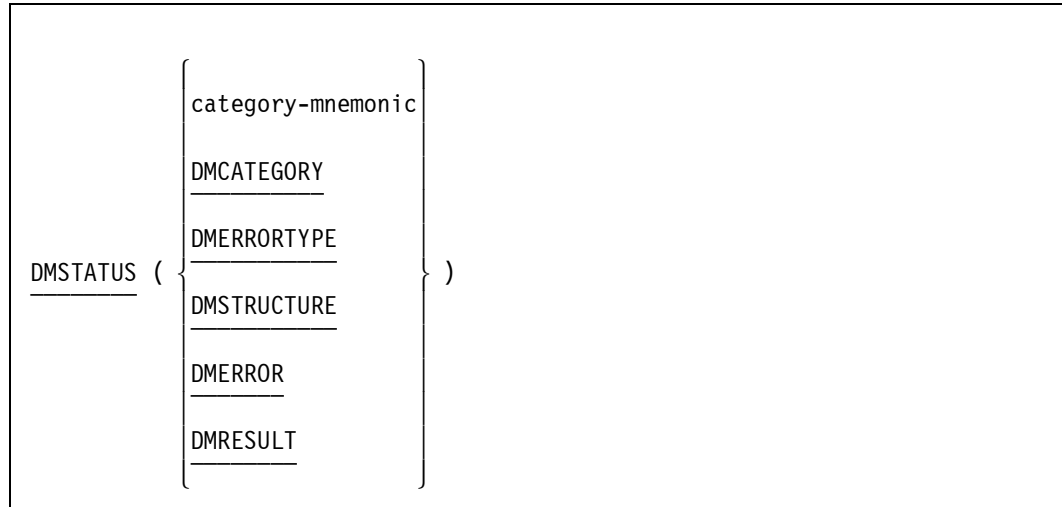
DMSTATUS Database Status Word

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 *Enterprise Database Server 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 the Enterprise Database Server at run time; it is not necessarily the same value every time.

General Format

The general format for the DMSTRUCTURE function is as follows.

<pre>data-name-1 (<u>DMSTRUCTURE</u>)</pre>

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".
```

Enterprise Database Server 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 Enterprise Database Server 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.

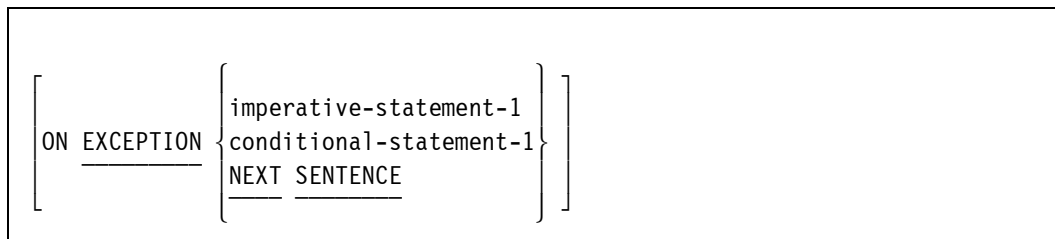
ABORT-TRANSACTION	INSERT
ASSIGN	LOCK
BEGIN-TRANSACTION	MODIFY
CANCEL TRANSACTION POINT	OPEN
CLOSE	RECREATE
CREATE	REMOVE
DELETE	SAVE TRANSACTION POINT
END-TRANSACTION	SECURE
FIND	SET
FREE	STORE
GENERATE	

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.



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
```

Enterprise Database Server Sample Program

Example 4–1 shows methods of using the Enterprise Database Server 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 UNIV 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 UNIV.
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.

```

Example 4-1. Enterprise Database Server Sample Program

```
014600      OPEN OUTPUT MONITOR-DMS.
014700      MOVE SPACES TO MONITOR-REC.
014800      OPEN UPDATE UNIV.
014900      BEGIN-TRANSACTION AUDIT RSTDS.
015000      PERFORM CREATE-ADR THRU CREATE-ADR-EXIT.
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
```

Example 4-1. Enterprise Database Server Sample Program

```
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
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"
```

Example 4-1. Enterprise Database Server Sample Program

```
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"
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
```

Example 4-1. Enterprise Database Server Sample Program

```
028400 400-LINK-MSF-TO-ADR.
028500     LOCK MSFSET AT SSNO = C-SSNO.
028600     LOCK SSAD AT SNO OF ADR = C-SSNO.
028700     ASSIGN ADR 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 SMARTANDSEXSET = SMART AND SEXSET.
030300     FIND FIRST SMARTANDSEXSET ON EXCEPTION
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 SMARTANDSEXSET = 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.
```

Example 4-1. Enterprise Database Server Sample Program

```
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"
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(DUPPLICATES)
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)
```

Example 4-1. Enterprise Database Server Sample Program

```
037600      MOVE "IOERROR" TO MONITOR-STATUS-B ELSE
037700      IF DMSTATUS(LIMITERROR)
037800      MOVE "LIMITERROR" TO MONITOR-STATUS-B ELSE
037900      IF DMSTATUS(OPENERERROR)
038000      MOVE "OPENERERROR" 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(AUDITERERROR)
038800      MOVE "AUDITERERROR" 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 UNIV ON EXCEPTION
039900      STOP RUN.
040000      XIT.
040100      STOP RUN.
```

Example 4–1. Enterprise Database Server Sample Program

Section 5

Using the Enterprise Database Server TPS Program Interface

The transaction processing system (TPS) provides users of the Enterprise Database Server 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 processing 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 Enterprise Database Server 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 “Enterprise Database Server TPS Extensions” in Section 1, “Introduction to COBOL74 Program Interfaces.”

See Also

Refer to the *DMSII 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*. This manual describes the way 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.

Refer to identifiers in a COBOL74 program 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.

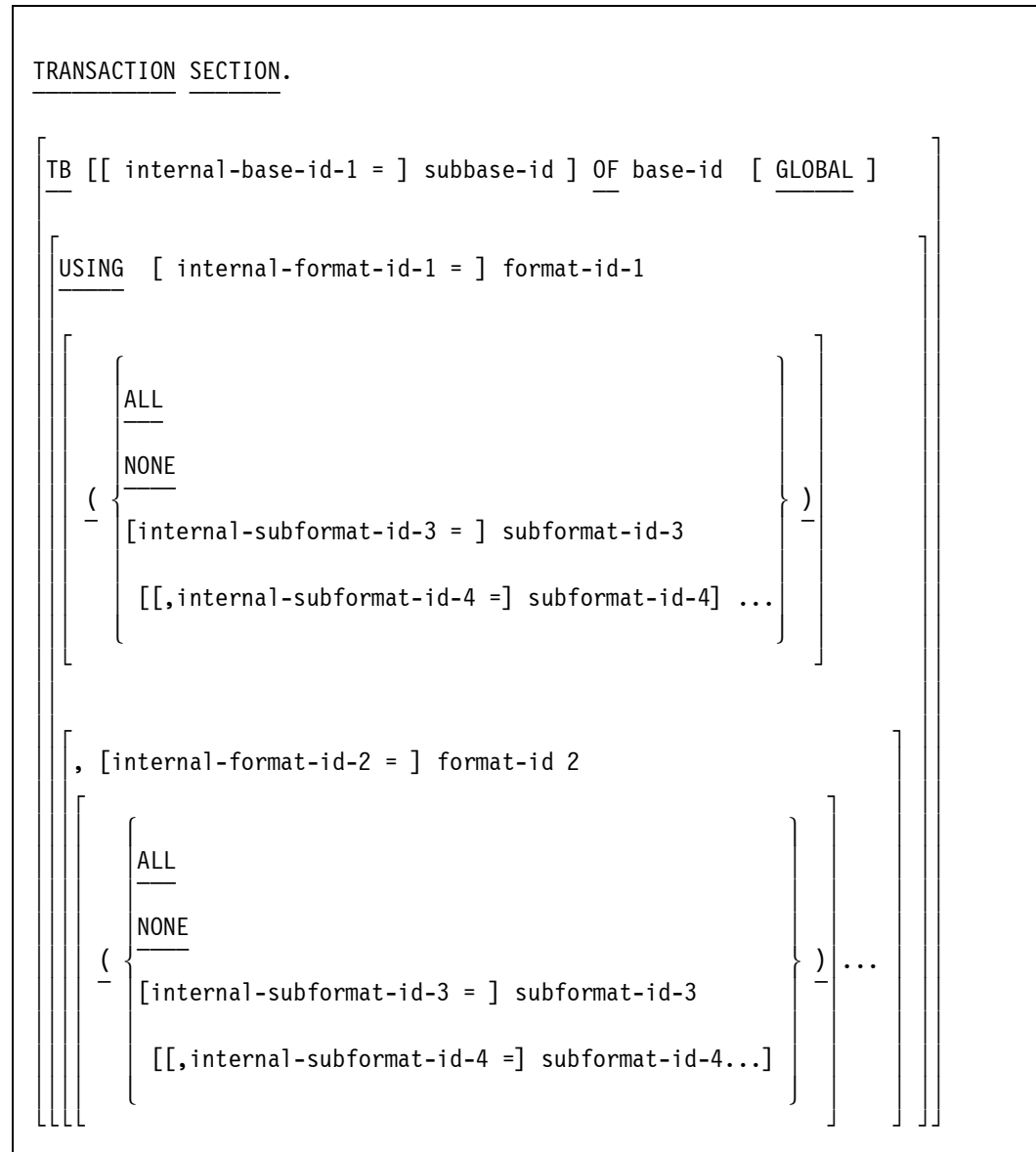
The program can designate alternative internal names for the transaction base and for any of the formats or subformats declared. If alternative internal names are used for the base name, subbase name, format name, or subformat name, the program must reference these internal identifiers rather than the 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.



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.  
01  TRIN.  
01  TROUT.  
01  LASTINPUT.  
01  RESTARTTRREC.  
01  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:

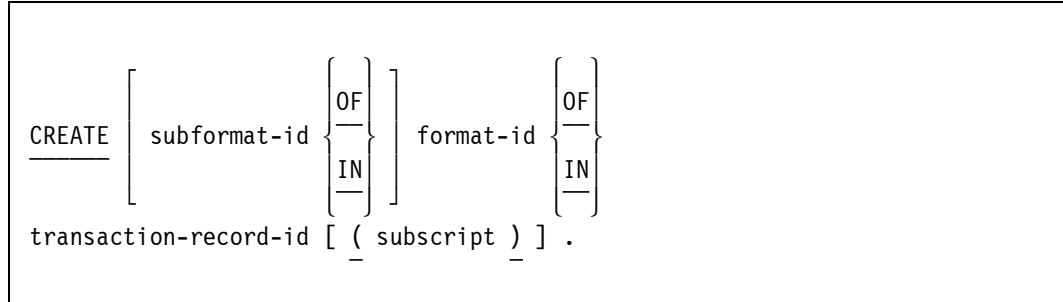
- Assigns to the record variable the initial values of all items in the transaction record format and subformat, if given. The items that are assigned are all declared in TFL.
- Initializes the transaction record control items.

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.



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

- 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:

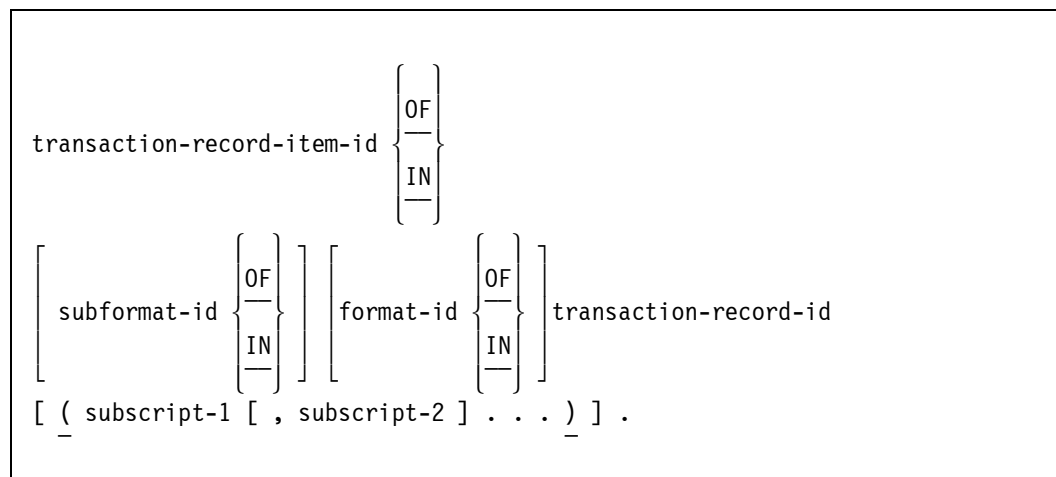
- A transaction record name alone is not considered a legitimate group name. Instead, a transaction record format name, a transaction subformat name, or a subordinate group data item must be referenced.
- When a transaction record format is referenced, only the data items in the common portion of the 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.



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	<p>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,</p> <p>DATAITEMNAME OF TRANREC</p>
Format name and data item name	<p>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</p> <p>DATAITEMNAME OF FORMATNAME IN TRANREC</p>
Subformat name and data item name	<p>Both the subformat name and the data item name are needed for qualification when all the following conditions prevail:</p> <ul style="list-style-type: none">• 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. <p>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</p> <p>DATAITEMNAME OF SUBFORMATNAME IN TRANREC</p>

Item	Qualification Requirement
Format name, subformat name, and data item name	<p>The format name, subformat name, and data item name are needed for qualification when all the following conditions prevail:</p> <ul style="list-style-type: none">• 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. <p>For example</p> <p>DATAITEMNAME OF SUBFORMATNAME OF FORMATNAME IN TRANREC</p>

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.

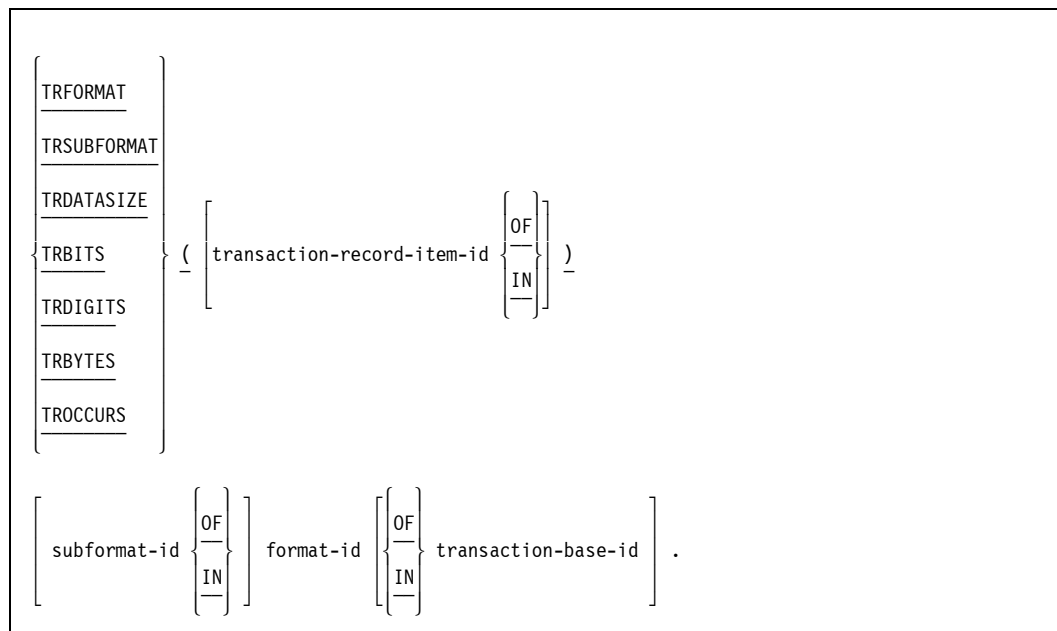
The compile-time functions are discussed in detail in the *DMSII TPS Programming Guide*. The format shown in the following text provides the syntax you use to access the functions.

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	Returns the . . .
TRFORMAT format-id	Numeric value assigned to the format-id.
TRSUBFORMAT subformat-id OF format-id	Numeric value assigned to the subformat-id.
TRDATASIZE format-id	Size, in bytes, of the common portion of the transaction record.
TRDATASIZE subformat-id OF format-id	Size, in bytes, of the subformat record.
TRBITS transaction-record-item-id OF format-id	Size, in bits, of the referenced data item.
TRBITS transaction-record-item-id OF subformat-id OF format-id	Size, in bits, of the referenced data items.
TRDIGITS transaction-record-item-id OF format-id	Size, in digits, of the referenced data item.
TRDIGITS transaction-record-item-id OF subformat-id OF format-id	Size, in digits, of the referenced data items.
TRBYTES transaction-record-item-id OF format-id	Size, in bytes, of the referenced data item.
TRBYTES transaction-record-item-id OF subformat-id OF format-id	Size, in bytes, of the referenced data items.
TROCCURS transaction-record-item-id OF format-id	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	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 the Enterprise Database Server 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 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 use of 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

Table 5–3. TPS Entry Points

Entry Point	Parameters
RETURNLASTRESPONSE	IDNUM, TROUT
RETURNRESTARTINFO	IDNUM, TROUT
SEEKTRANSACTION	FILENUMBER, BLOCKNUMBER, OFFSETS
SWITCHTRFILE	(No 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 Enterprise Database Server 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)\vfi1\
```

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 Enterprise Database Server 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 the Enterprise Database Server 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 the Enterprise Database Server and information on handling exceptions, refer to Section 4, “Using the Enterprise Database Server Program Interface.”
- For information about exception categories that are specific to the TPS transaction processing extensions, see the *DMSII TPS Programming Guide*.

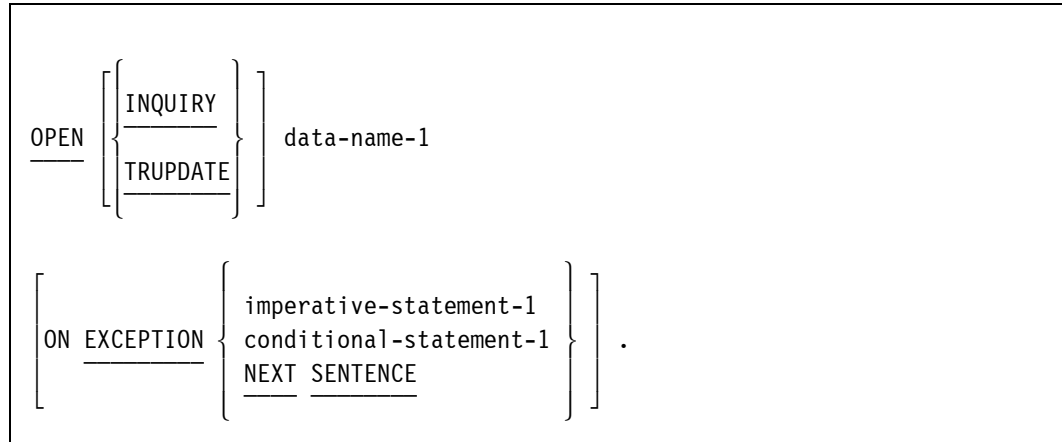
OPEN Statement

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.



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 Enterprise Database Server Exceptions" in Section 4, "Using the Enterprise Database Server 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 Access routines.

General Format

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

<pre>BEGIN-TRANSACTION (transaction-record-variable-id) restart-data-set-name [ON EXCEPTION { imperative-statement-1 conditional-statement-1 NEXT SENTENCE }] .</pre>
--

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 Enterprise Database Server Exceptions" in Section 4, "Using the Enterprise Database Server 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 Enterprise Database Server 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 Enterprise Database Server 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 Enterprise Database Server Exceptions" in Section 4, "Using the Enterprise Database Server 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,  
saveresponse-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.

saveresponse-tr-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 Enterprise Database Server 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 Enterprise Database Server Exceptions" in Section 4, "Using the Enterprise Database Server 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 the Enterprise Database Server 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.

Enterprise Database Server 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.
00      TRANDATE REAL;
        OLD-BALANCE REAL(S10,2);
        AMOUNT REAL(S10,2);    % NEGATIVE FOR WITHDRAWAL
        NEW-BALANCE REAL(S10,2);

```

Example 5–1. DASDL Description for Banking Transaction Program

```

);
);
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  
    ),  
),
```

Example 5–2. TFL Description for Banking Transaction Program


```
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
(
    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;);
);
```

Example 5–2. TFL Description for Banking Transaction Program

```
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,
    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.
```

Example 5-2. TFL Description for Banking Transaction Program

```
(
    USERCODE = SAMPLEUSER,
    FAMILY = PACK
),
DATA FILE
(
    USERCODE = SAMPLEUSER,
    DUPLICATED ON DISK
);
```

Example 5–2. TFL Description for Banking Transaction Program

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).
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.
```

Example 5–3. Update Library for Banking Transaction Program

```
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 CLOSEDDB.
003800  UPDATELIB-EXIT.
003900  EXIT PROGRAM.
04300 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.
006900*****
007000  CLOSE DBASE.
007100*
007200*
007300 CLOSEDDB.
007400*****
007500 CLOSE DBASE.
007600*
007700*
```

Example 5-3. Update Library for Banking Transaction Program

```
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 01210
012100     ERR.
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.
```

Example 5-3. Update Library for Banking Transaction Program

```
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 update routine performs withdrawals against 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

```
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

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.
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 STATUS=STATUS, CREATEACCT, RESTARTDETANKER,
003800     DEPOSIT, WITHDRAWAL, PURGEACCT, SYSTEMTR.
003900 01  TRIN.
004000 01  TROUT.
```

Example 5-4. Banking Transaction Program


```

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.
007700 MAIN SECTION.
007800 P1.
007900    OPEN I-O REMOTEFIL.
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.

```

Example 5-4. Banking Transaction Program

```
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.
012800     PERFORM WRITE-RMT.
012900     READ REMOTEFILE INTO SP; INVALID KEY GO TO EXT.
013300     IF SP = "PURGE"           PERFORM PURGEP           ELSE
013400     IF SP = "DEPOSIT"        PERFORM DEPOSITP          ELSE
013000     IF SP = "CREATEUSER"     PERFORM CREATEUSERP       ELSE
```

Example 5-4. Banking Transaction Program

```

013100      IF SP = "PURGEUSER"      PERFORM  PURGEUSERP      ELSE
013200      IF SP = "CREATE"          PERFORM  CREATEP        ELSE 0135
00      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 REMOTEFIL 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 PURGE.
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

```
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 REMOTEFIL 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 REMOTEFIL 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.
```

Example 5-4. Banking Transaction Program

```
022300*
022400 NEWUSERP.
022500*****
022600     MOVE "FUNCTION IS NEW USER" TO RMT-REC.
022700     PERFORM WRITE-RMT.
022800     MOVE "USER ID ? (X17)  " TO RMT-REC.
022900     PERFORM WRITE-RMT.
023000     READ REMOTEFIL 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 REMOTEFIL 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 REMOTEFIL 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 REMOTEFIL 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 REMOTEFIL INTO TEMP      ; INVALID KEY GO TO EXT.
026700     MOVE TEMP TO OFFSETS.
```

Example 5-4. Banking Transaction Program

```
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.
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 > 0 GO TO ERR.
029400      PERFORM LOGONTRUSER.
029500      IF TPS-RESULT > 0 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 > 0 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*
```

Example 5-4. Banking Transaction Program

```
031400 SWITCHP.
031500*****
031600     MOVE "FUNCTION IS SWITCH" TO RMT-REC.
031700     PERFORM WRITE-RMT.
031800     PERFORM SWITCHTRFILE.
031900     IF TPS-RESULT > 0 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.
033000     READ REMOTEFIL 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.
```

Example 5-4. Banking Transaction Program

```
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 PROCESSTRANSACT.
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*****
038100      IF OPT NOT EQUAL TANKING  GO TO GETLASTRESTRREC.
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      GETLASTRESTRREC.
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.
```

Example 5-4. Banking Transaction Program


```
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
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.
```

Example 5-4. Banking Transaction Program

```
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*
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

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.
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*

```

Example 5-5. Declaring the Transaction Base and Entry Points

```
004100 CREATETRUSER.
004200     CALL "CREATRUSER           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"
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*
```

Example 5-5. Declaring the Transaction Base and Entry Points

```
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

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 *Screen Design Facility (SDF) Operations and Programming Guide*.

You can use SDF with the Advanced Data Dictionary System (ADDS) and with the Transaction Server.

When you use SDF with ADDS, you can take advantage of the following ADDS capabilities:

- Defining prefixes for entities, including the Enterprise Database Server database elements, COBOL74 data description items, 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 Transaction Server 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 Transaction Server Program Interface.”

Identifying the Dictionary

The dictionary is identified in the SPECIAL-NAMES paragraph of the program, using the DICTONARY statement.

Note: *A program can invoke only one dictionary. Therefore, if a program accesses SDF forms and any other dictionary, then both must be in the same dictionary.*

General Format

The general format of the DICTONARY 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 DICTONARY statement to identify the SCREENDESIGN function name:

```
001000 IDENTIFICATION DIVISION.  
007000  
008000 ENVIRONMENT DIVISION.  
009000 CONFIGURATION SECTION.  
011100  
011200 SPECIAL-NAMES.  
011300    DICTONARY 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.

; (semicolon)

This option, the semicolon following the ending period, controls the behavior of compiler control records (CCRs) and format listings. The semicolon must be separated from the ending period of the data invocation by at least one space.

If a CCR immediately follows a data item invoked from the dictionary, the compiler control option changes might occur before the compiler processes the information invoked from the dictionary. You can avoid this situation by using the optional semicolon after the ending period. The semicolon ensures that the compiler processes the invoked information before the compiler control option changes.

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:

```
016000 DATA DIVISION.  
017000 FILE SECTION.  
018000 FD REMFILE.  
021200 01 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 Transaction Server error message. For information on how to process Transaction Server errors, refer to the *Transaction Server 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.  
    01 TEMPSAMPLEFORM PIC X(2000).  
.   
.   
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.

<u>WRITE</u> <u>FORM</u>	{	record-name <u>USING</u> form-name-2 form-name-1	}	[FROM	{	identifier-1 <u>DEFAULT FORM</u>	}]
[; <u>ON ERROR</u> 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 48-bit binary number uniquely identifying the form name for programmatic purposes. This function is required for using the data communications interface (DCI) library with the Transaction Server interface.

When using SDF with Transaction Server, you must perform these steps before the program can execute a SEND statement.

- Use the FORM-KEY function to move the form key into the first word of the conversation area of output.
- Declare the location to which the value produced by this function is moved USAGE REAL. Other USAGES will almost certainly cause data corruption. The value produced by the FORM-KEY function is a collection of 48 binary bits, in which both position and value are significant, and those characteristics are preserved only with USAGE REAL.

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 48-bit 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 Transaction Server is provided later in this section in "Using SDF with Transaction Server."
- Refer to Section 3, "Using the Transaction Server Program Interface," for information about the extensions developed for the Transaction Server.

Examples

The following is an example of the function syntax within the MOVE statement. The item SAVEKEY must be declared USAGE REAL. 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 Transaction Server interface to move the form key SDFFORM to the first word of the conversation

area. If the item COMS-OUT-CONVERSATION is an elementary item, it must be declared USAGE REAL.

```
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

```
02 EMPLOYEE-CURSOR
```

In the program, you can assign a value for the flag setting, as shown in the following example:

```
01 CONSTANTS.  
  02 TRUE-FLAG      PIC 9 COMP VALUE 1.  
  02 FALSE-FLAG     PIC 9 COMP VALUE 0.
```

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 form library (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 information.

Table 6–2. Programmatic Flag Information

Flag	COBOL74 Picture Representation	Valid Values
Cursor	PIC 9(1) COMP	<ul style="list-style-type: none"> • 0 (No cursor positioning) • 1 (Position cursor)
Data only	PIC 9(1) COMP	<ul style="list-style-type: none"> • 0 (No data only) • 1 (Data only)
Field suppress	PIC 9(1) COMP	<ul style="list-style-type: none"> • 0 (Not suppressed) • 1 (Suppressed)
Flag groups	Not applicable	Not applicable

Table 6–2. Programmatic Flag Information

Flag	COBOL74 Picture Representation	Valid Values
Highlight	PIC 9(1) COMP	Fixed highlighting <ul style="list-style-type: none"> • 0 (Not specified) • 1 (Specified) Variable highlighting <ul style="list-style-type: none"> • 0 (None) • 1 (Bright) • 2 (Reverse) • 3 (Secure) • 4 (Underline) • 5 (Blink)
Input/Output	PIC 9(1) COMP	<ul style="list-style-type: none"> • 0 (Input and output) • 1 (Input and output) • 2 (Input only) • 3 (Output) • 4 (Output) transmittable
No input	PIC 9(1) COMP	<ul style="list-style-type: none"> • 0 (Data input) • 1 (No data input)
Page	PIC 9(1) COMP	Terminal page 1 - 9
Specify	PIC 9(4) COMP	<ul style="list-style-type: none"> • 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.
```

Using the SDF Program Interface

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.

```
01 FORM-1.
  02 FIELD-1                                PIC X(10).
  02 FIELD-2                                PIC 9(6) V9(2).
  02 FORM-1-FLAGS.
    03 FORM-1-PAGE-FLAGS.
      04 FORM-1-PAGE.                      PIC 9 COMP.
    03 FORM-1-SPECIFY-FLAGS.
      04 FORM-1-SPECIFY                    PIC 9(4) COMP.
      04 FIELD-1-SPECIFY                   PIC 9(4) COMP.
      04 FIELD-2-SPECIFY                   PIC 9(4) COMP.
    03 FORM-1-IOTYPE-FLAGS.
      04 FIELD-1-IOTYPE                    PIC 9 COMP.
      04 FIELD-2-IOTYPE                    PIC 9 COMP.
    03 FORM-1-CURSOR-FLAGS.
      04 FIELD-1-CURSOR                    PIC 9 COMP.
      04 FIELD-2-CURSOR                    PIC 9 COMP.
    03 FORM-1-SUPPRESS-FLAGS.
      04 FIELD-1-SUPPRESS                  PIC 9 COMP.
      04 FIELD-2-SUPPRESS                  PIC 9 COMP.
    03 FORM-1-HIGHLIGHT-FLAGS.
      04 FIELD-1-HIGHLIGHT                 PIC 9 COMP.
      04 FIELD-2-HIGHLIGHT                 PIC 9 COMP.
    03 FORM-1-DATA-FLAGS.
      04 FORM-1-DATA                      PIC 9 COMP.
      04 FIELD-1-DATA                     PIC 9 COMP.
      04 FIELD-2-DATA                     PIC 9 COMP.
    03 FORM-1-NOINPUT-FLAGS.
      04 FIELD-1-NOINPUT                   PIC 9 COMP.
      04 FIELD-2-NOINPUT                   PIC 9 COMP.
```

Using SDF with Transaction Server

You can use SDF with Transaction Server to take advantage of Transaction Server 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 Transaction Server 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 Transaction Server 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 Transaction Server, 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 Transaction Server input message.

- It is recommended 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 Transaction Server header name:

```
006300     RECEIVE COMS-IN MESSAGE INTO COMS-MESSAGE-AREA.  
006310*     CHECK FOR COMS ERRORS
```

Once the Transaction Server 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 Transaction Server error checking, refer to the *Transaction Server 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 Transaction Server trancode. The corresponding Transaction Server 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 Transaction Server transcodes, 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.
```

- It is recommended 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 Transaction Server 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:

```
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. The Transaction Server 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 *Transaction Server Programming Guide* for detailed information on the use of the Transaction Server direct-window interface and for an illustration of the use of SDF and Transaction Server in an application.
- For information about the extensions used with Transaction Server, refer to the Section 3, "Using the Transaction Server 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 form library 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 "SCREENDSIGN".
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
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
```

Example 6-1. SDF Sample Program with a READ Statement

```

000140 02 ACTION PIC X(10). D
000150 02 ACCOUNT-NO PIC 9(9). D
000160 02 NAME PIC X(15). D
000170 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

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 form library was created using SDF. Sections of the program are explained in comment lines within the code.

```

11000 IDENTIFICATION DIVISION.
11100
12000 ENVIRONMENT DIVISION.
13000 CONFIGURATION SECTION.
13200
13300*****
13400* Specify the dictionary that stores the form library. *
13500*****
13600
14000 SPECIAL-NAMES.
15000 DICTIONARY IS "SCREENDESIGN".

```

Example 6–2. SDF Sample Program with READ and WRITE Statements

```

16000
16100*****
16200* Declare the remote file. *

```

```
16300*****
16400
17000 INPUT-OUTPUT SECTION.
18000 FILE CONTROL.
19000   SELECT REMFILE ASSIGN TO REMOTE.
19100
19200*****
19300* Invoke the formlibrary in the FILE SECTION, and associate      *
19400* it with the file. This action also causes the proper maximum*
19500* record size (MAXRECSIZE) to be designated for the file.      *
19600*****
19700
21000 DATA DIVISION.
22000 FILE SECTION.
23000 FD REMFILE.
24000 01 SAMPLELIB FROM DICTIONARY.
24100
24200*****
24500* Include program record descriptions for all forms in the      *
24600* formlibrary that are automatically invoked and copied into the *
24700* program during compilation (see the following dictionary data  *
24800* lines identified with the D flag).                               *
24900*****
24950
00100 *-DICTIONARY                                          D
00110 *-DICTIONARY FORMLIST <SAMPLELIB>.                    D
00120 01 SAMPLEFORM1.                                       D
00130   02 ACTION PIC X(10).                                D
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.                   *
```

Example 6–2. SDF Sample Program with READ and WRITE Statements

```
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

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*****

```

Example 6–3. SDF Sample Program with Programmatic Controls

```

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
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

```

Example 6–3. SDF Sample Program with Programmatic Controls

```

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*****
003610 MOVE 0 TO VRNAME-CURSOR
003620 MOVE 0 TO VRNAME-HIGHLIGHT
003700
003800 *****
003900 * Perform a loop to enter and store data in a disk file. *
004000 *****

```

Example 6-3. SDF Sample Program with Programmatic Controls

```

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
006200 MOVE 1 TO VRNAME-CURSOR.
006300
006400 END-DATA-ENTRY.
```

Example 6-3. SDF Sample Program with Programmatic Controls

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 "SCREENDSIGN".
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. 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

```

Example 6-4. SDF Sample Program with Message Keys

```

002600*****
002700*   Open remote file I/O.                                           *

```

```

002800*****
002900 OPEN I-O REMFILE.
003000
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 *
004140* 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

```

Example 6-4. SDF Sample Program with Message

```

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 *
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
```

Example 6-4. SDF Sample Program with Message

```
005320*****
005325* Insert code to modify an existing  *
005330* item.                               *
005335*****
005340
005400STOP RUN.
```

Example 6–4. SDF Sample Program with Message Keys

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 *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 formlibraries to SDF Plus, refer to the *Screen Design Facility Plus (SDF Plus) Installation and Operations Guide*. Consult the *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), and the Transaction Server. 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;” and Section 3, “Using the Transaction Server Program Interface” in this volume.

Understanding Interface Elements

The interface between COBOL74 application programs and SDF Plus formlibraries can be achieved through either the remote file or the Transaction Server interface. If you use the remote file interface, you can interact with SDF Plus applications by means of remote files. If you use the Transaction Server interface, you can interact with SDF Plus applications through the Transaction Server windows, and have access to all Transaction Server 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 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 DICTONARY clause.

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

General Format

The general format for the DICTONARY clause is as follows:

```
[, DICTONARY IS literal-1]
```

Explanation of Format Elements

DICTONARY

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 DICTONARY 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.

; (semicolon)

This option controls the behavior of compiler control records (CCRs) and format listings. The semicolon must be separated from the ending period of the data invocation by at least one space.

If a CCR immediately follows a data item invoked from the dictionary, the compiler control option changes might occur before the compiler processes the information invoked from the dictionary. You can avoid this situation by using the optional semicolon after the ending period. The semicolon ensures that the compiler processes the invoked information before the compiler control option changes.

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.  
01 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.  
01 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.  
01 SAMPLELIB FROM DICTIONARY.  
01 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.  
01 REM-REC PIC X(2400).  
WORKING-STORAGE SECTION.  
01 SAMPLELIB FROM DICTIONARY; SAME RECORD AREA.  
01 SAMPLELIB4 FROM DICTIONARY;  
    REDEFINES SAMPLELIB.  
01 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.

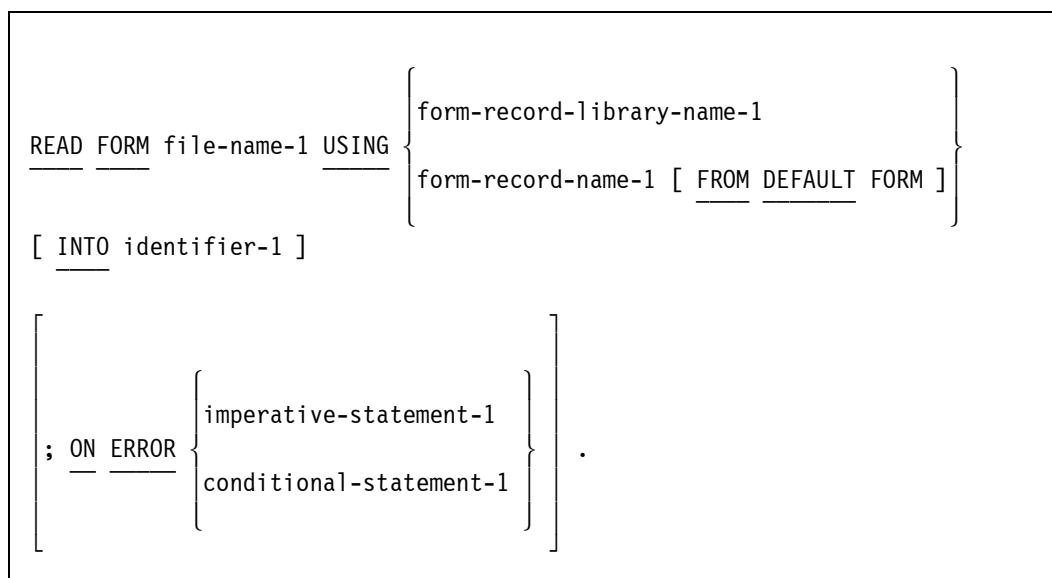
The appropriate form record library reads the form record, validates the record, performs screen error handling, and returns to the program the valid form record or any detected error condition.

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:



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.
01 TEMPSAMPLEFORM PIC X(2400).
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.

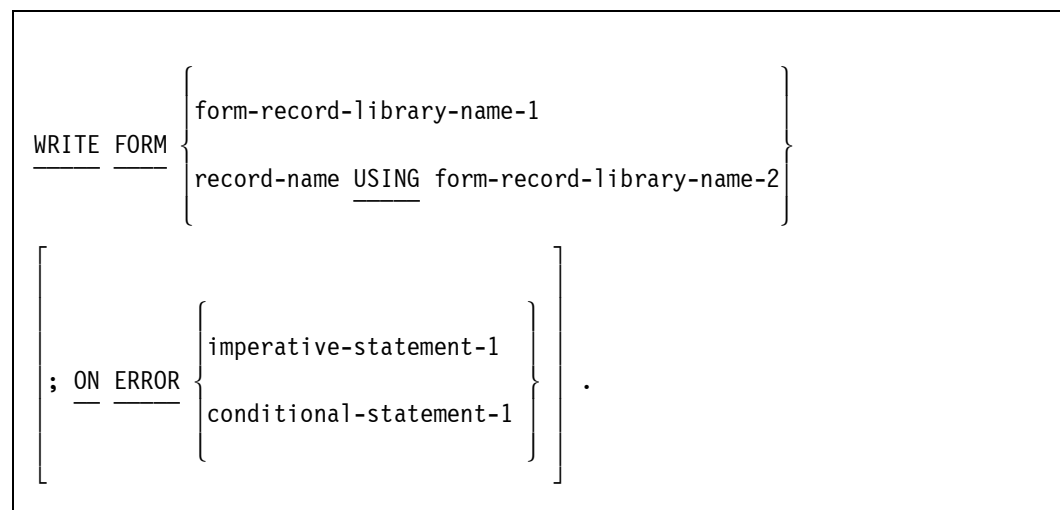
The appropriate form record library performs the write operation. Any detected error condition is returned to the program when the WRITE FORM statement is performed.

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:



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 SAMERECORD 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:



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.

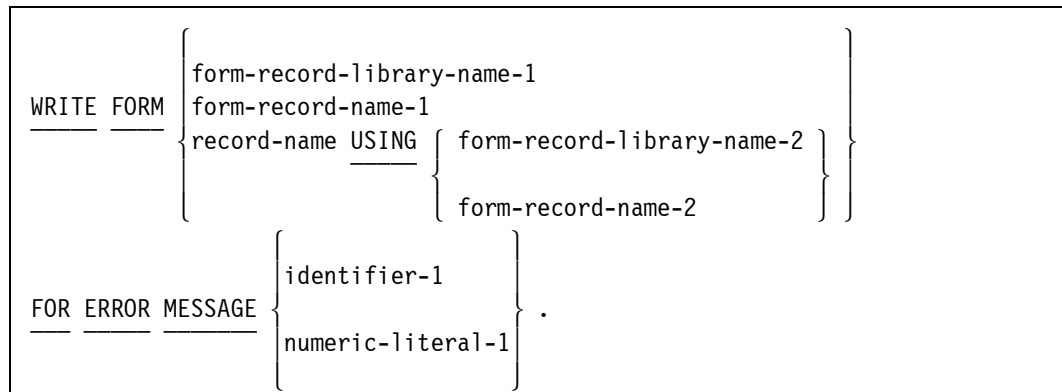
Each WRITE FORM FOR ERROR MESSAGE statement must be followed by an additional WRITE FORM statement, either format 1 or format 2, in order to initiate SDF Plus transaction error processing. At that point, SDF Plus redisplay the form that resulted in the transaction error, including the user's data, any 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)

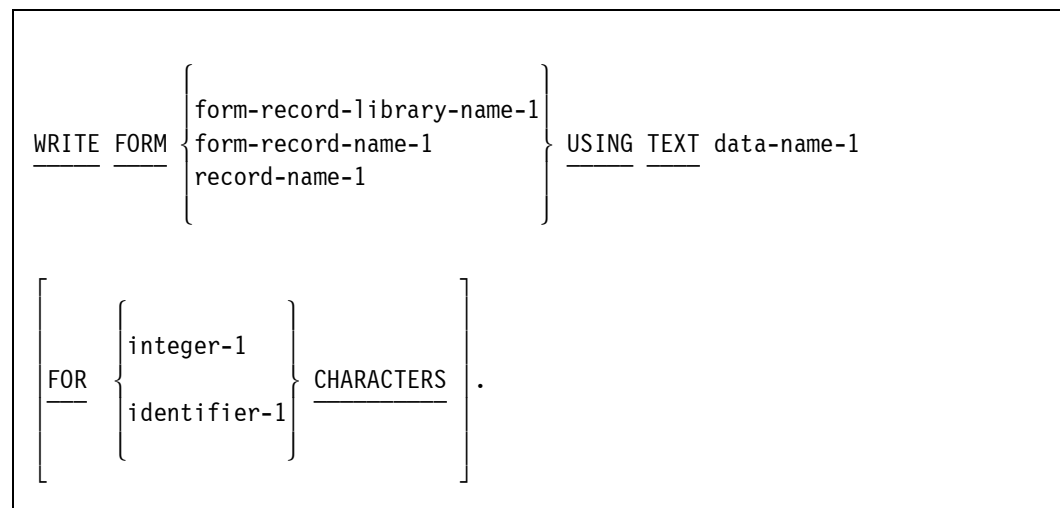
The WRITE FORM TEXT statement is an extended format of the WRITE FORM statement that causes the contents of text arrays to be written to a designated remote file by using a form record library.

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:



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:

<u>ATTRIBUTE</u>	<u>FORMRECNUM</u>	$\left\{ \begin{array}{l} \text{OF form-record-library-name-1} \\ \text{OF form-record-name-1} \end{array} \right\}$
------------------	-------------------	--

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:

<u>ATTRIBUTE</u>	<u>TRANSNUM</u>	$\left\{ \begin{array}{l} \text{OF form-record-library-name-1} \\ \text{OF form-record-name-1} \end{array} \right\}$
------------------	-----------------	--

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 the Transaction Server

SDF Plus can be used with the Transaction Server to take advantage of Transaction Server 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 Transaction Server window.

Refer to the Transaction Server *Programming Guide* for detailed information on the use of the Transaction Server direct-window interface. The following guidelines explain the steps to follow when using SDF Plus with Transaction Server.

Using Transaction Server Input/Output Headers

SDF Plus supports the use of Transaction Server 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 SDDFORMRECNUM 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 Transaction Server together, follow the usual statements for each product, with the following guidelines:

- It is recommended 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 SDDFORMRECNUM 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 SDDFORMRECNUM 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. Transaction Server 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 SDDFORMRECNUM 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 0 TO TEXTLENGTH OF COMS-OUT.  
MOVE 100 TO SDFINFO OF COMS-OUT.  
MOVE 1 TO SDDFORMRECNUM 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 the Transaction Server.
- 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 Transaction Server, refer to Section 3, "Using the Transaction Server 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 200 TO SDFINFO OF COMS-OUT.  
MOVE "This is an example of application text" TO SDF-BUFFER.  
SEND COMS-OUT FROM SDF-BUFFER.  
MOVE 0 TO SDFINFO OF COMS-OUT.  
MOVE ATTRIBUTE FORMRECNUM OF FORM1 TO  
    SDDFORMRECNUM 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 Transaction Server 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

```

Example 7–1. SDF Plus Program with a Remote File Interface

```
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
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.
```

Example 7-1. SDF Plus Program with a Remote File Interface

```

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
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

```

Example 7-1. SDF Plus Program with a Remote File Interface

```
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.
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

SDF Plus Program with a Transaction Server Interface

Example 7–2 shows the same programming logic as Example 7–1; however, the following Transaction Server interface example shows the application program interacting with users through a Transaction Server window. The SDFTRANSNUM field, which is imported from the Transaction Server input header, is interrogated to determine the form record that was read. The program indicates its response by setting the SDDFORMRECNUM field. This field was imported from the Transaction Server output header. Additionally, the program accepts two string or binary inputs from the Transaction Server 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
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

```

Example 7–2. Using SDF Plus with a Transaction Server Interface

```

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.
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*

```

Example 7-2. Using SDF Plus with a Transaction Server Interface


```

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 =
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

```

Example 7-2. Using SDF Plus with a Transaction Server Interface

```
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.
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
039000 END-OF-JOB.
```

Example 7-2. Using SDF Plus with a Transaction Server Interface

Section 8

Open Distributed Transaction Processing for ClearPath MCP

Open Distributed Transaction Processing for ClearPath MCP enables an application program to update multiple Enterprise Database Server databases in a coordinated fashion. That is, Open Distributed Transaction Processing for ClearPath Open Distributed Transaction Processing ensures that either all updates are committed or all are rolled back. Open Distributed Transaction Processing for ClearPath MCP (traditional online transaction processing) software enables an application to update multiple databases, but it is the responsibility of the application program to coordinate the updates and the recoveries of those databases.

Open Distributed Transaction Processing for ClearPath MCP also implements a client/server model. Clients invoke services but do not update the databases directly. Service providers can provide multiple services where a service is used to update an Enterprise Database Server database. The client/server model applies to the Transaction Server online programs only because the implementation of services is provided by the Transaction Server.

Open Distributed Transaction Processing for ClearPath MCP is based on the X/Open Distributed Transaction Processing (DTP) model, which is specified in standards developed by the X/Open Company, Ltd.

Example

The following example shows the logic for a client:

```
Open databases.
```

```
    Start global transaction.
```

```
    Call Service 1 to debit savings account.
```

```
    Call Service 2 to credit mutual fund account.
```

```
    If services completed successfully then
```

```
        Commit global transaction
```

```
    Else
```

```
        Rollback global transaction.
```

```
Close Databases.
```

Accessing Open Distributed Transaction Processing for ClearPath MCP

You can access Open Distributed Transaction Processing for ClearPath MCP in COBOL74 through the TX and XATMI interfaces by using an appropriate CALL statement.

Section 3, “Using the Transaction Server Program Interface,” of this manual, describes the following service functions related to the XATMI interface:

- GET_BUFFER_DESIGNATOR
- GET_DESIGNATOR_USING_DESIGNATOR
- GET_ERRORTTEXT_USING_NUMBER
- GET_INTEGER_USING_DESIGNATOR

For More Information

For complete information about creating applications that can use Open Distributed Transaction Processing for ClearPath MCP software, refer to the *Open Distributed Transaction Processing Programming Guide*.

Appendix A

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	cd-name
COMS-header-name	condition-name
data-name	family-name
file-name	formlibrary-name
form-name	group-list-name
index-name	level-number
library-name	mnemonic-name
paragraph-name	program-name
record-name	report-name
routine-name	section-name
segment-number	text-name

User-Defined Words

Index

A

- ABORT-TRANSACTION statement
 - in Enterprise Database Server, 1-5, 4-28
 - with Transaction Server, 3-2
- ACCEPT MESSAGE COUNT statement
 - in DCIENTRYPPOINT parameters, 3-14
 - in Transaction Server, 1-5, 3-18
- ACCEPT statement
 - using DCI library
 - for specifying device type, 3-14
- ACCESSDATABASE entry point
 - for TPS update library, 5-23
- Accessroutines
 - in Enterprise Database Server
 - databases, 4-18
 - in TPS
 - BEGIN-TRANSACTION statement, 5-28
 - MID-TRANSACTION statement, 5-29
- ADDS, *See* Advanced Data Dictionary System (ADDS)
- Advanced Data Dictionary System (ADDS), 2-1
 - and SDF, 6-1
 - accessing formlibraries with, 2-1
 - INVOKE ALL option in, 2-12
 - invoking formlibraries with
 - and SDF Plus, 2-13
 - sample program for, 2-21
 - and SDF Plus, 7-1
 - INVOKE ALL option in, 2-12
 - invoking formlibraries with, 2-13
 - assigning alias identifiers in, 2-4
 - data descriptions in, 2-13
 - DICTIONARY compiler control option, 2-2
 - DIRECTORY clause, 2-13
 - in DICTIONARY statement, 2-3
 - DIRECTORY option in DICTIONARY statement, 2-7
 - entities with specific status, 2-2
 - extensions of (list), 1-4
 - FROM DICTIONARY clause, 2-13
 - in SELECT statement, 2-8
 - INVOKE ALL option, 2-12
 - INVOKE clause
 - in data alias, 2-4
 - in file alias, 2-4
 - in SELECT statement, 2-8
 - invoking
 - entities with status in, 2-2
 - formlibraries with, 2-14
 - obtaining entity from dictionary, 2-13
 - options for, 2-1
 - PRODUCTION status in, 2-2
 - program samples, 2-15
 - PROGRAM-DIRECTORY option, 2-7
 - PROGRAM-NAME option, 2-7
 - PROGRAM-VERSION option, 2-7
 - RELEASE statement, 2-10
 - RETURN statement, 2-10
 - search rules in, 2-3
 - VERSION option
 - in DICTIONARY statement, 2-7
 - in SELECT statement, 2-8
- ADVANCING options
 - in Transaction Server for SEND statement, 3-27
- AFTER ADVANCING phrase
 - in Transaction Server
 - for SEND statement (format 2), 3-25
- AFTER option
 - in Transaction Server for SEND statement, 3-28
- AGENDA COBOL74 field name
 - input header in Transaction Server, 3-8
 - output header in Transaction Server, 3-10
- Agenda Designator field name
 - input header in Transaction Server, 3-8
 - output header in Transaction Server, 3-10
- ALGOL shell

- writing processing items in Transaction Server with, 3-2
- alias identifiers
 - in ADDS, 2-4
- ALL clause
 - in Enterprise Database Server for database declaration, 4-7
- AND operator
 - in Enterprise Database Server for GENERATE statement, 4-51
- application program
 - linking to Transaction Server, 3-11
- arrays
 - passing parameters to service functions, 3-35
 - rows in TPS transaction records, 5-6
- ASSIGN statement
 - in Enterprise Database Server, 1-5, 4-29
 - disadvantages of links, 4-30
 - with DM attributes, 4-23
- AT option
 - in Enterprise Database Server for selection expressions, 4-21
- ATTRIBUTE TITLE phrase
 - in Enterprise Database Server database equation, 4-18
- AUDIT clause
 - for END-TRANSACTION statement, 4-44
 - in Enterprise database Server for BEGIN-TRANSACTION statement, 4-32
- AUDITEROR 1, 4-74

B

- BEFORE ADVANCING phrase
 - in Transaction Server for SEND statement (format 2), 3-25
- BEFORE option
 - in Transaction Server for SEND statement, 3-28
- BEGINNING phrase
 - in Enterprise Database Server for SET statement, 4-69
- BEGIN-TRANSACTION ABORT statement
 - in DCIENTRYPPOINT parameters, 3-14
- BEGIN-TRANSACTION statement
 - in Enterprise Database Server, 1-5, 4-32
 - with Transaction Server, 3-2
 - in TPS, 1-7, 5-28

- BEGIN-TRANSACTION WITH TEXT statement
 - in DCIENTRYPPOINT parameters, 3-14
- BYFUNCTION mnemonic value
 - in Transaction Server
 - linking an application program example, 3-12
- BYINITIATOR mnemonic value
 - in Transaction Server
 - initializing an interface example, 3-13
 - linking an application program example, 3-12

C

- CALL statement
 - in TPS
 - invoking transaction Use procedures, 5-24
 - passing parameters with COBOL74 construct, 5-21
 - with bound procedure, 5-5
 - in Transaction Server, 3-33
 - with VALUE parameter, 1-5, 3-34
- CALL SYSTEM statement
 - in Enterprise Database Server, 4-44
- CANCEL TRANSACTION POINT statement
 - in Enterprise Database Server, 1-5, 4-35
- carriage control
 - in Transaction Server, 3-28
- CASUALOUTPUT COBOL74 field name
 - output header in Transaction Server, 3-10
- category-mnemonic
 - in Enterprise Database Server
 - with DMSTATUS word, 4-75
- CHANGE ATTRIBUTE statement
 - in Transaction Server, 3-12
- CHANGE statement
 - in Enterprise Database Server database TITLE attributes, 4-18
- CLOSE statement
 - in Enterprise Database Server, 1-5, 4-36
 - syntax used with Transaction Server, 4-37
- CLOSETRBASE entry point
 - parameters in TPS, 5-22
- COBOL74
 - constructs used in passing parameters, 5-20
 - exception handling in Enterprise Database Server, 4-74

- items mapped to in Transaction Server, 3-6
 - program interfaces, 1-3
 - user-defined words (list), A-1
- combined conditions
 - in Enterprise Database Server for IF statement, 4-54
- comments
 - in TFL descriptive information in TPS, 5-2
- communication module
 - DCI library, 3-14
- communication statements, using, 3-18
- communication structure
 - in Transaction Server, 3-2
 - constructs used in, 3-18
 - declaring the message area, 3-3
 - specifying the interface, 3-3
- compile-time functions
 - in TPS, 1-7, 5-18
 - in TPS (table), 5-19
- COMPUTE statement
 - in Enterprise Database Server, 1-5, 4-38
- COMSSUPPORT function name
 - in Transaction Server
 - linking an application program
 - example, 3-12
- CONFIRMFLAG COBOL74 field name
 - in Transaction Server, 3-10
- CONFIRMKEY COBOL74 field name
 - in Transaction Server, 3-10
- control items
 - inquiring about in TPS, 5-16
- conversation area
 - in Transaction Server header
 - declaration, 3-4
 - using SDF interface with Transaction Server, 6-11
- CONVERSATION AREA option
 - in Transaction Server header
 - declaration, 3-5
- CONVERT_TIMESTAMP service function
 - parameters in Transaction Server, 3-36
- count attribute
 - in Enterprise Database Server, 4-23
- Count field
 - in Enterprise Database Server, 4-23
- CREATE statement
 - in Enterprise Database Server, 1-6, 4-39
 - in TPS
 - initializing a transaction record, 1-7
 - initializing transaction record
 - formats, 5-8
- CREATETRUSER entry point

- parameters in TPS, 5-22
- CURRENT option
 - in Enterprise Database Server
 - for REMOVE statement, 4-64
- cursor positioning
 - in SDF, 6-13

D

- DASDL, *See* Data and Structure Definition Language (DASDL)
- data alias
 - using the INVOKE clause in, 2-4
- Data and Structure Definition Language (DASDL)
 - in Enterprise Database Server
 - data sets, 4-12
 - link items, 4-22
 - naming, 4-2
 - in TPS, sample program for, 5-31
- data communications interface (DCI) library
 - example of library object, 3-17
 - in Enterprise Database Server
 - for BEGIN-TRANSACTION
 - statement, 4-32
 - for END-TRANSACTION
 - statement, 4-45
 - in Transaction Server
 - linking program, 3-12
 - parameters for, 3-14
- DATA DIVISION of a program
 - invoking files and records from, 2-12
- data items
 - qualifying in Enterprise Database Server, 4-4
 - qualifying in TPS, 5-15
- data management (DM) attributes, 4-23
 - count attribute in, 4-23
 - in Enterprise Database Server, 1-6, 4-23
 - population attribute in, 4-25
 - record type attribute in, 4-24
- data management statements
 - in Enterprise Database Server, 4-28
- data only flag
 - in SDF, 6-13
- data set
 - in Enterprise Database Server
 - invoking, 4-10
 - qualifying names, 4-2
- data set reference entry

- in Enterprise Database Server, 1-6, 4-7, 4-10
- data types
 - in Transaction Server, 3-6
- database
 - in Enterprise Database Server and Accessroutines, 4-18
 - declaring, 1-6, 4-7
 - equation operation, 4-18
 - identifying database components, 4-2
 - referencing items from, 4-4
 - titles of operation, 4-18
 - using items of, 4-2
 - using the name of logical database in SECURE statement, 4-67
 - in TPS
 - opening for subsequent access, 5-26
- database declaration
 - in Enterprise Database Server, 1-6, 4-7
- database equations
 - in Enterprise Database Server, 4-18
- database items
 - in Enterprise Database Server, 4-2
- DATA-BASE SECTION
 - in Enterprise Database Server, 4-4
- data-description entry
 - in ADDS, 1-4, 2-13
 - for DIRECTORY clause, 2-3
 - for VERSION clause, 2-3
 - in SDF, 1-8, 6-3
 - in SDF Plus, 1-9, 7-6
- DATADICTINFO compiler option
 - in Enterprise Database Server, 4-27
- DB, *See* database
- DCI library, *See* data communications interface (DCI) library
- DCENTRYPOINT library object
 - advancing parameters in, 3-16
 - device type parameter in, 3-16
 - EGL option parameter in, 3-14
 - EMI option parameter in, 3-14
 - ESI option parameter in, 3-14
 - for the DCI library, 3-14
 - value specifying functions for, 3-14
- DCILIBRARY option
 - naming convention with VALUE parameter, 3-33
- deadlock, 4-57, 4-58, 4-59, 4-65
- declaring
 - a message area in Transaction Server, 3-3
 - an Enterprise Database Server database, 4-7
 - an interface in Transaction Server, 3-3
 - Transaction Server headers, 3-4
- DELETE statement
 - in Enterprise Database Server, 1-6, 4-41
- Delivery Confirmation Flag field
 - output header in Transaction Server, 3-10
- delivery confirmation in Transaction Server, 3-10
 - requesting, 3-11
- Delivery Confirmation Key field
 - output header in Transaction Server, 3-10
- designators
 - passing parameters to service functions, 3-35
 - used in Transaction Server, 3-7, 3-31
- DESTCOUNT COBOL74 field name
 - output header in Transaction Server, 3-10
- Destination Designator field name
 - output header in Transaction Server, 3-10
- DESTINATIONDESG COBOL74 field name
 - output header in Transaction Server, 3-10
- device type
 - parameter in DCENTRYPOINT library object, 3-16
- dictionary
 - identifying in ADDS, 2-6
 - identifying in SDF, 6-2
 - identifying in SDF Plus, 7-5
- DICTIONARY compiler control option
 - in ADDS, 1-4, 2-2
- DICTIONARY statement
 - in ADDS, 1-4, 2-6
 - in SDF, 1-8, 6-2
 - in SDF Plus, 1-9, 7-5
- DIRECTORY clause
 - in ADDS, 1-4
 - for data description, 2-3, 2-13
 - for DICTIONARY statement, 2-3
 - for file description, 2-3
- DIRECTORY option
 - in ADDS
 - for DICTIONARY statement, 2-7
 - for file selection in SELECT statement, 2-8
 - for SELECT statement, 2-8
 - in SDF Plus, 7-7
- DISABLE statement
 - in Transaction Server, 1-5, 3-19
 - using DCI library for specifying device type, 3-14
- DMCATEGORY attribute
 - in Enterprise Database Server for DMSTATUS format, 4-75
- DMERROR Use procedure

- in Enterprise Database Server, 1-6, 4-77
 - for DMSTATUS format, 4-77
- DMERRORTYPE attribute
 - in Enterprise Database Server for DMSTATUS format, 4-75
- DMRESULT attribute
 - in Enterprise Database Server for DMSTATUS format, 4-75
- DMSTATUS database status word
 - in Enterprise Database Server, 1-6, 4-74
- DMSTRUCTURE attribute
 - in Enterprise Database Server for DMSTATUS format, 4-75
 - for processing exceptions, 4-74
- DMSTRUCTURE function
 - in Enterprise Database Server, 1-6
 - number function, 4-76
- DMTERMINATE statement
 - in Enterprise Database Server, 1-6, 4-44
 - with Transaction Server, 3-2
- DMVERB, 4-55

E

- EBCDIC array
 - in DCIENTRYPPOINT library object
 - parameter length, 3-14
- EGI, *See* end-of-group indicator (EGI) option
- ELSE statement
 - in Enterprise Database Server
 - with IF statement, 4-54
- EMI, *See* end-of-message indicator (EMI) option
- ENABLE statement
 - in Transaction Server, 1-5, 3-20
 - using the MCS, 3-21
 - using DCI library
 - for specifying device type, 3-14
- ENDING phrase
 - in Enterprise Database Server
 - for SET statement, 4-69
- end-of-group indicator (EGI) option
 - in Transaction Server
 - for SEND statement, 3-25
 - parameter in DCIENTRYPPOINT library object, 3-14
- end-of-message indicator (EMI) option
 - in Transaction Server
 - for SEND statement (format 2), 3-25
 - parameter in DCIENTRYPPOINT library object, 3-14
- end-of-segment indicator (ESI) option
 - in Transaction Server
 - for SEND statement (format 2), 3-25
 - parameter in DCIENTRYPPOINT library object, 3-14
- END-TRANSACTION ABORT statement
 - in DCIENTRYPPOINT parameters, 3-14
- END-TRANSACTION statement
 - in Enterprise Database Server, 1-6, 4-44
 - with Transaction Server, 3-2
 - in TPS, 1-7, 5-30
- END-TRANSACTION WITH NO TEXT statement
 - in DCIENTRYPPOINT parameters, 3-14
- END-TRANSACTION WITH TEXT statement
 - in DCIENTRYPPOINT parameters, 3-14
- Enterprise Database Server, 4-1, *See* Enterprise Database Server, *See* Enterprise Database Server
 - accessing an established database, 4-60
 - Accessroutines, 5-28
 - used with TPS, 5-28, 5-29
 - ALL clause
 - in database declaration, 4-7
 - AND operator in GENERATE statement, 4-51
 - and Transaction Server
 - ABORT-TRANSACTION statement, 4-28
 - BEGIN-TRANSACTION statement, 4-32
 - DMTERMINATE statement, 4-44
 - END-TRANSACTION statement, 4-45
 - statements used with, 3-2
 - ASSIGN data management statement, 4-29
 - effect on Count field, 4-23
 - assigning a Boolean value, 4-38
 - AT and WHERE clauses in, 4-21
 - ATTRIBUTE TITLE phrase
 - in database equation, 4-18
 - AUDIT clause
 - in BEGIN-TRANSACTION statement, 4-32
 - in END-TRANSACTION statement, 4-44
 - BEGINNING phrase in SET statement, 4-69
 - BEGIN-TRANSACTION statement, 4-32
 - CANCEL TRANSACTION POINT statement, 4-35
 - category-mnemonic value specification, 4-75
 - changing current record path or value, 4-68

- CLOSE data management statement, 4-36
- closing a database, 4-36
- COMPUTE data management
 - statement, 4-38
- Count field in, 4-23
- CREATE data management
 - statement, 4-39
- creating a subset in one operation, 4-51
- CURRENT phrase in REMOVE
 - statement, 4-64
- DASDL link items, 4-22
- data management (DM) attributes in, 4-23
- data management statements for, 4-28
- data set referencing, 4-10
- data set structure number
 - determination, 4-76
- database
 - data and the object code, 4-27, 4-28
 - equation operation in, 4-18
 - sections and the compiler, 4-7
 - status word for, 4-74
- DATADICTINFO compiler option, 4-27
- DELETE data management
 - statement, 4-41
- deleting a record, 4-41
- DMERROR attribute for DMSTATUS
 - format, 4-75
- DMERROR Use procedure in, 4-77
- DMERRORTYPE attribute for DMSTATUS
 - format, 4-75
- DMRESULT attribute for DMSTATUS
 - format, 4-75
- DMSTATUS word, 4-75
- DMSTRUCTURE attribute for DMSTATUS
 - format, 4-75
- DMSTRUCTURE function, 4-76
- DMTERMINATE statement, 4-44
- ELSE statement with IF statement, 4-54
- ENDING phrase in SET statement, 4-69
- END-TRANSACTION statement, 4-44
- establishing record relationships in, 4-29
- exception handling
 - examples of, 4-37
 - exception categories, 4-74
 - for ABORT-TRANSACTION
 - statement, 4-28
 - for ASSIGN statement, 4-30
 - for CANCEL TRANSACTION POINT
 - statement, 4-35
 - for CLOSE statement, 4-37
 - for CREATE statement, 4-39
 - for DEADLOCK exception, 4-58
 - for DELETE statement, 4-42
 - for END-TRANSACTION
 - statement, 4-45
 - for FIND statement, 4-49
 - for FREE statement, 4-50
 - for GENERATE statement, 4-52
 - for INSERT statement, 4-56
 - for LOCK statement, 4-59
 - for MODIFY statement, 4-59
 - for OPEN statement, 4-61
 - for RECREATE statement, 4-63
 - for REMOVE statement, 4-65
 - for SAVE TRANSACTION POINT
 - statement, 4-66
 - for SET statement, 4-70
 - for STORE statement, 4-72
 - with DMERROR procedure, 4-77
 - with DMSTATUS word, 4-74
- extensions of (list), 1-5
- FALSE clause for COMPUTE
 - statement, 4-38
- FIND data management statement, 4-47
- FIND KEY OF clause in FIND
 - statement, 4-48
- FIRST option
 - for selection expression, 4-20
- FREE data management statement, 4-49
- GENERATE data management
 - statement, 4-51
- GLOBAL clause
 - in database declaration, 4-7
- IF data management statement, 4-53
- INDEPENDENTTRANS option
 - and FREE statements, 4-49
 - in record locking, 4-58
- initializing a user work area, 4-39, 4-62
- INQUIRY phrase in OPEN statement, 4-60
- INSERT data management
 - statement, 4-56
- INVOKE clause
 - in data set references, 4-10
- INVOKE option, 4-10
- invoking data sets, 4-7
- key condition option in, 4-21
- LAST option
 - for selection expression, 4-20
- LOCK/MODIFY data management
 - statements, 4-58
- minus (-) operator in GENERATE
 - statement, 4-51
- modifying an audited area, 4-33
- MOVE CORRESPONDING statement, 4-5
- name qualification in, 4-2

- naming database components with
 - DASDL, 4-2
- naming database items, 4-2
- NEXT option in, 4-21
- NEXT SENTENCE option
 - in ABORT-TRANSACTION statement, 4-28
 - in ASSIGN statement, 4-29
 - in CANCEL TRANSACTION POINT statement, 4-35
 - in CLOSE statement, 4-36
 - in CREATE statement, 4-39
 - in DELETE statement, 4-42
 - in END-TRANSACTION statement, 4-44
 - in FIND statement, 4-48
 - in FREE statement, 4-50
 - in GENERATE statement, 4-51
 - in INSERT statement, 4-56
 - in LOCK/MODIFY statement, 4-58
 - in ON EXCEPTION option, 4-78
 - in OPEN statement, 4-60
 - in RECREATE statement, 4-62
 - in REMOVE statement, 4-64
 - in SAVE TRANSACTION POINT statement, 4-66
 - in SECURE statement, 4-67
 - in SET statement, 4-69
 - in STORE statement, 4-72
- NO-AUDIT clause
 - in BEGIN-TRANSACTION statement, 4-32
 - in END-TRANSACTION statement, 4-44
- NOT option in IF statement, 4-54
- NOTFOUND exception in, 4-21
- NULL clause
 - in ASSIGN statement, 4-29
- NULL phrase
 - in GENERATE statement, 4-51
 - in IF statement, 4-54
 - in SET statement, 4-69
- ON EXCEPTION option, 4-78
 - in ASSIGN statement, 4-29
 - in CLOSE statement, 4-36
- OPEN data management statement, 4-60
- operating system, role in constructing a database, 4-9
- OR operator in GENERATE statement, 4-51
- partitioned structure numbers in, 4-76
- placing a program in transaction state, 4-32
- plus (+) operator in GENERATE statement, 4-51
- population DM attribute in, 4-25
- PRIOR option
 - for selection expression, 4-20
- processing exceptions, 4-74
- program interface for, 4-1
- program removal from transaction state, 4-44
- qualifying set and data set names, 4-2
- record inserting into a manual subset, 4-56
- record locking
 - against modification, 4-58
 - and removing, 4-64
- record type attribute in, 4-24
- Record Type field in, 4-23
- RECREATE data management statement, 4-62
- remaps, declaring in DASDL, 4-12
- REMOVE data management statement, 4-64
- sample program with Transaction Server, 3-52
- SAVE TRANSACTION POINT data management statement, 4-66
- SECURE data management statement, 4-67
- selection expressions in, 4-20
- set referencing in, 4-11
- SET statement, 4-68
- specifying the database access mode, 4-60
- stopping record updates by other programs, 4-67
- STORE data management statement, 4-71
- storing a record into a data set, 4-71
- structure name of population in, 4-23
- STRUCTURE phrase
 - in FREE statement, 4-50
- synchronizing transaction and recovery with Transaction Server, 4-28
- syncpoint
 - using in END-TRANSACTION statement, 4-44
- terminating the program, 4-44
- testing for a NULL value, 4-53
- TITLE attribute
 - in Enterprise Database Server, 4-18
- TPS interface sample program for, 5-31
- transaction point record for audit, 4-66
- transaction updates, 4-28
 - discarding, 4-35
- transferring a record to user work area, 4-47

- TRUE clause for COMPUTE
 - statement, 4-38
- unlocking the current record, 4-49
- unlocking the current structure, 4-49
- UPDATE phrase in OPEN statement, 4-60
- uses (list), 4-1
- USING clause in data set references, 4-10
- using DASDL to invoke data sets,
 - examples of, 4-12
- using database items, 4-2
- USING option, 4-11
- using variable-format records, 4-5
- VALUE OF TITLE clause
 - in database declaration, 4-7
- VIA option
 - for selection expression, 4-20
- Enterprise Database Server TPS, *See*
 - transaction processing system (TPS)
- entities
 - in ADDS
 - identifying, 2-3
 - obtaining from dictionary, 2-13
 - using the DIRECTORY clause, 2-3
 - using the FROM DICTIONARY clause, 2-13
 - using the VERSION clause, 2-3
 - with specific status, 2-2
- entry points
 - in TPS
 - transaction library, 5-20
 - use of CALL statement, 5-22
 - in TPS (table), 5-22
 - in Transaction Server, 3-33
- equation operations
 - in Enterprise Database Server, 4-18
- ESI, *See* end-of-segment indicator (ESI)
 - option
- example of DCI library object code, 3-17
- examples of Enterprise Database Server
 - application programs
 - ASSIGN statement, 4-31
 - BEGIN-TRANSACTION statement, 4-34
 - CLOSE statement, 4-37
 - CREATE statement, 4-40
 - data set referencing to invoke disjoint data sets, 4-14
 - database equation operations, 4-19
 - DELETE statement, 4-42
 - designating sets as visible or invisible, 4-15
 - DMERROR Use procedure
 - declarations for, 4-77
 - DMTERMINATE statement, 4-44
 - END-TRANSACTION statement, 4-46
 - exception handling, 4-79
 - FREE statement, 4-50
 - GENERATE statement, 4-52
 - group move of database items, 4-5
 - host program declarations for using
 - GLOBAL option, 4-17
 - INSERT statement, 4-57
 - LOCK statement with ON EXCEPTION
 - option, 4-60
 - MERROR Use procedure
 - and exception handling, 4-78
 - MODIFY statement with ON EXCEPTION
 - option, 4-60
 - MOVE CORRESPONDING statement with
 - database items, 4-6
 - NULL phrase with IF statement, 4-55
 - OPEN statement with INQUIRY
 - phrase, 4-62
 - population attribute, 4-26
 - procedure to reference a database with
 - GLOBAL option, 4-16
 - record type attribute, 4-25
 - RECREATE statement, 4-63
 - REMOVE statement, 4-66
 - STORE statement, 4-73
 - valid and invalid name qualification, 4-3
- examples of Transaction Server application
 - programs
 - application program linking (example 1), 3-12
- CALL statement
 - with service functions, 3-33
 - with VALUE parameter, 3-35
- input and output header declarations, 3-5
- interface initialization, 3-13
- message area declaration, 3-3
- message placement in WORKING-STORAGE SECTION, 3-24
- SEND statements with ESI and EGI
 - options, 3-29
- service functions
 - CONVERT_TIMESTAMP, 3-36
 - GET_BUFFER_DESIGNATOR, 3-37
 - GET_DESIGNATOR_ARRAY_USING_DESIGNATOR, 3-38
 - GET_DESIGNATOR_USING_DESIGNATOR, 3-39
 - GET_DESIGNATOR_USING_NAME, 3-41
 - GET_ERRORTEXT_USING_NUMBER, 3-40
 - GET_INTEGER_ARRAY_USING_DESIGNATOR, 3-42

- GET_INTEGER_USING_DESIGNATOR, 3-43
 - GET_NAME_USING_DESIGNATOR, 3-45
 - GET_REAL_ARRAY, 3-46
 - GET_STRING_USING_DESIGNATOR, 3-47
 - TEST_DESIGNATORS, 3-50
 - with Enterprise Database Server database, 3-52
 - exception categories
 - in Enterprise Database Server, 4-74
 - exceptions
 - in Enterprise Database Server
 - categories, 4-74
 - DMERROR Use procedure, 4-77
 - ON EXCEPTION option, 4-77
 - processing, 4-74
 - using COBOL74 exception handling, 4-74
 - using DMSTATUS word, 4-74
 - using DMSTRUCTURE function, 4-74
 - in TPS, 5-26
 - expressions in Enterprise Database Server
 - for CREATE statement, 4-39
 - extensions
 - in ADDS (list), 1-4
 - in Enterprise Database Server (list), 1-5
 - in Enterprise Database Server TPS (list), 1-7
 - in SDF Plus (list), 1-9
 - in Transaction Server (list), 1-5
 - of each product, 1-4
- F**
- FALSE clause
 - in Enterprise Database Server for COMPUTE statement, 4-38
 - FD level indicator
 - in ADDS, 1-4
 - field suppress flag
 - in SDF, 6-13
 - file alias
 - using the INVOKE clause, 2-4
 - file description
 - in ADDS
 - FD entry, 1-4, 2-10
 - for DIRECTORY clause, 2-3
 - for VERSION clause, 2-3
 - SD entry, 2-10
 - FILE SECTION of a program
 - invoking record descriptions, 2-14
 - file status update
 - when reading forms in SDF, 6-7
 - FIND KEY OF clause
 - in Enterprise Database Server for FIND statement, 4-48
 - FIND statement
 - in Enterprise Database Server, 1-6, 4-47
 - FIRST option
 - in Enterprise Database Server for selection expression, 4-20
 - flag groups flag
 - in SDF, 6-13
 - flags
 - resetting in SDF, 6-15
 - FOR option
 - in SDF Plus for WRITE FORM TEXT statement, 7-22
 - form record library
 - in SDF Plus, 7-2
 - data-description entry, 7-8
 - form records, 7-2
 - record numbers, 7-3
 - form record number attribute
 - in SDF Plus, 1-9, 7-23
 - FORM-KEY function
 - in SDF, 1-8
 - with Transaction Server, 6-11
 - formlibrary
 - as an ADDS entity, 2-1
 - in ADDS data-description entry, 2-14
 - in SDF data-description entry, 6-5
 - formlibrary name
 - in SDF data-description entry, 6-4
 - forms
 - default
 - in SDF, 6-10
 - in Transaction Server, 6-18
 - FREE statement
 - in Enterprise Database Server, 1-6, 4-49
 - freeing data set records
 - constructs (list), 4-49
 - FROM DEFAULT FORM option
 - in SDF for WRITE statement, 6-10
 - FROM DEFAULT FORM phrase
 - in SDF for READ FORM statement, 6-7
 - FROM DICTIONARY clause
 - in ADDS, 1-4
 - for data-description entry, 2-13
 - for INVOKE ALL option, 2-13
 - for SELECT statement, 2-8

- in SDF, 1-8, 6-3
 - in SDF Plus, 1-9, 7-6
- FROM option
 - in SDF Plus
 - for READ FORM statement, 7-12
 - in SDF Plus
 - for WRITE FORM statement (format 2), 7-18
- FROM phrase
 - in Transaction Server
 - for SEND statement (format 2), 3-25
 - in Transaction Server for SEND statement (format 1), 3-24
- Function Index field name
 - input header in Transaction Server, 3-8
- Function Status field name
 - input header in Transaction Server, 3-8
- FUNCTIONINDEX COBOL74 field name
 - input header in Transaction Server, 3-8
- FUNCTIONNAME attribute
 - in Transaction Server
 - initializing an interface example, 3-13
 - linking an application program example, 3-12
- functions
 - of a DCI library, 3-14
- FUNCTIONSTATUS COBOL74 field name
 - input header in Transaction Server, 3-8

G

- GENERATE statement
 - in Enterprise Database Server, 1-6, 4-51
- GET_BUFFER_DESIGNATOR service function
 - parameters in Transaction Server, 3-37
- GET_DESIGNATOR_ARRAY_USING_DESIGNATOR service function
 - parameters in Transaction Server, 3-38
- GET_DESIGNATOR_USING_DESIGNATOR service function
 - parameters in Transaction Server, 3-39
- GET_DESIGNATOR_USING_NAME service function
 - parameters in Transaction Server, 3-41
- GET_ERRORTEXT_USING_NUMBER service function
 - parameters in Transaction Server, 3-40
- GET_INPUT_QUEUE_STATUS service function
 - parameters in Transaction Server, 3-44

- GET_INTEGER_ARRAY_USING_DESIGNATOR service function
 - parameters in Transaction Server, 3-42
- GET_INTEGER_USING_DESIGNATOR service function
 - parameters in Transaction Server, 3-43
- GET_NAME_USING_DESIGNATOR service function
 - example with CALL statement in Transaction Server, 3-34
 - parameters in Transaction Server, 3-45
- GET_REAL_ARRAY service function
 - parameters in Transaction Server, 3-46
- GET_STRING_USING_DESIGNATOR service function
 - parameters in Transaction Server, 3-47
- GET_WINDOW_NAME_USING_DESIGNATOR service function
 - parameters in Transaction Server, 3-48
- GIVING clause
 - passing parameters with COBOL74 constructs in TPS, 5-21
- GLOBAL clause
 - in Enterprise Database Server
 - for database declaration, 4-7
 - in SDF Plus, 1-9
 - in TPS
 - transaction Use procedures, 5-24
- GLOBAL option
 - in SDF Plus, 7-7
 - in TPS
 - transaction base declaration in, 5-5
 - using to declare record variables, 5-7

H

- HANDLESTATISTICS entry point
 - in TPS parameters, 5-22
- headers
 - declaring in Transaction Server, 3-4
 - fields of input header, 3-7
 - fields of output header, 3-9
 - using in Transaction Server, 1-5
- highlight flag
 - in SDF, 6-14
- hyphenation
 - in TPS
 - using hyphenated identifier, 5-2
 - in Transaction Server
 - with service function mnemonic names, 3-34

- with service function names, 3-30
- I**
- identifier
 - in Enterprise Database Server database components, 4-2
 - in TPS, 5-2
 - identifying records in a data set, 4-19
 - IF statement
 - in Enterprise Database Server, 1-6, 4-53
 - INDEPENDENTTRANS option
 - in Enterprise Database Server
 - and FREE statement, 4-49
 - for CREATE statement, 4-39
 - for LOCK/MODIFY statement, 4-58
 - initializing a program in Transaction Server, 3-13
 - INPUT HEADER phrase
 - in Transaction Server header declaration, 3-4
 - input headers
 - in Transaction Server, 3-4, 3-7
 - fields of (table), 3-7
 - INPUT TERMINAL phrase
 - in Transaction Server
 - for DISABLE statement, 3-19
 - for ENABLE statement, 3-20, 3-21
 - input/output flag
 - in SDF, 6-14
 - INQUIRY option
 - for OPEN statement
 - in Enterprise Database Server TPS, 5-27
 - INQUIRY phrase
 - for OPEN statement
 - in Enterprise Database Server, 4-60
 - INSERT statement
 - in Enterprise Database Server, 1-6, 4-56
 - integers
 - using in Transaction Server, 3-7
 - INTO option
 - in SDF for READ FORM statement, 6-7
 - in SDF Plus for READ FORM statement, 7-12
 - INVOKE ALL clause
 - in ADDS, 1-4
 - INVOKE ALL option
 - invoking files and records in ADDS, 2-12
 - INVOKE clause
 - in ADDS, 1-4
 - assigning a data alias, 2-4
 - assigning a file alias, 2-4
 - for SELECT statement, 2-8
 - restricting in SDF formlibrary, 6-5
 - in Enterprise Database Server
 - using in data set references, 4-10
 - INVOKE option
 - in Enterprise Database Server
 - for data set reference entry, 4-10
 - for database declaration, 4-7
 - invoking structures
 - in a database declaration
 - explicitly, 4-11
 - implicitly, 4-11
 - invoking data sets, 4-10
 - more than once, 4-7
 - selectively, 4-7
 - IS option
 - in Transaction Server, 3-5
- K**
- key condition
 - in Enterprise Database Server
 - for selection expressions, 4-21
 - key condition option
 - in Enterprise Database Server
 - for selection expressions, 4-21
 - KEY values
 - in Transaction Server
 - for DISABLE statement, 3-19
 - for ENABLE statement, 3-20, 3-21
- L**
- LAST option
 - in Enterprise Database Server
 - for selection expression, 4-20
 - level number
 - in SDF data-description entry, 6-3
 - LIBACCESS attribute
 - in Transaction Server
 - initializing an interface example, 3-13
 - linking an application program example, 3-12
 - library attributes
 - example used in Transaction Server program link, 3-12
 - linking messages

- from a program to Transaction Server, 3-11
- links
 - disadvantages of, 4-30
- LIST option
 - in TPS compiler control option, 5-2
- LOCK/MODIFY statement
 - in Enterprise Database Server, 1-6, 4-58
- locking records
 - in Enterprise Database Server
 - with a LOCK/MODIFY statement, 4-58
 - with a SECURE statement, 4-67
- LOGOFFTRUSER parameters
 - in TPS, 5-22
- LOGONTRUSER parameters
 - in TPS, 5-22
- LOW-VALUES figurative constant
 - using with SDF programmatic flags, 6-15

M

- mapping Transaction Server data types into COBOL74, 3-6
- MCS, *See* message control system (MCS)
- message area
 - declaring in Transaction Server
 - example, 3-3
- message control system (MCS)
 - linking application programs to Transaction Server, 3-12
 - using
 - in ENABLE statement, 3-21
- Message Count field
 - in Transaction Server
 - for ACCEPT MESSAGE COUNT statement, 3-18
- Message Count field name
 - input header in Transaction Server, 3-8
- message keys
 - in SDF for READ FORM statement, 6-7
- message length
 - using the DCI library, 3-14
- MESSAGE phrase
 - in Transaction Server for RECEIVE statement, 3-22
- MESSAGECOUNT COBOL74 field name
 - input header in Transaction Server, 3-8
- messages
 - in SDF Plus
 - sending, 7-29
 - in Transaction Server

- receiving, 3-11
- sending, 3-11
- MID-TRANSACTION statement
 - in TPS, 1-7, 5-29
- minus (-) operator
 - in Enterprise Database Server
 - for GENERATE statement, 4-51
- mnemonic-name option
 - in Transaction Server
 - for SEND statement, 3-28
- mnemonics
 - for passing parameters to service functions, 3-35
 - passing to get a numeric result in Transaction Server, 3-34
 - service function (table), 3-31
- MODIFY statement
 - in Enterprise Database Server, 4-58
- MOVE CORRESPONDING statement
 - accessing record items in TPS, 5-12
 - with database items in Enterprise Database Server, 4-5
- MOVE statement
 - in Enterprise Database Server
 - for database TITLE attributes, 4-18
 - in SDF
 - for FORM-KEY function, 6-10
 - for READ FORM statement, 6-7
 - for WRITE FORM statement, 6-10
 - in SDF Plus
 - for READ FORM statement, 7-12
 - in SDF Plus
 - for WRITE FORM statement (format 2), 7-18
 - in TPS
 - for assigning record variables, 5-11

N

- naming database items, 4-2
- network support processor (NSP)
 - delivery confirmation in Transaction Server, 3-11
- Next Input Agenda Designator field name
 - output header in Transaction Server, 3-10
- NEXT option
 - in Enterprise Database Server
 - for selection expressions, 4-21
- NEXT SENTENCE option
 - in BEGIN-TRANSACTION statement, 4-32
 - in Enterprise Database Server

- for ABORT-TRANSACTION statement, 4-28
- for ASSIGN statement, 4-29
- for BEGIN-TRANSACTION statement, 4-32
- for CANCEL TRANSACTION POINT statement, 4-35
- for CLOSE statement, 4-36
- for CREATE statement, 4-39
- for END-TRANSACTION statement, 4-44
- for FIND statement, 4-48
- for FREE statement, 4-50
- for GENERATE statement, 4-51
- for INSERT statement, 4-56
- for LOCK/MODIFY statement, 4-58
- for NEXT TRANSACTION POINT statement, 4-66
- for ON EXCEPTION option, 4-78
- for OPEN statement, 4-60
- for RECREATE statement, 4-62
- for REMOVE statement, 4-64
- for SECURE statement, 4-67
- for SET statement, 4-69
- for STORE statement, 4-72
- in Transaction Server
 - for RECEIVE statement, 3-23
- NEXTINPUTAGENDA COBOL74 field name
 - output header in Transaction Server, 3-10
- NO DATA option
 - in Transaction Server for RECEIVE statement, 3-22
- no input flag
 - in SDF, 6-14
- NO-AUDIT clause
 - for END-TRANSACTION statement, 4-44
 - in Enterprise Database Server
 - for BEGIN-TRANSACTION statement, 4-32
- NOT option
 - in Enterprise Database Server
 - for IF statement, 4-54
- NOTFOUND exception
 - and Enterprise Database Server selection expressions, 4-21
- NSP, *See* network support processor (NSP)
- NULL clause
 - in Enterprise Database Server
 - for ASSIGN statement, 4-29
- NULL option
 - in Enterprise Database Server
 - for CREATE statement, 4-39
- NULL phrase

- in Enterprise Database Server
 - for GENERATE statement, 4-51
 - for IF statement, 4-54
 - for SET statement, 4-69

O

- OCCURS clause
 - in TPS
 - declaring record variables, 5-7
 - subscript required, 5-13
- ON ERROR condition
 - in SDF
 - for READ FORM statement, 6-8
 - for WRITE FORM statement, 6-10
 - in SDF Plus
 - for READ FORM statement, 7-12
 - in SDF Plus
 - for WRITE FORM statement (format 1), 7-16
 - in SDF Plus
 - for WRITE FORM statement (format 2), 7-18
- ON EXCEPTION clause
 - in Enterprise Database Server
 - for BEGIN-TRANSACTION statement, 4-33
- ON EXCEPTION option
 - in Enterprise Database Server, 1-7, 4-78
 - for ASSIGN statement, 4-29, 4-30
 - for CANCEL TRANSACTION POINT statement, 4-35
 - for CLOSE statement, 4-36
 - for CREATE statement, 4-39
 - for DELETE statement, 4-42
 - for END-TRANSACTION statement, 4-45
 - for FIND statement, 4-49
 - for FREE statement, 4-50
 - for GENERATE statement, 4-52
 - for INSERT statement, 4-56
 - for LOCK statement, 4-59
 - for MODIFY statement, 4-59
 - for OPEN statement, 4-61
 - for RECREATE statement, 4-63
 - for REMOVE statement, 4-65
 - for SET statement, 4-70
 - for STORE statement, 4-72
 - in TPS
 - for BEGIN-TRANSACTION statement, 5-28

- for END-TRANSACTION statement, 5-30
- for MID-TRANSACTION statement, 5-29
- for OPEN statement, 5-27
- online transaction processing, 8-1
- Open Distributed Transaction Processing, 8-1
- Open Distributed Transaction Processing (DTP), 4-74
- OPEN statement
 - in Enterprise Database Server, 1-7, 4-60
 - in TPS
 - for TRUPDATE option, 5-26
- OPENTRBASE parameters
 - in TPS, 5-22
- operating system
 - using to construct a database, 4-9
- OR operator
 - in Enterprise Database Server
 - for GENERATE statement, 4-51
- OUTPUT HEADER phrase
 - in Transaction Server header declaration, 3-4
- output headers
 - in Transaction Server, 3-4, 3-9
 - fields of (table), 3-10
- output message in Transaction Server
 - delivery confirmation in, 3-11
 - output header field used in, 3-9
- OUTPUT TERMINAL phrase
 - in Transaction Server
 - for DISABLE statement, 3-19
 - for ENABLE statement, 3-20, 3-21

P

- page flag
 - in SDF, 6-14
- page specification
 - using in the Transaction Server SEND statement, example of, 3-28
- parameters
 - in DCIENTRYPPOINT library object, 3-14
 - in TPS
 - passing transaction record variables in, 5-10
 - passing with COBOL74 constructs, 5-20
 - received by update library, 5-25
- partitioned structure

- in Enterprise Database Server, 4-76
- password
 - in DCIENTRYPPOINT library object
 - parameter length, 3-14
- peripherals
 - using symbolic sources and destinations with, 3-14
- placing program in transaction state, 5-28
- plus (+) operator
 - in Enterprise Database Server
 - for GENERATE statement, 4-51
- population attribute
 - in Enterprise Database Server, 4-25
 - of a Enterprise Database Server structure, 4-23
- PRIOR option
 - in Enterprise Database Server
 - for selection expression, 4-20
- PROCESSTRANSACTION parameters
 - in TPS, 5-22
- PROCESSTRFROMTANK parameters
 - in TPS, 5-22
- PROCESSTRNORESTART parameters
 - in TPS, 5-22
- PRODUCTION status
 - in ADDS, 2-2
- Program Designator field name
 - input header in, 3-8
- program interfaces
 - as COBOL74 extensions, 1-3
 - extensions, by product (list), 1-4
 - general concepts of, 1-3
 - with combined products, 1-3
 - with Transaction Server, 3-1
- program tracking
 - in ADDS, 2-6
- PROGRAMDESG COBOL74 field name
 - input header in Transaction Server, 3-8
- PROGRAM-DIRECTORY option
 - for program tracking in ADDS, 2-7
- programmatic controls
 - in SDF
 - control information (table), 6-13
 - flag settings, 6-12, 6-13
 - flag suffixes (table), 6-13
 - generating flag groups, 6-15
 - naming conventions for, 6-12
 - resetting flags, 6-15
 - sample program using, 6-23
- PROGRAM-NAME option
 - for program tracking in ADDS, 2-7
- PROGRAM-VERSION option
 - for program tracking in ADDS, 2-7

PURGETRUSER parameters
in TPS, 5-22

Q

qualification
in Enterprise Database Server
of set and data set names, 4-2
in TPS
data items, 5-15
requirements for (list), 5-15

R

RDS, *See* restart data set (RDS)

READ FORM statement
in SDF, 1-8, 6-6
for default forms, 6-7
for FROM DEFAULT FORM option, 6-7
for INTO option, 6-7
for self-identifying forms, 6-7
for specific forms, 6-7
for USING clause, 6-7
sample program for, 6-19
in SDF Plus, 1-9, 7-10
for default form records, 7-12
for FROM DEFAULT FORM option, 7-12
for INTO option, 7-12
for self-identifying reads, 7-11
for specific form records, 7-12
for USING clause, 7-11, 7-12
sample programs for, 7-13

READTRANSACTION parameters
in TPS, 5-22

RECEIVE statement
in DCIENTRYPPOINT parameters, 3-14
in SDF, 6-17
in Transaction Server, 1-5, 3-22, 3-23
using DCI library
for specifying an end-indicator
option, 3-14
for specifying whether to wait for a
message, 3-14

record area
in Enterprise Database Server
setup, 4-4

record type attribute
in Enterprise Database Server, 4-24

Record Type field
in Enterprise Database Server, 4-23

RECREATE statement
in Enterprise Database Server, 1-7, 4-62

REDEFINES clause

in SDF, 1-8, 6-3

in SDF Plus, 1-9

REDEFINES option

in SDF

data-description entry, 6-4

used with Transaction Server, 6-17

in SDF Plus, 7-7

reentrant capability

in Enterprise Database Server

Accessroutines, 4-18

referencing database items, 4-4

RELEASE statement

in ADDS, 2-10

remaps

in Enterprise Database Server, 4-12

REMOVE statement

in Enterprise Database Server, 1-7, 4-64

RESET_PROGRAM_STATISTICS service

function parameters in Transaction
Server, 3-48

resetting flags

in SDF, 6-15

RESTART COBOL74 field name

input header in Transaction Server, 3-8

restart data set (RDS)

Enterprise Database Server option used
in Transaction Server, 3-52

in Transaction Server, 3-52

Restart field name

input header in Transaction Server, 3-8

Retain Transaction Mode field name

output header in Transaction Server, 3-10

RETAINTRANSACTIONMODE COBOL74 field
name

output header in Transaction Server, 3-10

RETURN statement

in ADDS, 2-10

RETURNLASTADDRESS parameters

in TPS, 5-22

RETURNLASTRESPONSE entry point

in TPS, 5-23

RETURNRESTARTINFO parameters

in TPS, 5-23

run time

modifying database titles in Enterprise
Database Server during, 4-18

S

- SAME RECORD AREA clause
 - in SDF, 1-8, 6-3
 - in SDF Plus, 1-9
- SAME RECORD AREA option
 - in SDF
 - data-description entry, 6-4
 - with Transaction Server, 6-17
 - in SDF Plus, 7-7
- sample programs
 - in ADDS, 2-15
 - in Enterprise Database Server TPS, 5-31
 - in SDF, 6-19
 - in SDF Plus, 7-30
- SAVE TRANSACTION POINT statement
 - in Enterprise Database Server, 1-7, 4-66
- SAVEINPUTTR formal procedure
 - in TPS, 5-29
- SAVERESPONSETR formal procedure
 - in TPS, 5-30
- Screen Design Facility (SDF), 6-1
 - and ADDS, 6-1
 - and Transaction Server, 6-1
 - default form transmittal, 6-18
 - first word into conversation area, 6-18
 - for RECEIVE statement, 6-17
 - for REDEFINES option, 6-17
 - for SAME RECORD AREA option, 6-17
 - form key movement, syntax of, 6-18
 - interface function, 6-11
 - interface guidelines, 6-17
 - invoking a formlibrary, 6-17
 - using SDF forms, 6-17
 - cursor positioning, 6-13
 - data item characteristics, identifying, 6-3
 - default values
 - when reading forms, 6-7
 - when writing forms, 6-10
 - dictionary identification, 6-2
 - DICTIONARY statement, 6-2
 - error handling when reading forms, 6-8
 - error recovery when writing forms, 6-10
 - file status update when reading forms, 6-7
 - flag groups for programmatic control, 6-15
 - flag setting for programmatic controls, 6-12
 - for FROM DEFAULT FORM option, 6-10
 - for FROM DICTIONARY clause, 6-3
 - FORM-KEY function for Transaction Server interface, 6-11
 - formlibrary data-description entry rules, 6-5
 - formlibrary name in data-description entry, 6-4
 - highlight flag, 6-14
 - invoking form descriptions in formlibrary, 6-3
 - level number in data-description entry, 6-3
 - LOW-VALUES figurative constant with programmatic flags, 6-15
 - MOVE statement
 - when reading forms, 6-7
 - when using WRITE FORM statement, 6-10
 - naming convention for programmatic control, 6-12
 - ON ERROR condition, 6-8
 - program interfaces used with, 6-1
 - programmatic control
 - flag information (table), 6-13
 - flag settings (table), 6-13
 - flag suffixes (table), 6-13
 - READ FORM statement, 6-6, 6-7
 - reading forms from station to program, 6-6
 - record moving when reading forms, 6-7
 - REDEFINES clause, 6-3
 - REDEFINES option in data-description entry, 6-4
 - redefining formlibraries, 6-3
 - SAME RECORD AREA clause, 6-3
 - SAME RECORD AREA option in data-description entry, 6-4
 - sample program
 - for reading forms, 6-19
 - for using message keys, 6-27
 - USING clause
 - in READ FORM statement, 6-6
 - in WRITE FORM statement, 6-9
 - using programmatic controls and a remote file, 6-23
 - WRITE FORM statement, 6-9
 - writing forms from program to station, 6-9
- Screen Design Facility Plus (SDF Plus), 7-1
 - and ADDS, 7-1
 - and Transaction Server, 7-1, 7-26
 - data-description entry in, 7-6
 - default form records, 7-12
 - DEFAULT FORM statement, 7-12

- dictionary identification, 7-5
- DICTIONARY statement, 7-5
- direct windows used with Transaction Server, 7-26
- DIRECTORY clause, 7-7
- error handling when reading forms, 7-12
- error recovery when writing forms, 7-16, 7-18
- extensions (list), 1-9, 7-1
- form record library, 7-2, 7-3
 - data description, 7-8
 - importing, 7-7
 - storage directory, 7-7
- form record numbers, 7-3
 - obtaining, 7-23
- form records, 7-2
 - I/O operations for, 7-3
 - invoking all descriptions of, 7-7
 - invoking in form record library, 7-7
 - obtaining from library, 7-6
- FROM DICTIONARY clause, 7-6
- GLOBAL option, 7-7
- global remote file
 - selecting, 7-10
- guidelines for Transaction Server
 - interface, 7-27
- interface elements, list of, 7-2
- invoking data descriptions in, 7-6
- MOVE statement, 7-12
- MOVE statement, comparison with, 7-18
- moving data when writing forms, 7-18
- ON-ERROR condition in writing
 - forms, 7-16, 7-18
- ON-ERROR option, 7-12
- preassigned transaction number
 - return, 7-25
- READ FORM statement, 7-11
- READ FORM statement, 7-10
- reading forms, 7-10
- reading specific form records, 7-11
- record default values, writing forms, 7-18
- REDEFINES option, 7-7
- SAME RECORD AREA option, 7-7
- sample program for, 7-30
- SELECT statement, 7-10
- sending and receiving messages with Transaction Server, 7-27
- sending text messages through the Transaction Server, 7-29
- sending transaction errors, 7-28
- SEPARATE RECORD AREA option, 7-7
- sharing memory in data descriptions, 7-7
- subprogram reference in library, 7-7
- transaction errors with Transaction Server, 7-27
- transaction numbers, 7-3
 - returning, 7-25
- transaction type record, identifying, 7-3
- USING clause
 - in WRITE FORM TEXT, 7-21
- USING clause, 7-11
- using the FORMRECNUM attribute, 7-23
- using the FROM DICTIONARY option, 7-7
- using the Transaction Server input/output headers, 7-26
- using the TRANSNUM attribute, 7-25
- VERSION clause, 7-7
- WRITE FORM statements, 7-13
- writing forms, 7-13
- writing text arrays to a remote file, 7-21
- Screen Design Facility Plus (SDF Plus)
 - WRITE FORM TEXT statement, 7-21
- SD, *See* sort-merge file description (SD)
 - entry
- SDF Plus, *See* Screen Design Facility Plus (SDF Plus)
- search rules
 - in ADDS, 2-3
- SECURE statement
 - in Enterprise Database Server, 1-7, 4-67
- Security Designator field name
 - input header in Transaction Server, 3-8
- SECURITYDESGCOBOL74 field name
 - input header in Transaction Server, 3-8
- SEEKTRANSACTION parameters
 - in TPS, 5-23
- segmented output
 - using the SEND statement in Transaction Server, 3-26
- SELECT statement
 - in ADDS, 1-4, 2-8
 - for INVOKE clause, 2-8
 - in SDF Plus, 7-10
- selection expressions
 - in Enterprise Database Server, 1-7, 4-20
 - for AT option, 4-21
 - for WHERE option, 4-21
 - key condition in, 4-21
- SEND statement
 - in DCIENTRYPPOINT parameters, 3-14, 3-16
 - in Transaction Server, 1-5, 3-24
 - affecting the MCS, 3-25
 - SDF FORM-KEY function used
 - with, 6-11
 - using DCI library

- for designating an advancing value, 3-14
 - for specifying advancing control, 3-14
 - for specifying an end-indicator option, 3-14
 - SEPARATE RECORD AREA clause
 - in SDF Plus, 1-9
 - SEPARATE RECORD AREA option
 - in SDF Plus, 7-7
 - service functions
 - in Transaction Server
 - calling by name, 3-33
 - calling by value, 3-34
 - hyphenating names, 3-30
 - passing parameters to, 3-35
 - translating a designator, 3-7, 3-9
 - mnemonics used in (table), 3-31
 - names of (list), 3-30
 - Set Next Input Agenda field name
 - output header in Transaction Server, 3-10
 - set reference
 - in Enterprise Database Server, 4-7, 4-11
 - SET statement
 - in Enterprise Database Server, 1-7, 4-68
 - set, qualifying names of, 4-2
 - SETNEXTINPUTAGENDA COBOL74 field name
 - output header in Transaction Server, 3-10
 - shared lock
 - in Enterprise Database Server, 4-67
 - SIZE option
 - in Transaction Server, 3-5
 - sort-merge file description (SD) entry
 - in ADDS
 - level indicator, 1-4
 - space fill
 - aligning Transaction Server messages
 - in RECEIVE statement, 3-23
 - in SEND statement, 3-25
 - SPECIAL-NAMES paragraph
 - in ADDS
 - for identifying dictionary, 2-6
 - in Transaction Server for SEND statement, 3-26
 - specify flag
 - in SDF, 6-14
 - STATION COBOL74 field name
 - input header in Transaction Server, 3-8
 - Station Designator field name
 - input header in Transaction Server, 3-8
 - STATION_TABLE_ADD parameters
 - in Transaction Server, 3-48
 - STATION_TABLE_INITIALIZE parameters
 - in Transaction Server, 3-49
 - STATION_TABLE_SEARCH parameters
 - in Transaction Server, 3-49
 - status
 - for invoking entities
 - in ADDS, 2-2
 - Status field name
 - input header in Transaction Server, 3-8
 - output header in Transaction Server, 3-10
 - Status Value field
 - in Transaction Server for SEND statement, 3-24
 - status word
 - in Enterprise Database Server
 - for exception handling, 4-74
 - significance in the CLOSE statement, 4-36
 - STATUSVALUE COBOL74 field name
 - input header in Transaction Server, 3-8
 - output header in Transaction Server, 3-10
 - STORE statement
 - in Enterprise Database Server, 1-7, 4-71
 - structure number
 - in Enterprise Database Server, 4-76
 - STRUCTURE phrase
 - in Enterprise Database Server
 - for FREE statement, 4-50
 - subscript
 - in TPS
 - for CREATE statement, 5-8
 - instructions for using, 5-13
 - using in record formats, 5-9
 - SWITCHTRFILE parameters
 - in TPS, 5-23
 - SYNC option
 - in Enterprise Database Server
 - for END-TRANSACTION statement, 5-30
 - synchronized recovery
 - in Transaction Server
 - with Enterprise Database Server, 4-37
 - sample program, 3-51
 - syncpoint
 - in Enterprise Database Server
 - for END-TRANSACTION statement, 4-44
- T**
- TANKTRANSACTION parameters
 - in TPS, 5-23
 - TANKTRNORESTART parameters

- in TPS, 5-23
- TB, *See* transaction base
- TERMINAL optional word
 - in DISABLE statement, 3-19
- TEST status
 - in ADDS, 2-2
- TEST_DESIGNATORS parameters
 - in Transaction Server, 3-50
- Text Length field name
 - input header in Transaction Server, 3-8
 - output header in Transaction Server, 3-10
- TEXTLENGTH COBOL74 field name
 - input header in Transaction Server, 3-8
 - output header in Transaction Server, 3-10
- TFL, *See* Transaction Formatting Language (TFL)
- TIME(6) field
 - in Transaction Server, 3-7, 3-36
- TIMESTAMP COBOL74 field name
 - input header in Transaction Server, 3-8
- Timestamp field name
 - input header in Transaction Server, 3-8
- TPS, *See* transaction processing system (TPS)
- trancode, *See* transaction processing
- Transaction, 3-26
- transaction base
 - alternate internal names, 5-3
 - bound programs, 5-5
 - declarations in, 1-7, 5-3
 - global declaration in, 5-5
 - invoking, 5-3
 - USING clause in declaration, 5-5
- transaction compile-time functions
 - in TPS, 5-18
- transaction errors
 - with SDF Plus, 7-27
- Transaction Formatting Language (TFL)
 - declaring a transaction base, 5-3
 - in TPS item interpretation, 5-2
- transaction functions
 - in DCI library, 3-14
- transaction library
 - in TPS
 - declaring entry points, 5-5
 - for CALL statement, 5-5
 - passing parameters to update library, 5-25
 - using entry points, 5-20, 5-22
 - using the INTEGER function, 5-20
 - using the STRING function, 5-20
- transaction numbers
 - attribute in SDF Plus, 1-9, 7-25
 - in SDF Plus, 7-3
- transaction processing
 - routines used in update library, 5-23
 - using Transaction Server for, 3-1, 3-9
- transaction processing system (TPS), 5-28
 - ACCESSDATABASE entry point in update library, 5-23
 - accessing transaction record items, 5-12
 - Accessroutines
 - in update library, 5-28
 - alternate internal names in transaction base, 5-3
 - assigning record variables, 5-11
 - banking transaction, sample program for, 5-40
 - BEGIN-TRANSACTION statement in update library, 5-28
 - CALL statement
 - in transaction base library entry, 5-5
 - in Use procedures, 5-24
 - calling entry points in, 5-22
 - COBOL74 constructs, passing parameters with, 5-20
 - compiler call procedure with Enterprise Database Server, 5-29
 - compile-time functions (table), 5-19
 - CORRESPONDING phrase, special rules for, 5-12
 - CREATE statement in record formats, 5-8
 - DASDL, sample program for, 5-31
 - declarations in transaction base, 5-3
 - declaring library entry point in transaction base, 5-5
 - declaring Use procedures, 5-24
 - END-TRANSACTION statement, 5-30
 - Enterprise Database Server interface
 - library routines, 5-23
 - recovery synchronization, 5-26
 - sample program for, 5-31
 - entry point to update library, 5-23
 - entry points (table), 5-22
 - exceptions
 - return in update library, 5-27
 - when database is open, 5-27, 5-28, 5-29, 5-30
 - extensions (list), 1-7
 - GLOBAL option in transaction record, 5-7
 - GLOBAL statement in Use procedures, 5-24
 - hyphenated identifiers in, 5-2
 - inquiring about control items, 5-16
 - INQUIRY option in update library, 5-27
 - MID-TRANSACTION statement, 5-29

- MOVE CORRESPONDING statement, 5-12
- MOVE statement to assign variables, 5-11
- OCCURS clause in transaction record, 5-7
- ON EXCEPTION option
 - in update library, 5-27, 5-28, 5-29, 5-30
- OPEN statement, 5-26
- opening database in update library, 5-26
- parameter passing
 - from ALGOL library, 5-25
- passing display item to string
 - parameter, 5-20
- passing numeric item to integer
 - parameter, 5-20
- passing transaction record variable, 5-10
- procedures in update library, 5-24
- program types used in, 5-1
- programming for a Enterprise Database Server interface, 5-23
- qualification of data items, 5-15
- removing a program from transaction state, requirements for, 5-30
- sample programs for, 5-31
- SAVEINPUTTR procedure, 5-29
- SAVERESPONSETR procedure, 5-30
- subscripts with OCCURS clause, 5-13
- SYNC option in END-TRANSACTION statement, 5-30
- syncpoint in transaction removal, 5-30
- TFL item interpretations (table), 5-2
- transaction base declaration in, 5-3
- transaction base global declaration in, 5-5
- transaction compile-time functions, 5-18
- Transaction Formatting Language (TFL), 5-2
- transaction library entry points, 5-20
- transaction record
 - creating, 5-6, 5-8
 - declared in host, 5-7
 - restrictions, 5-10
- transaction state removal procedure, 5-30
- transaction use procedures in, 5-24
- TRUPDATE option in OPEN statement, 5-26
- update library
 - parameter passing, 5-25
 - program conventions, 5-23
 - recovery in, 5-26
- USING clause in transaction base declaration, 5-5
 - using subscripts in record formats, 5-9
- transaction record control item reference in TPS, 1-8
- transaction record reference entry
 - in TPS, 1-8
- transaction records
 - accessing items in, 5-12
 - assigning variables in, 5-11
 - association with bases and subbases, 5-6
 - control items for, 5-16
 - creating, 5-8
 - declaring variables, 5-6
 - passing variables as parameters, 5-10
 - restrictions for use, 5-10
 - special rules in CORRESPONDING phrase, 5-12
 - using subscripts in format, 5-9
 - variable declaration for, 1-8
 - when declared in host program, 5-7
- Transaction Server
 - and SDF Plus
 - with direct windows, 7-26
- Transaction Server, *See* Transaction Server
- ACCEPT MESSAGE COUNT statement, 3-18
- AFTER ADVANCING phrase
 - in SEND statement (format 2), 3-25
- and Enterprise Database Server, 3-2
- BEGIN-TRANSACTION statement
 - in, 4-32
- DMTERMINATE statement in, 4-44
- END-TRANSACTION statement in, 4-45
- and SDF, 6-1
 - FORM-KEY function, 6-11
 - steps for using forms, 6-17
 - with direct windows, 6-17
- and SDF Plus, 7-1
- BEFORE ADVANCING phrase
 - in SEND statement (format 2), 3-25
- Boolean items in, 3-6
- calling service functions in, 3-33
- carriage control in, 3-28
- communication constructs used in, 3-18
- converting
 - a designator to a designator name in, 3-45
 - a name variable in, 3-41
 - a timestamp in, 3-36
- data structure, with no connection, 3-46
- declaring headers for, 3-4
- designator and integer values in, 3-7
- designators in, 3-6, 3-31
- DISABLE statement, 3-19
- ENABLE statement, 3-20
 - key values, examples of, 3-22
- end-of-group indicator (EGI) option
 - in SEND statement (format 2), 3-25

- end-of-message indicator (EMI) option
 - in SEND statement (format 2), 3-25
- Enterprise Database Server database
 - update, sample program, 3-51
- extensions of (list), 1-5
- FROM phrase
 - in SEND statement (format 1), 3-24
 - in SEND statement (format 2), 3-25
- functions of, 3-1
- getting a buffer designator in, 3-37
- getting a designator array in, 3-38
- getting an error message in, 3-40
- initializing a program interface
 - example, 3-13
- input header fields (list), 3-8
- input header fields in, 3-6, 3-7
- input header tasks, 3-7
- input headers
 - fields and types (list), 3-8
 - using with SDF Plus, 7-26
- INPUT phrase
 - in DISABLE statement, 3-19
 - in ENABLE statement, 3-20, 3-21
- KEY value
 - in DISABLE statement, 3-19
 - in ENABLE statement, 3-20, 3-21
- linking program to, 3-11
- MESSAGE phrase in RECEIVE statement, 3-22
- messages
 - delivery confirmation, 3-11
 - receiving, 3-11
 - releasing, 3-24
 - sending, 3-11
 - using output header fields, 3-9
- NO DATA option in RECEIVE statement, 3-22
- obtaining
 - a specific designator, 3-39
 - a specific integer, 3-43
 - an array of integers, 3-42
 - an EBCDIC string, 3-47
- output headers, 3-9
 - field names (table), 3-10
 - fields and types (table), 3-10
 - fields in, 3-6
 - using with SDF Plus, 7-26
- OUTPUT phrase
 - in DISABLE statement, 3-19
 - in ENABLE statement, 3-20, 3-21
- passing mnemonics for a numeric result in, 3-34
- program interface, 3-1
- RECEIVE statement, 3-22
- releasing messages, 3-24
- representing a structure test, 3-50
- sample programs with a Enterprise Database Server database, 3-51
- SEND statement, 3-24
 - SPECIAL-NAMES paragraph in, 3-26
- service function
 - mnemonics in (table), 3-31
 - names of (list), 3-30
 - parameters in, 3-35
- station designator, adding to a table, 3-48
- station index values, initializing table
 - for, 3-49
- transferring data with the RECEIVE statement in, 3-22
- updating input headers of, 3-18
- using
 - in RECEIVE statement, 3-23
 - using a service function in, 3-7, 3-9
 - using an ALGOL shell with, 3-2
 - using DCI library
 - for specifying device type, 3-14
- WITH DATA option in RECEIVE statement, 3-22
- Transaction Server headers
 - as the Transaction Server interface, 3-3
 - declaring, 1-5, 3-4
 - fields of input header, 3-7
 - fields of input header (list), 3-8
 - fields of output header, 3-9
 - fields of output header (table), 3-10
 - using with SDF Plus, 7-26
- Transaction Server interface, declaring, 3-4
- Transaction Server TRANSACTION statement in, 4-28
- transaction types
 - in SDF Plus, 7-3
- transaction updating
 - in Enterprise Database Server, 4-28
- transaction Use procedures
 - in TPS, 5-24
- TRANSPARENT COBOL74 field name
 - input header in Transaction Server, 3-8
 - output header in Transaction Server, 3-10
- Transparent field name
 - input header in Transaction Server, 3-8
 - output header in Transaction Server, 3-10
- TRUE clause
 - in Enterprise Database Server
 - for COMPUTE statement, 4-38
- TRUPDATE option
 - in TPS for OPEN statement, 1-7, 5-26

TRUSERIDSTRING parameters
 in TPS, 5-23
types
 in Transaction
 items mapped to COBOL74, 3-6

U

unindexed descriptor
 in DCIENTRYPPOINT library object
 parameter length, 3-14
update library
 in TPS
 ACCESSDATABASE entry point, 5-23
 declaring transaction Use
 procedures, 5-24
 passing parameters, 5-25
 programming conventions, 5-23
 receiving parameters, 5-25
 using, 5-23
UPDATE phrase
 in Enterprise Database Server
 for OPEN statement, 4-60
USE statement
 in TPS, 1-8
 to declare Use procedures, 5-24
USER clause
 in ADDS, 2-4
 replacing with DIRECTORY clause, 2-4
USERCODE COBOL74 field name
 input header in Transaction Server, 3-8
Usercode Designator field name
 input header in Transaction Server, 3-8
user-defined COBOL74 field name
 input header in Transaction Server, 3-8
 output header in Transaction Server, 3-10
user-defined Conversation Area field name
 input header in Transaction Server, 3-8
 output header in Transaction Server, 3-10
user-defined words (list), A-1
USING clause
 in data set references, 4-10
 in SDF
 for READ FORM statement, 6-6
 for WRITE FORM statement, 6-9
 in SDF Plus
 for READ FORM statement, 7-11
 for WRITE FORM TEXT statement, 7-21
 in TPS
 for passing parameters with COBOL74
 constructs, 5-21

 for transaction base declaration, 5-5
USING option
 in Enterprise Database Server
 for invoking data sets, 4-11

V

VALUE clause
 in Transaction Server
 for CALL statement, 1-5
VALUE OF TITLE clause
 in Enterprise Database Server
 for database declaration, 4-7
VALUE parameter
 in CALL statement for Transaction
 Server, 1-5, 3-34
 naming convention with
 DCILIBRARY, 3-33
variable-format records
 problem with using in Enterprise
 Database Server, 4-5
variables
 passing as parameters in TPS, 5-10
VERSION clause
 in ADDS, 1-4, 2-3
 for data description, 2-3
 for DICTIONARY statement, 2-3
 for file description, 2-3
version number
 assigning in ADDS, 2-3
VERSION option
 in ADDS
 for DICTIONARY statement, 2-7
 for SELECT statement, 2-8
 in SDF Plus, 7-7
VERSIONERROR, clarifying NULL value, 4-55
VIA option
 in Enterprise Database Server
 for selection expression, 4-20
VT Flag field name
 input header in Transaction Server, 3-8
 output header in Transaction Server, 3-10
VTFLAG COBOL74 field name
 input header in Transaction Server, 3-8
 output header in Transaction Server, 3-10

W

WFL, *See* Work Flow Language (WFL)
WHERE option

- in Enterprise Database Server
 - for selection expressions, 4-21
- windows in Transaction Server
 - using to send messages, 3-9
 - using with SDF Plus, 7-26
- WITH DATA option
 - in Transaction Server
 - for RECEIVE statement, 3-22
- Work Flow Language (WFL)
 - in database equation operations, 4-18
 - overriding database titles in, 4-18
- WRITE FORM FOR ERROR MESSAGE
 - statement
 - in SDF Plus
 - sample programs for, 7-21
- WRITE FORM statement
 - in SDF, 1-8, 6-9
 - for DEFAULT FORM option, 6-10
 - for USING clause, 6-10
 - in SDF Plus, 1-9

- WRITE FORM statement (format 1)
 - in SDF Plus
 - for ON ERROR MESSAGE clause, 7-16
 - sample programs for, 7-16
- WRITE FORM statement (format 2)
 - in SDF Plus
 - for DEFAULT FORM option, 7-18
 - for FOR ERROR MESSAGE clause, 7-18
 - sample programs for, 7-18
- WRITE FORM statements
 - in SDF Plus, 7-13
- WRITE FORM TEXT statement
 - in SDF Plus, 1-9, 7-21
 - sample programs for, 7-22

X

- XATMI buffer
 - LENGTH, 3-43
 - type and subtype, 3-47

Special Characters

- 01-level indicator
 - in TPS record variable declarations, 5-7

