

**MCP/AS**

**DMALGOL**  
**Programming Reference Manual**

MCP 12.0

April 2008





**MCP/AS**

**DMALGOL**  
**Programming Reference Manual**

MCP 12.0

April 2008

8600 0874-203

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.

MCP/AS  
DMALGOL  
**Programming Reference  
Manual**  
**MCP 12.0**

MCP/AS  
DMALGOL  
**Programming  
Reference  
Manual**  
**MCP 12.0**

8600 0874-203

8600 0874-203

Bend here, peel upwards and apply to spine.



# Contents

## **Section 1. DMALGOL Concepts**

Using DMALGOL Safely .....	1-1
Unsafe Extensions .....	1-2
Unsafe Functions and Statements .....	1-2
Using the Compile-Time Facility .....	1-4
Environments .....	1-5
Reference Variables .....	1-7
ONCE-ONLY Compilation of Procedures .....	1-7

## **Section 2. DMALGOL Extensions to ALGOL Declarations and Statements**

Array Declaration .....	2-1
Concurrency Control Facility .....	2-2
ABORTTRANSACTION Statement .....	2-2
CANCELTRPOINT Statement .....	2-3
FREE Statement .....	2-4
LOCK Statement .....	2-5
SAVETRPOINT Statement .....	2-6
SECURE Statement .....	2-7
Compile-Time Facility .....	2-8
'DEFINE Statement .....	2-8
'FOR Statement .....	2-9
'INCLUDE Statement .....	2-10
'INVOKE Statement .....	2-10
NODE Declaration .....	2-11
'PRINT, 'ERROR, 'DISPLAY Statements .....	2-12
PROPERTY Declaration .....	2-13
DMIO File Attribute .....	2-14
DYNAMIC DATABASE Declaration .....	2-14
EXCEPTION PROCEDURE Declaration .....	2-15
Nonlocal Message References .....	2-16
ONCE-ONLY Compilation .....	2-17
OPEN Statement .....	2-19
Procedure Declaration .....	2-20
Reference Variables Declaration and Assignment .....	2-21
Array References .....	2-21
Procedure Reference Arrays .....	2-22
Procedure References .....	2-23
File References .....	2-24
Up-Level Event References .....	2-25

### **Section 3. DMALGOL-Specific Functions and Statements**

ALLOW Statement .....	3-1
DISALLOW Statement .....	3-1
DMINQ Functions.....	3-2
DMINQ Interface Function .....	3-2
Procedure: Pathfinder (A[0] = 1) .....	3-4
Procedure: Set to Beginning (A[0] = 10) .....	3-5
Procedure: Data Finder (A[0] = 11) .....	3-5
Procedure: GETDATA (A[0] = 12) .....	3-5
Procedure: DMSREAD (A[0] = 13) .....	3-6
Procedure: Get Link (A[0] = 14) .....	3-6
Procedure: Store Current (A[0] = 16) .....	3-6
Procedure: Free Current (A[0] = 17) .....	3-6
Procedure: Set or Check Date-Timestamp for TPS (A[0] = 18) .....	3-7
Procedure: Clear TPS Flag and Date-Timestamp (A[0] = 19) .....	3-7
Procedure: Return Last Transaction Address (A[0] = 20) .....	3-7
Procedure: Get Status of Abort (A[0] = 21) .....	3-8
Procedure: Rerun Finished – Reset TPS Abort Pending (A[0] = 22) .....	3-8
Procedure: Create (A[0] = 23) .....	3-8
Procedure: Delete Current (A[0] = 24) .....	3-8
Procedure: Get Statistics Information (A[0] = 25) .....	3-9
Procedure: Visible DBS Message (A[0] = 26) .....	3-18
Procedure: Return Displayed Messages (A[0] = 27) .....	3-18
Procedure: Return Software Version (A[0] = 28) .....	3-19
Procedure: Clear TPS Address (A[0] = 29) .....	3-19
Procedure: Return Standard Data Set DATAEOFs (A[0] = 30) .....	3-19
Procedure: DATAEOF Values (A[N+1]) .....	3-20
Procedure: GETDATA with KEYCOMPARE (A[0] = 31) .....	3-20
Procedure: Off-line Dump (A[0] = 32) .....	3-20
Procedure: Set Cursor (A[0] = 33) .....	3-21
Procedure: Set Current Path to Another Cursor's Path (A[0] = 34) .....	3-21
DMINQ Arrays .....	3-22
DMPROCOFF Statement .....	3-23
DMPROCREF Function.....	3-24
DMSCAUSE Statement.....	3-25
DMSFREE Statement.....	3-26
DMSFREEZE Function.....	3-27
DMSIBINDEX Intrinsic .....	3-28
DMSLOCALREF Statement .....	3-28
DMSUPDATEDISKHEADER Statement .....	3-29
DMSWAIT Function.....	3-29
DMTRANSLOCK Statement.....	3-31
DSED FUNCTION .....	3-31
DSWAIT and DSWAITANDRESET Functions .....	3-32



ENVIRONMENT Declaration .....	3-33
LOCK [DSABLE] Interlock Function .....	3-33
MEMORYDUMP Statement .....	3-34
PURGEASM Statement .....	3-35
READLOCKNOPURGE Function .....	3-35
SIBOFFSET Function .....	3-36
TPS TRANSACTION RECORD CONTROL ITEM ASSIGNMENT Statement .....	3-37
TRY Statement .....	3-37

## **Section 4. Compiler Control Options**

## **Appendix A. Understanding Railroad Diagrams**

Railroad Diagram Concepts .....	A-1
Paths .....	A-1
Constants and Variables .....	A-2
Constraints .....	A-3
Vertical Bar .....	A-3
Percent Sign .....	A-3
Right Arrow .....	A-4
Required Item .....	A-4
User-Selected Item .....	A-5
Loop .....	A-5
Bridge .....	A-6
Following the Paths of a Railroad Diagram .....	A-7
Railroad Diagram Examples with Sample Input .....	A-8

## **Appendix B. Reserved Words**

<b>Index</b> .....	<b>1</b>
--------------------	----------



# Tables

Section 3-1 DMINQ Array Values and Procedures.....	3-3
Section 3-2 Word Values and Descriptions for the Get Statistics Information Procedure .....	3-9
Section 3-3 Global Static Statistics [word 1 = 0, word 2 = 0] .....	3-11
Section 3-4 Global Dynamic Statistics (word 1 = 0, word 2 = 1) .....	3-12
Section 3-5 Structure Static Statistics (word 1 = <structure number>, word 2 = 0).....	3-15
Section 3-6 Structure Dynamic Statistics.....	3-16
Appendix A-1 Elements of a Railroad Diagram .....	A-2
B-1.    DMALGOL Reserved Words .....	B-1



# About This Manual

## Purpose

This reference manual describes the constructs that are unique to the DMALGOL programming language. DMALGOL is an implementation language which is based on Unisys Extended ALGOL, a high-level, structured programming language designed for enterprise server systems. Unisys Extended ALGOL implements virtually all of ALGOL 60 and, in addition, includes extensions that enhance the basic capabilities of the language.

The DMALGOL extensions have been implemented to support enterprise server systems software; these extensions allow the user to write and update Data Management System II (DMSII) software and other specialized system programs. DMALGOL also includes all BDMSALGOL and DCALGOL extensions.

Users should realize that DMALGOL is a powerful implementation language that allows them to perform tasks directly affecting system software. They should also be aware that new features can be added at any time and existing features can be changed or deleted without notice. For these reasons, DMALGOL is not intended for general use, and users should not rely on it as an application programming language.

## Scope

This manual provides reference information for systems programmers who must use DMALGOL to recompile products supplied by Unisys. It includes syntax, explanation, and examples for those constructs unique to DMALGOL. It is designed to be used in conjunction with the *ALGOL Programming Reference Manual, Volume 1: Basic Implementation* and the *ALGOL Programming Reference Manual, Volume 2: Product Interfaces*, which describe the other constructs that can be used in DMALGOL programs.

## Audience

The manual is intended for systems programmers who must use DMALGOL to develop, update, and maintain system software for DMSII and certain other Unisys products.

## Prerequisites

This manual is written for systems programmers who are familiar with Unisys Extended ALGOL, as described in the *ALGOL Programming Reference Manual, Vol. 1: Basic Implementation* and the *ALGOL Programming Reference Manual, Volume 2: Product Interfaces*. It assumes also that readers are familiar with the product or products for which they are developing or recompiling system software and with the implementation of Accessroutines and MCP-based systems stack architecture.

## How to Use This Reference Manual

The constructs in this manual can be referenced randomly. It is advisable to have a copy of the *ALGOL Programming Reference Manual, Vol. 1: Basic Implementation* for reference while using this manual because some explanations refer to that manual.

## Organization

Section 1 of this manual explains some of the concepts of the DMALGOL programming language. Sections 2 and 3 describe the functions and statements used in DMALGOL that are not common to ALGOL. Section 4 provides compiler control options. The sections and appendixes are described as follows:

### **Section 1. DMALGOL Concepts**

This section provides an introduction to the DMALGOL programming language and discusses some of the features that make it unique.

### **Section 2. DMALGOL Extensions to ALGOL Declarations and Statements**

This section describes the extensions to certain ALGOL constructs that are available in DMALGOL.

### **Section 3. DMALGOL-Specific Functions and Statements**

This section describes the functions and statements that are unique to DMALGOL. These constructs are ordered alphabetically by function or statement name.

### **Section 4. Compiler Control Options**

This section describes the two compiler control options, WARNSAFE and WARNALLUNSAFE, that DMALGOL recognizes in addition to the standard ALGOL compiler control options.

### Appendix A. Understanding Railroad Diagrams

This appendix explains how to read the railroad diagrams used in this manual to describe DMALGOL syntax.

### Appendix B. Reserved Words

This appendix lists the identifiers that need not be declared in a DMALGOL program before they are used, if they appear in recognized contexts.

## Related Product Information

Unless otherwise stated, all documents referred to in this publication are MCP/AS documents. The titles have been shortened for increased usability and ease of reading.

The following documents are included with the software release documentation and provide general reference information:

- The *Glossary* includes definitions of terms used in this document.
- The *Documentation Road Map* is a pictorial representation of the Product Information (PI) library. You follow paths through the road map based on tasks you want to perform. The paths lead to the documents you need for those tasks. The Road Map is available on the PI Library CD-ROM. If you know what you want to do, but don't know where to find the information, start with the Documentation Road Map.
- The *Information Availability List* (IAL) lists all user documents, online help, and HTML files in the library. The list is sorted by title and by part number.

The following documents provide information that is directly related to the primary subject of this publication.

### ***ALGOL Programming Reference Manual, Volume 1: Basic Implementation***

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

### ***ALGOL Programming Reference Manual, Volume 2: Product Interfaces***

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





# Section 1

## DMALGOL Concepts

### 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) 18567351. 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/18567351>

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

The DMALGOL language is based on Unisys Extended ALGOL. Its purpose is to provide users with a tool to develop, update, and maintain software for the Data Management System II (DMSII) and certain other enterprise server products. DMALGOL consists of ALGOL with extensions to allow programs to retrieve information from Data and Structure Definition Language (DASDL) description files and to build the special stack structures required by DMSII.

This section discusses some of the features that make DMALGOL a unique language. The specific constructs that are used in DMALGOL are described in the later sections of this manual.

## Using DMALGOL Safely

DMALGOL is a powerful programming language that gives users greater privileges than they normally receive through ALGOL. For example, programs written in DMALGOL can perform tasks involving system programs that should not be made available in application programs. For this reason, it is advisable to use DMALGOL only when it is necessary to recompile DMSII software. DMALGOL should not be used to write application programs.

Programs using DMALGOL constructs can have far-reaching effects, and users must ensure that the constructs are used correctly to avoid problems. To assist users in identifying constructs that require extra care in their use, the DMALGOL compiler designates some constructs as “unsafe.” A construct is considered unsafe when its misuse might lead to violation of system integrity or security policy, or when it has not been examined in enough detail to rule out the possibility of such misuse.

Programs that use unsafe constructs are marked as “nonexecutable-unsafe.” Such programs cannot be run on some system configurations without special privileged action. If unsafe programs or program units are used for binding, the bound code file is also marked unsafe.

An unsafe code file can be executed as a by-function library if the file has been installed using the SL (Support Library) system command. It can be made executable under more general conditions by the MP (Mark Program) system command. For a discussion of unsafe code files, including the conditions under which they can be executed, see the *Security Administration Guide*. For additional information on the SL and MP commands, see the *System Commands Operations Reference Manual*.

At the end of a compilation in which an unsafe construct occurs, a warning message is generated to ERRORFILE if ERRLIST is TRUE, and to LINE if a listing is being generated. The number of occurrences of unsafe constructs is included in the trailer on the LINE file if LINE is used.

Specific usage of unsafe constructs can be flagged through the use of the WARNUNSAFE and WARNALLUNSAFE compiler control options. For more information on these control options, see Section 4.

## Unsafe Extensions

The following DMALGOL extensions to ALGOL are considered unsafe. For more information on these constructs, see Section 2.

- SAVE Array declaration option
- PROCEDURE declaration when used for ONCE-ONLY compilation
- PROCEDURE REFERENCE declaration
- OPEN statement options MAPPER, INQUIRYSEMANTIC, and UPDATESEMANTIC
- DMIO attribute
- DYNAMIC DATABASE declaration
- Nonlocal message references
- Array references for transaction input or output records
- Up-level ARRAY reference assignments
- Up-level FILE reference assignments
- Up-level READ/WRITE event designations
- “: PROTECTED” clause

## Unsafe Functions and Statements

The following DMALGOL functions and statements are considered unsafe. For more information on these constructs, see Section 3.

- DISALLOW statement
- DMPROCOFF statement
- DMPROCREF function
- DMSCAUSE statement
- DMSFREE statement

- DMSFREEZE function
- DMSLOCALREF statement
- DMSUPDATEDISKHEADER statement
- DMSWAIT function

- DMTRANSLOCK statement
- ENVIRONMENT declaration
- MEMORYDUMP statement
- PURGEASM statement
- READLOCKNOPURGE statement
- TPS Transaction Record Control Item Assignment statement

**Note:** *For safety precautions, the use of all DMALGOL constructs, including those not designated as unsafe, should be restricted to authorized personnel who must use DMALGOL to compile programs.*

## Using the Compile-Time Facility

The compile-time facility is used to conditionally compile DMALGOL source data. The DMALGOL compile-time facility consists of elements of a general nature that have been documented in ALGOL, and elements specific to DMSII.

The basic technique used to write DMSUPPORT libraries and other tailored software is conditional compilation. Various pieces of code are omitted, included, and assigned parameters based on information from the DASDL description file. This file is read by the DMALGOL compiler and the DMALGOL language contains elements used to conveniently reference its information.

NODE variables and PROPERTY definitions are used to refer to nodes within the description file. A node is a data structure that consists of three parts: a list, a set of properties, and a block part. Either the list or properties can be absent. The elements of lists are frequently other nodes, but they can also be integers or other data items. The syntax in DMALGOL to reference list element I of node N is N[I]. The properties of a node are contained in a nonhomogeneous substructure whose format is defined by PROPERTY declarations.

## Examples

The following statement extracts the property RECORDSZ of the node N. The properties actually used in the Access routines are included from the file called DATABASE/PROPERTIES.

```
NODE N;  
INTEGER T;  
PROPERTY RECORDSZ=[11].[11:12];  
T:=N.RECORDSZ;
```

The following example shows a special form of the 'FOR statement that is used to access the members of a node's list. The compiler obtains the number of items in the list from DASDL, which keeps a record of the number of list items at OFFSET=0.

```
NODE STRUCTURE, SPANSET;  
'FOR EACH SPANSET OF STRUCTURE DO . . .
```

This statement goes through each member of the list belonging to the node STRUCTURE and assigns its successive list elements to the node SPANSET.

The next example shows a special form of the 'INCLUDE statement that is used to access the text section of the DASDL description file. This section consists of a set of source language DMALGOL constructs and the text is always referred to by an appropriate property. For example, data set nodes contain a property called VERIFYSTORETEXT that checks whether a record meets all verification rules before it is stored. To access this text for node DS, the following statement is written:

```
'INCLUDE DS.VERIFYSTORETEXT.
```

This statement causes DMALGOL to extract the appropriate text and compile it.

For more information on compile-time facility constructs, refer to "Compile-Time Facility" in Section 2.

## Environments

DMSII builds the Database Stack (DBS), which is a true running stack and resembles a shared-by-all library. The stack is built up through the Master Control Program (MCP) procedure DMSOPEN. One task of DMALGOL is to build the stack image. This is done by using the ENVIRONMENT declaration, which delimits the boundaries of each environment in much the same way as a PROCEDURE declaration does. Environments cannot be nested more than three deep.

When the ENVIRONMENT declaration is used in a program, the code is generated in a special way and is usable only for the Accessroutines of DMSII. As the program is executed, the outermost environment begins execution, sets up its stack, executes its outer block, then calls the last compiled environment at the next level. Each inner environment in turn builds its portion of the stack, executes its outer block, then calls the next environment.

The Accessroutines code file is an executable program that builds the DBS then freezes in a special way using the DMSFREEZE construct of DMALGOL. After the stack is built up, the code executes its outer block and calls the three last compiled environments at the next level. The first is a general D[3] environment (lexical level 3) used to attach pre-Mark 3.5 Structure Information Block (SIB) environments for data sets. The second D[3] environment is used to attach pre-Mark 3.5 SIB environments for sets. The third environment is a DMSFREEZE environment. During OPEN, a series of D[3] environments are added and tailored for each structure opened. After the last structure for the first user is opened, the DMSFREEZE environment is called again. When another user opens different structures, new D[3] environments are added on a demand basis as structures are opened.

DMSII uses environments at two levels. The outermost environment contains the outer block declarations used to control and synchronize operations affecting the entire database. Also present at D[2] (lexical level 2) are the DMSUPPORT library entry point and structures used to tailor the database stack for a particular DASDL. There are three inner environments declared (mentioned in the preceding paragraph). The environment of each invoked structure is built at run time by the MCP interfacing with the DMSFREEZE environment.

The following construct allows the invocation of a procedure declared in a different environment:

<procedure identifier> <structure number>

The <structure number> construct specifies the number in the ENVIRONMENT declaration that contains the desired procedure.

### Example

```
ENVIRONMENT X OF 2;  
BEGIN  
    PROCEDURE P;  
END;  
  
ENVIRONMENT X OF 3;  
BEGIN  
    PROCEDURE P;  
    P;  
    P'2  
END;
```

In this example, when P is invoked within environment X of 3, the procedure executed is the procedure P declared within that local environment. When P'2 is invoked within environment X of 3, the procedure executed is the procedure P declared within the environment X of 2.

## Reference Variables

DMALGOL provides the ability to declare array reference variables, procedure reference variables, file reference variables, and direct file reference variables. These reference variables allow for dynamic selection of procedures, files, and direct files within the DMSII system.

In addition, DMALGOL allows up-level assignments to PROCEDURE reference array elements and FILE references. It allows up-level ARRAY reference assignments only in the Accessroutines. An assignment to a reference variable is considered up-level if the reference variable is declared in a more global block than the array, procedure, or file to which it refers. (The reference variable thus could remain dangling if the block containing the referent has been exited.)

***Note:** Although permitted in DMALGOL, up-level assignments are considered unsafe. Up-level assignments are prohibited in ALGOL, BDMSALGOL, and DCALGOL.*

## ONCE-ONLY Compilation of Procedures

ONCE-ONLY compilation is used to save compilation time and code space. If one procedure does not differ in any major respect in two environments, this procedure can be compiled only once.

It is not possible to make calls on the shared procedure using the <procedure identifier> <structure number> syntax described under “Environments,” because that would change the environment as well. Rather, a Program Control Word (PCW) is constructed for each environment; for a shared procedure, the PCW in each environment points to the same segment descriptor. The PCW allows the procedure to be shared by different environments, but behave as if it were local to each environment.

The mechanism for constructing a PCW is described under “ONCE-ONLY Compilation” in Section 2.

## Section 2

# DMALGOL Extensions to ALGOL Declarations and Statements

This section describes the extensions to various ALGOL constructs that are available in DMALGOL. For more information on the constructs, see the *ALGOL Programming Reference Manual, Vol. 1: Basic Implementation*.

## Array Declaration

DMALGOL supports all ALGOL array declarations. As an extension of the ALGOL construct, DMALGOL provides the user with a SAVE option. This option, used only in the Accessroutines, causes the array be placed in SAVE memory to ensure it a permanent location for the duration of the Accessroutines execution.

**<array declaration>**



### Explanation

For a detailed description of the array declaration construct, see the *ALGOL Programming Reference Manual, Vol. 1: Basic Implementation*.

**Note:** The SAVE form of the array declaration is considered unsafe. Use of this construct in new code is not recommended.

### Example

```
SAVE ARRAY A[0:100];
```

In this example, the one-dimensional array A is placed in SAVE memory.



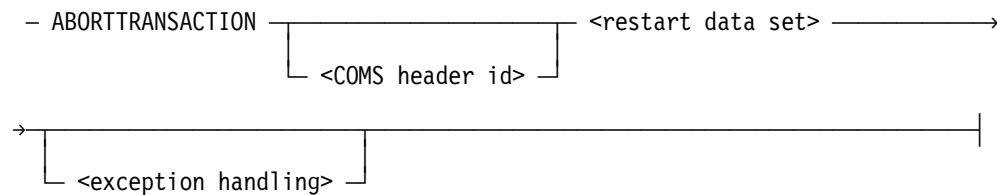
# Concurrency Control Facility

DMALGOL supports DMSII concurrency control by implementing six host language statements, which are described as follows.

## ABORTTRANSACTION Statement

The ABORTTRANSACTION statement backs out all updates that occurred during the transaction and removes the program from transaction state.

**<ABORTTRANSACTION statement>**



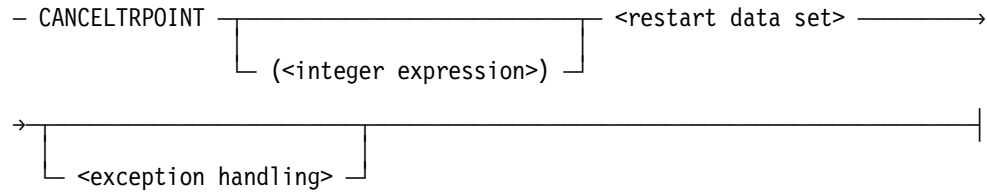
### Examples

```
ABORTTRANSACTION VENDOR_RESTART;  
ABORTTRANSACTION OUTPUT_HEADER MY_RESTART;
```

## CANCELTRPOINT Statement

The CANCELTRPOINT statement is used to back out all updates in a transaction to an intermediate save point or to the beginning of the transaction.

**<CANCELTRPOINT statement>**



### Explanation

If there is an associated arithmetic expression that is nonzero, the save point identified by the arithmetic expression is used. Otherwise, all updates that occurred during the transaction are backed out. In either case, the program is left in transaction state.

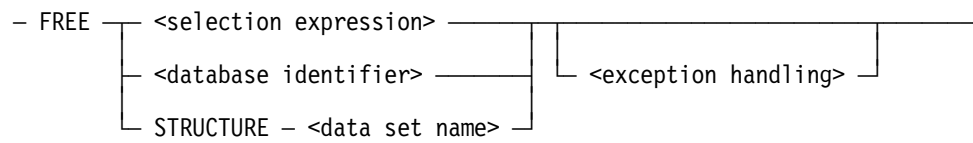
### Example

```
CANCELTRPOINT (MAIN_SAVE_POINT) MY_RESTART;
```

### FREE Statement

The FREE statement is used to release locked or secured records.

**<FREE statement>**



#### Explanation

The FREE STRUCTURE statement frees the records in a structure that was explicitly locked by a *LOCK STRUCTURE data set* statement or explicitly secured by a *SECURE STRUCTURE data set* statement.

An explicitly locked or secured structure is not freed at ENDTRANSACTION.

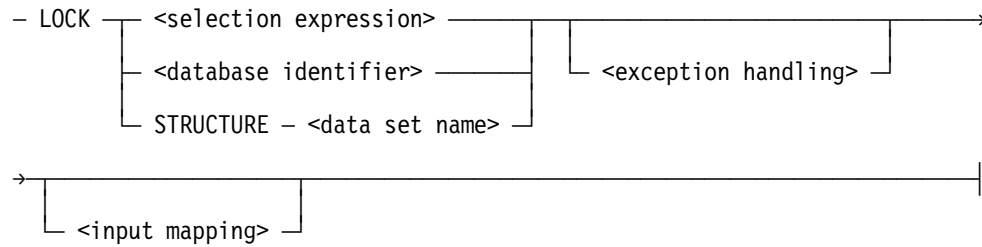
#### Example

```
FREE STRUCTURE VENDOR_DATA;
```

## LOCK Statement

The LOCK statement is used to lock records for exclusive use by one program (exclusive lock). No other program can access the locked records. A user can lock individual records or all the records in a structure.

### <LOCK statement>



### Explanation

A structure lock can be explicit or implicit.

An explicit lock occurs when the *LOCK STRUCTURE data set* Syntax is used, which locks all the records in the structure. The records are freed by using the *FREE STRUCTURE data set* statement described earlier in this section.

An implicit lock occurs when a user locks more than 50 records in any structure. The records are freed at ENDTRANSACTION.

If a user attempts to lock a structure in which other users have locked or secured records, the structure will not be locked until the users free the records or reach ENDTRANSACTION. If the users attempt to lock or secure additional records while the structure is being locked, a deadlock occurs.

### Example

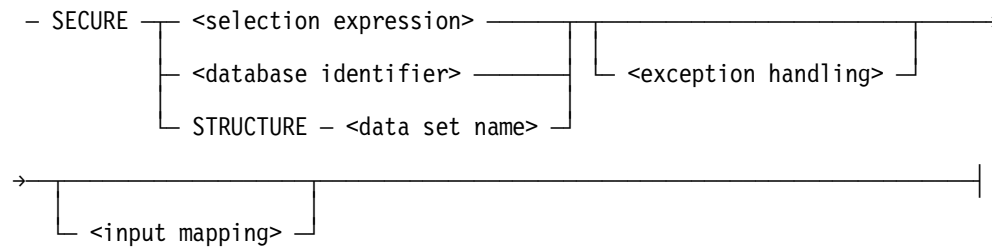
```
LOCK STRUCTURE VENDOR_DATA;
```



## SECURE Statement

The SECURE statement enables multiple programs to share the same locked records for read only access (shared lock). Individual records or all the records in a structure may be secured.

### <SECURE statement>



### Explanation

A structure secure can be explicit or implicit.

An explicit secure occurs when the *SECURE STRUCTURE data set* Syntax is used, which secures all the records in the structure. The records are freed by using the *FREE STRUCTURE data set* statement described earlier in this section.

An implicit secure occurs when you secure more than 50 records in any structure. The records are freed at ENDTRANSACTION.

If you attempt to secure a structure in which other users have locked or secured records, the structure will not be secured until the other users free the records or reach ENDTRANSACTION. If the users attempt to lock or secure additional records while the structure is being secured, a deadlock occurs.

You can lock records that are secured by issuing a LOCK statement. However, if two users try to lock the secured records, a deadlock occurs.

# Compile-Time Facility

DMALGOL provides the user with several extensions to the ALGOL compile-time facility. For information on the ALGOL compile-time facility, see the section on the compile-time facility in the *ALGOL Programming Reference Manual, Vol. 1: Basic Implementation*.

The compile-time facility is offered as a normal feature in DMALGOL. It is not necessary to set the CTPROC compiler control option, as must be done to use the compile-time facility in ALGOL.

## 'DEFINE Statement

The 'DEFINE statement can be used in DMALGOL to declare an identifier to be a preprocessor define identifier.

**<compile-time 'DEFINE statement>**

– 'DEFINE – <identifier> – = – <compile-time statement> —————|

### Explanation

The compile-time 'DEFINE statement is processed when referenced by the identifier in a subsequent 'INVOKE statement.

### Example

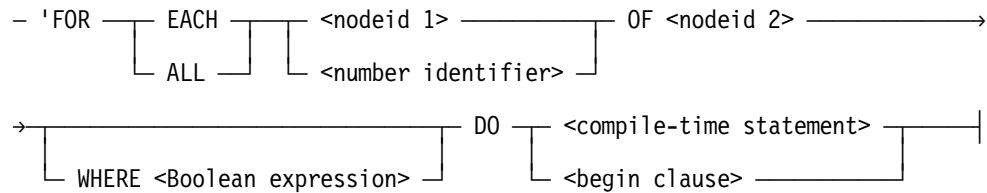
```
BEGIN
    INTEGER I;

    'DEFINE A = 'BEGIN I:=5; 'END;
    .
    .
    .
    'IF . . . THEN
        'INVOKE A;
```

## 'FOR Statement

The 'FOR statement extension provides for iterative compilation of ALGOL source code.

### <compile-time 'FOR statement>



### <begin clause>

– 'BEGIN – <text 1> – 'NEXT – <text 2> – 'PRIOR – <text3> – 'END

### Explanation

One iteration is made over the source code for each entry in the list belonging to the <nodeid 2> construct. The <nodeid 2> construct must be the index of a node in the Data and Structure Definition Language (DASDL) array.

For each iteration, the control variable is assigned the index of the list element if the <nodeid 1> construct is specified or the contents of the list element if the number identifier construct is specified.

If the WHERE clause is used, a constant Boolean expression must be specified. Before the statement following DO is processed but after the control variable has assumed its next value, the Boolean expression is evaluated. If the expression is TRUE, the statement following DO is processed; otherwise, the statement is ignored.

The compile-time statement can include any of the DMALGOL extensions to the ALGOL compile-time facility.

If the body of the 'FOR statement is the 'BEGIN-'NEXT-'PRIOR-'END form, a “telescoping” form of iteration is performed. In that case, the 'NEXT behaves like an 'END, with the following results:

1. The <text 1> construct is processed for each iteration.
2. After the last iteration, the <text 2> construct is processed just once.
3. Finally, the iteration is repeated, backwards, for the <text 3> construct.

If a WHERE clause is used, the backwards iteration processes exactly the same cases as the first iteration; that is, the Boolean expression is not reevaluated while backing out of the “telescope.”



### 'INCLUDE Statement

The 'INCLUDE statement causes the compiler to process text directly from the DASDL array.

**<compile-time 'INCLUDE statement>**

– 'INCLUDE – <nodeid> – . – <property id> \_\_\_\_\_|

#### **Explanation**

The value of the specified property is assumed to be an index of text in the DASDL array. That text must be terminated by a pound sign (#) and at least one null character (4" 00").

The compiler does not expand defines in database declarations where the define identifier was the result of a compile-time 'INCLUDE statement.

### 'INVOKE Statement

The 'INVOKE statement causes the compile-time statement previously associated with the compile-time define identifier in a 'DEFINE statement to be processed.

**<compile-time 'INVOKE statement>**

– 'INVOKE – <preprocessor define identifier> \_\_\_\_\_|

#### **Explanation**

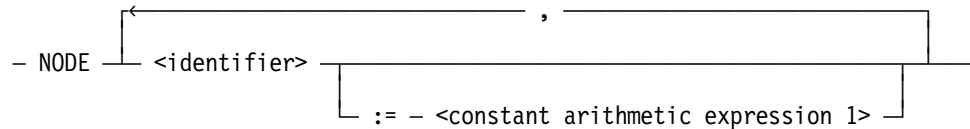
The original line numbers of the 'DEFINE declaration are preserved.

For an example of how the 'INVOKE statement is used, see the example provided for the 'DEFINE statement.

## NODE Declaration

The NODE declaration is used to declare a node identifier. It is similar to the ALGOL compile-time variable declaration NUMBER. An identifier declared to be a node is a compile-time variable that must be used in conjunction with a compile-time array created by the Data and Structure Definition Language (DASDL). The node variable represents an index into the DASDL array.

### <compile-time variable declaration>



### Explanation

The value of the node variable can be changed at any time during compilation by means of a compile-time 'LET statement.

The <constant arithmetic expression 1> construct represents the initial index of the node variable. By default, the initial value is zero, which is otherwise an illegal value; thus, a node must be initialized or assigned a value before it is used.

Normally, the node variable represents the index of a node in the DASDL array. A list and a set of properties are associated with a node. The list is usually a list of nodes, and the properties contain values. The node variable can be used to reference members of the list and values of the properties. For example, if N is a node variable and P is a property identifier, then

- N[<i>] is the <i>th member of the list of N
- N.P is the value of the property P of N

where <i> is a constant or constant expression.

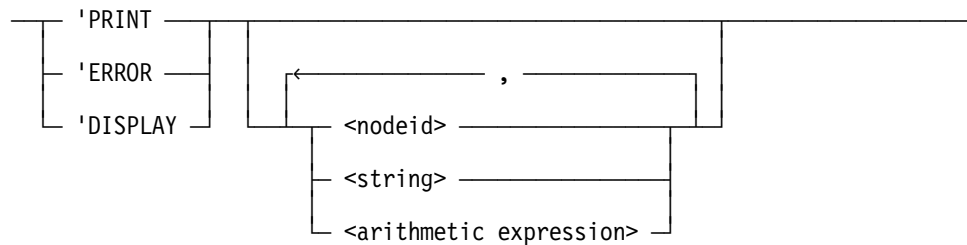
### Example

```
NODE NODULE := 6;
```

This example declares a node identifier called "NODULE" with an initial index value of 6.

### 'PRINT, 'ERROR, 'DISPLAY Statements

These statements cause one or more lines to be printed on the compilation listing.



#### Explanation

Each line can contain up to 87 characters, starting in position 18.

The lines are created from the information given in the statement, as follows:

- If the `<nodeid>` construct is specified, the property “name” (an alpha property) for the node is inserted into the line.
- If the `<string>` construct is specified, it is inserted into the line.
- If the `<arithmetic expression>` construct is used, it must be a constant arithmetic expression. It is assumed to be an integer, and the value of the integer, zero suppressed, is inserted into the string.

No blanks are inserted into the line between specified items.

If a 'PRINT or 'ERROR statement is processed (not skipped) by the compile-time processor, the lines are printed whether or not any listing dollar card options (LIST, LISTOMIT, and so on) are initialized.

If the 'ERROR form is used, the compiler's error count is incremented by one, and the printed line is enclosed in angle brackets (`<>`) as with a typical syntax error message.

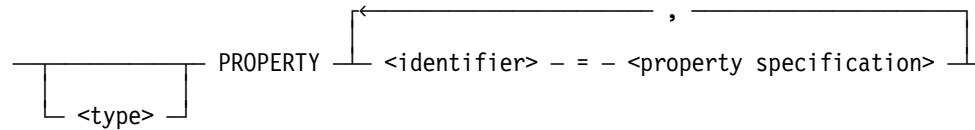
#### Examples

```
'PRINT "BEGINNING OUTER BLOCK CODE"
'ERROR "NO HANDLING FOR THIS CASE HAS BEEN IMPLEMENTED"
'DISPLAY I;
```

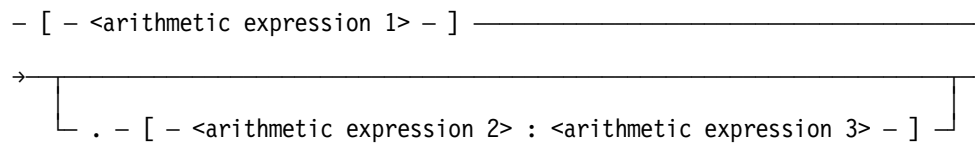
## PROPERTY Declaration

The PROPERTY declaration declares a property identifier that defines the location and format of a property value associated with a node in a DASDL array.

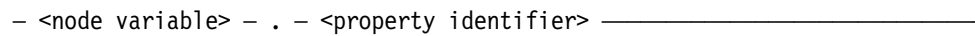
### <compile-time PROPERTY identifier declaration>



### <property specification>



### <node property construct>



### Explanation

Each arithmetic expression must be a constant arithmetic expression. Arithmetic expression 1 specifies the word in the property set. Arithmetic expressions 2 and 3 specify a field within the word; if they are not given, the entire word is used.

If a type is declared, it must be single-precision arithmetic or Boolean. When no type is specified, REAL is assumed.

A property identifier is used only with a node variable. This node property construct represents the value of the property for the given node. It can be used wherever constants of the specified type can appear.

The node property construct is used only to retrieve a value from the DASDL array. It cannot be used to change a property value. (In fact, no construct can change the DASDL array, as it is read-only.)

### DMIO File Attribute

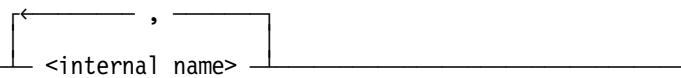
The DMIO direct file attribute is used by the Accessroutines to indicate that the Data Management System II (DMSII) system is using the file. When the DMIO attribute is specified, certain MCP checks are turned off to avoid problems when multiple stacks are sharing a file.

*Note:* The DMIO attribute is considered unsafe.

### DYNAMIC DATABASE Declaration

The DYNAMIC DATABASE declaration is a DMALGOL extension to the DATABASE declaration. It enables you to declare databases that can be invoked at run time.

**<DYNAMIC DATABASE declaration>**

– DYNAMIC – DATABASE     **<internal name>**

#### Explanation

“Dynamic” specifies that the description of the database is to be provided by the application at run time rather than at compile time.

Dynamic databases are a type of Data Management System II (DMSII) database. They are not bindable.

Dynamic databases **cannot** be declared within an ALGOL structure block.

*Note:* DYNAMIC DATABASE declarations are considered unsafe.

#### Examples

```
DYNAMIC DATABASE D81;  
DYNAMIC DATABASE D81, D82;
```

## EXCEPTION PROCEDURE Declaration

DMALGOL supports the ALGOL EXCEPTION PROCEDURE declaration. As an extension of the ALGOL construct, DMALGOL enables the EXCEPTION PROCEDURE to be declared as PROTECTED.

### <EXCEPTION PROCEDURE declaration>

```

┌──────────────────┐ EXCEPTION PROCEDURE – <exception procedure identifier> →
│                   │
└── PROTECTED ───┘

```

```
→ <unlabeled statement> ───────────────────────────────────────────────────┘
```

### <exception procedure identifier>

```
– <identifier> ───────────────────────────────────────────────────────────┘
```

### Explanation

An exception procedure is subject to interruptions such as those caused by software interrupts, stack stretching, and the DS and STOP system commands. A protected exception procedure is not subject to such interruptions; however, the protected status of the procedure does not apply when the procedure is called directly.

If a protected exception procedure is declared, the code file is marked as unsafe and nonexecutable. The code file can be opened only with the SL or MP system command. For additional information on the SL and MP commands, see the *System Commands Operations Reference Manual*.

### Nonlocal Message References

In DCALGOL, references to a MESSAGE variable must be local; that is, the message must be declared in the block or procedure in which the reference occurs. DMALGOL extensions allow references to be made to globally declared MESSAGEs; however, such references are considered unsafe. For more information on message references, see the section on message and message array declarations in the *DCALGOL Programming Reference Manual*.

#### Example

```
BEGIN
  MESSAGE M1;
PROCEDURE P;
  BEGIN
    MESSAGE M2;
    IF ... THEN
      DCWRITE(M1)
    ELSE
      DWRITE(M2);
  END
END;
```

In this example, procedure P contains references to a locally declared message, M2, and to a globally declared message, M1.

**Note:** *References to globally declared messages are considered unsafe.*

## ONCE-ONLY Compilation

ONCE-ONLY compilation is used to save compilation time and code space. It is established using the compile-time 'LET statement and a special form of the PROCEDURE declaration.

### <compile-time 'LET statement>

– 'LET – <number variable> – := – <procedure identifier> —————|

### <procedure declaration>

– <procedure type> – PROCEDURE – <procedure heading> – ; – EXTERNAL —————>

→– <number variable> —————|

### Explanation

ONCE-ONLY compilation is restricted to use by programs having an ENVIRONMENT declaration (that is, Accessroutines). It is used when a procedure declared in one environment is common to one or more other environments. Normally, ALGOL scope rules require that each instance of the procedure be compiled; thus, each instance receives its own code segment. With the DMALGOL extensions, a procedure need only be compiled once, in the first environment. Subsequent environments refer back to that declaration.

When the ONCE-ONLY compilation feature is used, the same code segment (or procedure) is executed in different addressing environments. Therefore, the procedure declaration must be identical except for the lack of a procedure body in the external procedure. DMALGOL does not check the procedure declarations for consistency; it simply copies the program control word (PCW) for the original procedure into the environment containing the EXTERNAL PROCEDURE declaration.

The procedures must be declared at the same lexical level in equivalent environments. If the procedure body refers to global variables, those variables must be declared in each environment so that they are allocated at the same stack addresses. The variables must also be of the same type.

**Note:** *This form of PROCEDURE declaration is considered to be extremely unsafe.*



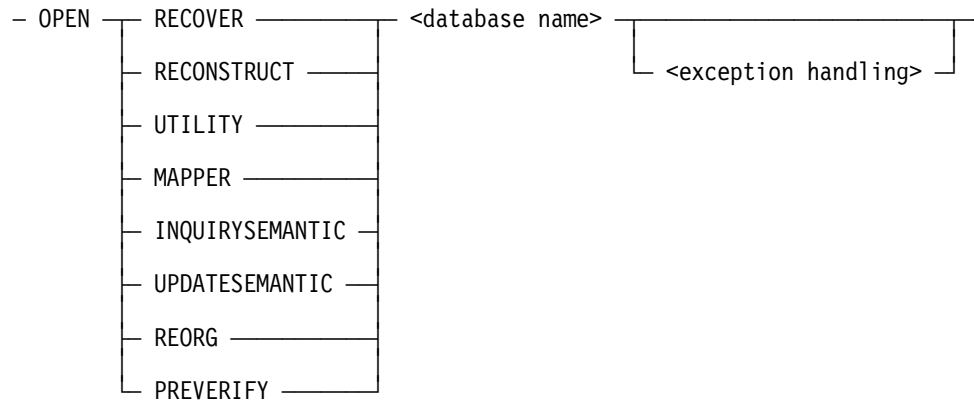
### Example

```
NUMBER GETADDRESSPCW;  
BOOLEAN PROCEDURE GETADDRESS (A);  
  VALUE A;  
  REAL A;  
  BEGIN  
    .  
    .  
    .  
  END;  
'LET GETADDRESSPCW := GETADDRESS;  
  
BOOLEAN PROCEDURE GETADDR (A);  
  VALUE A;  
  REAL A;  
  EXTERNAL GETADDRESSPCW;
```

## OPEN Statement

DMALGOL contains six options for the OPEN statement that are related to opening a database. These options are used to indicate special circumstances to the Access routines.

### <OPEN statement>



### Explanation

The exception handling construct is used to denote those statements where a program variable can be designated to receive the value of the database status word. For more information on exception handling, see the section on exception processing in the *ALGOL Programming Reference Manual, Vol. 1: Basic Implementation*.

### Example

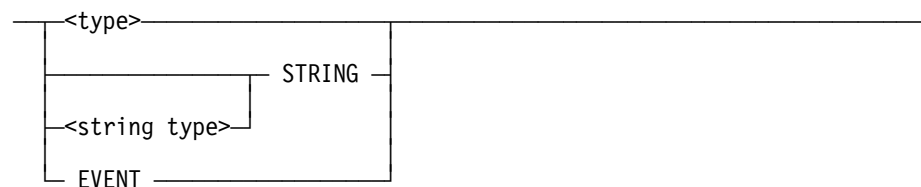
```
OPEN RECOVER DB1:B;
```

**Note:** The *MAPPER*, *INQUIRYSEMANTIC*, and *UPDATESEMANTIC* options are considered unsafe.

### Procedure Declaration

DMALGOL supports all ALGOL procedure declarations. As an extension of the ALGOL construct, DMALGOL provides the user with an EVENT procedure type.

**<procedure type>**



#### Explanation

An EVENT-valued procedure returns a reference to an event as its procedure value. This reference takes the form of an SIRW to a single EVENT or an indexed descriptor to an element of an EVENT ARRAY. The event reference that is returned can be used anywhere a regular event designator is used.

For further information regarding procedure declarations, refer to the *ALGOL Programming Reference Manual, Vol.1: Basic Implementation* manual.

## Reference Variables Declaration and Assignment

DMALGOL provides the ability to declare array reference variables, procedure reference variables, file reference variables, and direct file reference variables. These reference variables allow for dynamic selection of arrays, procedures, files, and direct files within the Data Management System II (DMSII) system. In some cases, DMALGOL permits reference assignments to be up-level; that is, the reference variable can refer to another entity at a higher lexical level.

### Array References

An array reference assignment associates a variable, called an array reference, with an <array designator>, which in ALGOL can designate only an array or subarray. DMALGOL allows several additional types of array designations to be assigned to the array reference identifier. In addition, DMALGOL allows for up-level array reference assignments in some cases.

#### <array reference assignment>

– <array reference variable> – :=	<table border="0" style="width: 100%;"> <tr><td style="border-bottom: 1px solid black; width: 80%;">&lt;array designator&gt;</td><td style="border-bottom: 1px solid black; width: 20%; text-align: right;"> </td></tr> <tr><td style="border-bottom: 1px solid black;">&lt;database id&gt;.DMSIB1</td><td style="border-bottom: 1px solid black; text-align: right;"> </td></tr> <tr><td style="border-bottom: 1px solid black;">&lt;database id&gt;.DMSIB2</td><td style="border-bottom: 1px solid black; text-align: right;"> </td></tr> <tr><td style="border-bottom: 1px solid black;">&lt;database id&gt;.DMSIBDESC</td><td style="border-bottom: 1px solid black; text-align: right;"> </td></tr> <tr><td style="border-bottom: 1px solid black;">DMKEYAREA</td><td style="border-bottom: 1px solid black; text-align: right;"> </td></tr> <tr><td style="border-bottom: 1px solid black;">DMWORKAREA[&lt;structure environment index&gt;]</td><td style="border-bottom: 1px solid black; text-align: right;"> </td></tr> <tr><td style="border-bottom: 1px solid black;">DMSIBDESC</td><td style="border-bottom: 1px solid black; text-align: right;"> </td></tr> <tr><td style="border-bottom: 1px solid black;">&lt;transaction variable id&gt;</td><td style="border-bottom: 1px solid black; text-align: right;"> </td></tr> </table>	<array designator>		<database id>.DMSIB1		<database id>.DMSIB2		<database id>.DMSIBDESC		DMKEYAREA		DMWORKAREA[<structure environment index>]		DMSIBDESC		<transaction variable id>	
<array designator>																	
<database id>.DMSIB1																	
<database id>.DMSIB2																	
<database id>.DMSIBDESC																	
DMKEYAREA																	
DMWORKAREA[<structure environment index>]																	
DMSIBDESC																	
<transaction variable id>																	

#### Explanation

The <array designator> construct indicates the array or array portion to be associated with the array reference variable. In DMALGOL, the array designator can be any valid ALGOL array designator. Following an array reference assignment, the array reference variable becomes a referred array, describing the same data as the array designator, which can itself be an original array or another referred array.

DMALGOL allows an array reference assignment to an array designator to be up-level only in the Accessroutines. These up-level references are considered unsafe.

DMALGOL accepts the three <database id> arrays as valid array designators. These three arrays are constituents of the database declaration. Unless the database identifier was declared dynamic, the array reference being assigned is marked read-only; data can be retrieved but not stored through that array reference.

When there is exactly one database declared in a program, and that database is not dynamic, DMALGOL accepts the three designators DMKEYAREA, DMWORKAREA, and DMSIBDESC. For more information about these designators, see “DMINQ Arrays” in Section 3.

DMALGOL allows an array reference to be assigned to a transaction input record or transaction output record by using the transaction variable id construct as the array designator. The transaction variable id is an identifier that is the name of the transaction record variable. Subsequent use of the array reference variable references the Transaction Processing System (TPS) transaction variable. Such an assignment is considered unsafe.

Transaction record variables are described in the section describing the user language interface to TPS in the *DMSII Transaction Processing System (TPS) Programming Guide*.

### Example

```
A := DB.DMSIB1
```

In this example, the array reference A is associated with the array designator DB.DMSIB1.

**Note:** *If the array reference is up-level or is assigned to a transaction variable identifier, this construct is considered unsafe.*

## Procedure Reference Arrays

A procedure reference array declaration declares an array that allows a group of like procedures to be treated as a single entity. The DMALGOL extension to this construct permits up-level reference assignments.

For more information about the PROCEDURE REFERENCE ARRAY declaration, refer to the *ALGOL Programming Reference Manual, Vol. 1: Basic Implementation*.

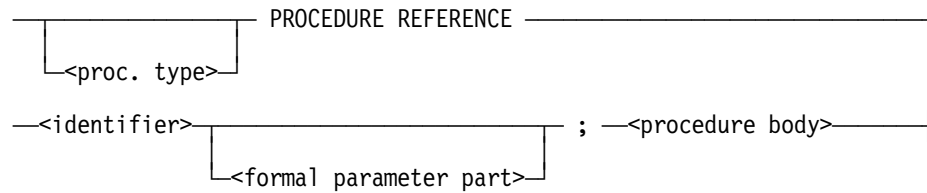
**Note:** *Up-level procedure reference array assignments are considered unsafe.*

## Procedure References

A procedure reference declaration declares a procedure reference identifier. This enables dynamic selection of procedures within DMSII. The DMALGOL extension to this construct permits up-level reference assignments.

In addition to the ALGOL syntax for declaring PROCEDURE REFERENCES, DMALGOL enables another form of syntax.

### <PROCEDURE REFERENCE Declaration>



### Explanation

This form of procedure reference declaration specifies a body of code to be executed if the procedure reference id construct is used before an assignment is made to it.

**Note:** This form of the PROCEDURE REFERENCE Declaration is considered unsafe.

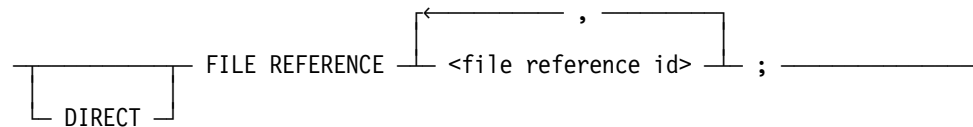
For more information regarding the PROCEDURE REFERENCE Declaration, refer to the *ALGOL Programming Reference Manual, Vol. 1: Basic Implementation*.

### File References

DMALGOL provides the ability to declare reference variables for files and direct files. This ability allows for dynamic selection of files and direct files within DMSII.

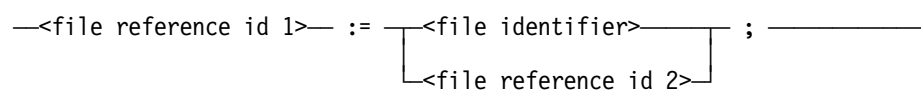
### Declaration Syntax

**<FILE REFERENCE declaration>**



### Assignment Syntax

**<FILE REFERENCE assignment>**



### Explanation

Prior to being assigned a value, a file reference variable does not point to a file.

In the file reference assignment statement, the left and right file identifiers must both be DIRECT or both be non-DIRECT.

### Example

```
FILE REFERENCE FR1;  
FR1 := RMT;
```

This example first declares a file reference FR1, then assigns an RMT to FR1.

**Note:** *If the assignment is up-level, the construct is considered unsafe.*

## Up-Level Event References

When an event designator is provided in a READ or WRITE statement for a direct file, ALGOL requires the event to be no less global than the direct array referenced in the statement. DMALGOL relaxes this restriction, but up-level READ/WRITE event designations are considered unsafe. The DMALGOL compiler checks to verify that the event (if supplied) is no less global than the direct array. There are three possible outcomes of the verification checking:

- The static relationship is correct and, thus, no run-time checking is required.
- The static relationship is incorrect. DMALGOL marks the construct as unsafe. (ALGOL would generate a syntax error in this case.)
- The relationship cannot be verified statically because the array designator is an array reference or a formal parameter, or because the event designator is a formal parameter. In this case, the MCP is directed to perform the verification check at run time and faults the up-level event references even for DMALGOL. (However, if the direct file has the DMIO attribute set, no check is made. For this reason, the DMIO attribute is considered unsafe.)

There are two points to notice about verification checking. First, there is no way to turn it off completely for non-DMIO files. If the test cannot be performed at compile time, it is performed at run time. Second, the MCP run-time checks are conservative and can at times reject a legitimate situation. For Example, if either the event or the direct array is in a library or database stack, the only acceptable case is when both are in the same stack and the event is lower in the stack than the direct array. If the event and the direct array are in the same block, the event must be declared first.





## Section 3

# DMALGOL-Specific Functions and Statements

This section describes the functions and statements that are unique to DMALGOL.

## ALLOW Statement

The ALLOW statement is used with the DISALLOW statement to mark critical sections of code in the Accessroutines where a series of operations must be performed without interruption. The ALLOW statement permits interrupt code to be executed.

**<ALLOW statement>**

– ALLOW —————|

### Explanation

The ALLOW statement appears at the end of a section of critical code to reenable external interrupts that were previously disabled by a DISALLOW statement.

## DISALLOW Statement

The DISALLOW statement is used with the ALLOW statement to mark critical sections of code in the Accessroutines where a series of operations must be performed without interruption. The DISALLOW statement prevents interrupt code from being executed.

**<DISALLOW statement>**

– DISALLOW —————|

### Explanation

The DISALLOW statement appears at the beginning of the section of critical code to disable external interrupts. The external interrupts can be reenabled at the end of the section using the ALLOW statement.

**Note:** *The DISALLOW statement is considered unsafe.*

### DMINQ Functions

The DMINQ functions consist of the DMINQ interface function and the DMINQ arrays.

#### DMINQ Interface Function

The DM INQUIRY (DMINQ) interface permits direct communication with the Data Management System II (DMSII) Accessroutines.

In a program that uses a DMINQ function, one database (no more, no less) must be invoked in the normal manner, or a syntax error is returned.

##### <DMINQ function>

– DMINQ – [ – <arithmetic expression> – ] – ( – <array row> – ) ————|

##### Explanation

The arithmetic expression specifies the index (ENVINX) within the Structure Information Block (SIB) for the desired structure. The array row specifies a one-dimensional array used to communicate with the Accessroutines. The contents of the array control the function performed by the system.

The value of the array element, A[0], identifies the specific procedure as described in Table 3–1:

**Table Section 3–1. DMINQ Array Values and Procedures**

Array [0] Value	Procedure
1	Pathfinder (find key only)
10	Set to beginning
11	Data finder (find/lock next/current)
12	GETDATA
13	DMSREAD (access data portion only)
14	Get link
16	Store current
17	Free current
18	Set or check date-timestamp for Transaction Processing System (TPS)
19	Clear TPS flag and date-timestamp
20	Return last transaction address
21	Get status of abort
22	Rerun finished -- reset TPS, abort pending
23	Create
24	Delete current
25	Get statistics information
26	Visible Database Stack (DBS) message
27	Return displayed messages
28	Return software version
29	Clear TPS address
30	Return standard data set DATAEOFs
31	GETDATA with KEYCOMPARE
32	Off-line dump
33	Set cursor
34	Set current path to a different cursor path

The procedures outlined in the previous table are described in detail under the following headings.

### Procedure: Pathfinder (A[0] =1)

This procedure finds a record in a set.

#### Parameters

A[1] = FIND:

-1 = current record

0 = next in set

1 = next in set = User Key Area (UKA)

2 = next in set > UKA

3 = next in set >= UKA

4 = link in set

5 = prior in set

6 = prior in set = UKA

7 = prior in set < UKA

8 = prior in set <= UKA

A[2] = Size (SZ) parameter to pathfinder

This parameter specifies the size of the key passed to pathfinder in the user key area. If SZ is less than zero (0), then ABS(SZ) is the number of hex characters in the user key; otherwise, SZ is the number of bytes in the user key.

A[3] = Size (SZ2) parameter to pathfinder

This parameter specifies the size of the major portion of the user key which must exactly match the retrieved key. If SZ is less than zero (0), then ABS(SZ2) is the number of hex characters in the user key; otherwise, SZ2 is the number of bytes in the user key.

As an example, if an index set had a concatenated key (A,B,C) with each portion two bytes long, to retrieve the next key where A=UKA and B=UKA requires SZ=4 and SZ2=4. To retrieve the next key where A=UKA and B>UKA requires SZ=4 and SZ2=2.

If a record is found as specified, the AA word is returned in A[1]. A GETDATA call must be used to move the record to the user's work area.

### **Procedure: Set to Beginning (A[0] = 10)**

This procedure sets the current record parameter to the beginning or the ending of the record.

#### **Parameters**

A[1] = FINDTYPE:

0 = set to beginning

1 = set to ending

A[i], i>1, not used

### **Procedure: Data Finder (A[0] = 11)**

This procedure finds a record in a dataset. If a record is found, its AA word is returned in A[1], and the record is moved to the user's work area.

#### **Parameters**

A[1] = FIND:

0 = find current data set

1 = lock current data set

2 = find next data set

3 = lock next data set

4 = find prior data set

5 = lock prior data set

### **Procedure: GETDATA (A[0] = 12)**

This procedure moves the desired record to the user's work area.

#### **Parameters**

A[1] = AA word of desired record

### Procedure: DMSREAD (A[0] = 13)

This procedure is similar to the GETDATA procedure, except that only the data portion of the desired record is moved to the user's work area. Structures embedded in the accessed record are unaffected.

#### Parameters

A[1] = AA word of desired record

### Procedure: Get Link (A[0] = 14)

This procedure gets the link address and performs a fetchkey call on the control manager. The link entry is returned in the array A[\*].

#### Parameters

A[1] = Unused

A[2] = LLOC parameter

A[3] = LLEN parameter

### Procedure: Store Current (A[0] = 16)

This procedure stores the current record.

#### Parameters

A[i], i>0, not used

### Procedure: Free Current (A[0] = 17)

This procedure frees the current record.

#### Parameters

A[i], i>0, not used

### Procedure: Set or Check Date-Timestamp for TPS (A[0] = 18)

This procedure is used to set or check the value of the timestamp (TIME(6)) given to the Accessroutines when the Transaction Processing System (TPS) journal library is initiated. A[1] contains the timestamp to be given to the Accessroutines.

If the existing timestamp in the Accessroutines is not zero and a call on this DMINQ function is made (passing a new timestamp), this DMINQ function returns a value of TRUE and the new timestamp is not captured by the Accessroutines.

#### Parameters

None

### Procedure: Clear TPS Flag and Date-Timestamp (A[0] = 19)

This procedure signals the Accessroutines to set the TPSCLOSEDFLAG true and reset the timestamp, which was given to the Accessroutines upon initiation of a TPS Journal library, to zero.

#### Parameters

None

### Procedure: Return Last Transaction Address (A[0] = 20)

This procedure returns the last transaction address in the database control file.

#### Parameters

The transaction address in the Database Control File is returned in words A[0], A[1] and A[2], as follows:

A[0] is the file number of the address.

A[1] is the block number of the address.

A[2] is the offset number of the address.



### Procedure: Get Status of Abort (A[0] = 21)

This procedure returns a value to TPS (in A[0]) that indicates the status of an abort. The possible values returned are as follows:

- 1 Accessroutines are waiting for updaters to leave transaction state.
- 2 All updaters are gone. Accessroutines are waiting for an ABORT.
- 3 The abort is finished. Accessroutines are waiting for TPS to reprocess transactions.
- 4 No abort or reprocessing of transactions is necessary.

#### Parameters

None

### Procedure: Rerun Finished -- Reset TPS Abort Pending (A[0] = 22)

This procedure tells the Accessroutines that TPS has finished reprocessing transactions. The Accessroutines resets the TPSABORTPENDING flag to FALSE.

#### Parameters

None

### Procedure: Create (A[0] = 23)

This procedure creates a record.

#### Parameters

A[1] = CREATETYPE:

0 = create

1 = recreate

A[2] = Record type, if format is variable; else unused

### Procedure: Delete Current (A[0] = 24)

This procedure deletes a record.

#### Parameters

A[i], i>0, not used

## Procedure: Get Statistics Information (A[0] = 25)

This procedure gets the database statistics. The parameters are described in Table 3–2.

### Parameters

**Table Section 3–2. Word Values and Descriptions for the Get Statistics Information Procedure**

Word	Contents	Description
Word 0	25	Specifies a statistics request to the DMINQ interface
Word 1	Structure number	Indicates the structure number of the desired statistics (or equals 0 if global statistics are desired).
Word 2	Type of statistics	Specifies whether static statistics (value of 0) or dynamic statistics (value of 1) are desired.
Word 3 (See note at end of table.)	Statistics result word	Indicates the result of the statistics request. If the request was correctly formatted and honored, this word is 0 (zero). Otherwise, [0:1] is equal to 1 and [35:8] contains an error category. The currently defined error categories are as follows:  <b>Error Number</b>  <ol style="list-style-type: none"> <li>1. The structure number provided in word 1 did not correspond to an existing data set or set in the database.</li> <li>2. The statistics request type provided in word 2 was invalid (not 0 or 1).</li> <li>3. A fault was encountered while the program was attempting to retrieve statistics.</li> <li>4. A dynamic statistics request was made for an inactive structure.</li> <li>5. A function is not available for this structure.</li> </ol>
Word 4	Total number of words returned	Contains the total number of words returned in the array, including all of the fixed words at the front of the array. Note that if the array provided is too small to receive all of the statistics, it is resized.
Word 5	Index to header word for subgroup 1	Contains the index of the first group of statistics information returned.

continued

**Table 3–2. Word Values and Descriptions for the Get Statistics Information Procedure (cont.)**

Word	Contents	Description								
Word[Word 5]	Header word for subgroup 1	<p>Precedes each group and indicates the Group type and the number of words of information in the group.</p> <p>The layout of this header word is as follows:</p> <table><tr><th>Field</th><th>Contents</th></tr><tr><td>[47:24]</td><td>Not used</td></tr><tr><td>[23:8]</td><td>Group type</td></tr><tr><td>[15:16]</td><td>Number of words in group (including header)</td></tr></table>	Field	Contents	[47:24]	Not used	[23:8]	Group type	[15:16]	Number of words in group (including header)
Field	Contents									
[47:24]	Not used									
[23:8]	Group type									
[15:16]	Number of words in group (including header)									
Word[Word 5 + ]	First data word for subgroup 1									
Word[Word 5 + n]	Header word for subgroup 2									
Word[Word 4 - 1]	End-of-statistics flag	Follows the last group of statistics and has a Group type of zero (0).								

**Note:** The Accessroutines process the request and return the result in the array starting at Word 3. Words 0 through 2 are unaffected by the Accessroutines. The format of the result is stated in the description column.

The various group types and their layouts are shown in the following tables. Ticks refer to time on the processor clock. For example, one tick equals 2.4 microseconds. The group types can appear in any order.

**Table Section 3–3. Global Static Statistics [word 1 = 0, word 2 = 0]**

Word	Contents
<b>Group type 6</b>	
1	Time database opened (TIME(7) value)
2	Maximum valid structure number in database
3	Database options: [0:1] = 1 if STATISTICS is initialized in database [1:1] = 1 if database is audited [2:1] = 1 if LOCKSTATISTICS is initialized in database
4	Compile-time \$options: [10:01] = 1 if database is partitioned [13:01] = 1 if B7700 optimize is initialized [17:01] = 1 if database STATISTICS is initialized [22:01] = 1 if database is audited
5	Database DASDL options: [4:1] = 1 if POPULATIONWARN is enabled for any structure [5:1] = 1 if POPULATIONINCR is enabled for any structure [8:1] = 1 if INDEPENDENTTRANS is initialized [9:1] = 1 if REAPPLYCOMPLETED is initialized [10:01] = 1 if database is partitioned [12:01] = 1 if TPSDUALUPDATE is initialized [13:01] = 1 if B7700 optimization is initialized [15:01] = 1 if CHECKSUM is initialized [16:01] = 1 if REBLOCK is initialized [17:01] = 1 if database STATISTICS is initialized [18:01] = 1 if KEYCOMPARE is initialized [19:01] = 1 if RDSSTORE is initialized [20:01] = 1 if ALLFAULTRESTART is initialized [21:01] = 1 if BACKOUTTOSPT is initialized [22:01] = 1 if database is audited [23:01] = 1 if ADDRESSCHECK is initialized [23:24] = 1 if all DASDL options are initialized

continued

**Table 3–3. Global Static Statistics [word 1 = 0, word 2 = 0] (cont.)**

Word	Contents
<b>Group type 5</b>	
1-n	Database name including usercode prefix, if any. Database name is followed by 4"00" .

**Table Section 3–4. Global Dynamic Statistics (word 1 = 0, word 2 = 1)**

Word	Contents
<b>Group type 1</b>	
1	Current database open count
2	Current number of users that have database open for update
3	Current database open state: <ul style="list-style-type: none"> <li>0 -- database is not open</li> <li>1 -- database is opened temporary</li> <li>2 -- database is opened and initialized</li> <li>3 -- database is open</li> <li>4 -- database is undergoing recovery</li> </ul>
4	Unused
5	Maximum total buffer space in words (entire database)
6	Current total buffer space in words (entire database)
7	Current ALLOWEDCORE value (entire database)
8	Current OVERLAYGOAL value multiplied by 1000 (entire database)
9	Current overlay rate (entire database)
10	Maximum number of buffers allocated (entire database)
<b>Group type 2</b>	
1	Time statistics collections started or were last changed (TIME(7) value)
2	Number of forced database overlays (system-wide)
3	Number of normal database overlays (system-wide)
4	Unused

continued

**Table Section 3–4. Global Dynamic Statistics (word 1 = 0, word 2 = 1) (cont.)**

Word	Contents
<b>Group type 3</b>	<b>(present only if STATISTICS is initialized and database is audited)</b>
1	First audit file number
2	Current audit file number
3	Starting audit block serial number
4	Current audit block serial number
5	Average number of words used in audit blocks
6	Actual audit block size
7	Number of audit input/outputs initiated
8	Total wait time accumulated on primary audit, in ticks ("Ticks" refers to ticks of the processor clock at 2.4 microseconds/tick.)
9	Total wait time accumulated on secondary audit, in ticks
10	Total transaction count
11	Total number of times processes were held up at BEGINTRANSACTION
12	Total time spent waiting at BEGINTRANSACTION, in ticks
13	Total number of syncpoints taken
14	Total number of controlpoints taken
15	Total time spent taking controlpoints, in ticks
16	Sum of the number of buffers present at each controlpoint
17	Sum of the number of buffers flushed at each controlpoint
18	Number of forced audit write operations at syncpoint time, in ticks
19	Total waiting time for forced audit write operations at syncpoint time, in ticks
20	Average waiting time for forced audit write operations at syncpoint time, in ticks
21	Number of forced audit write operations to unconstrain buffers
22	Total waiting time for forced audit write operations to unconstrain buffers, in ticks
23	Average waiting time for forced audit write operations to unconstrain buffers, in ticks
24	Number of other forced audit write operations
25	Total waiting time for other forced audit write operations, in ticks
26	Average wait time for other forced audit write operations, in ticks
27	Number of normal audit write operations

continued

**Table 3–4. Global Dynamic Statistics (word 1 = 0, word 2 = 1) (cont.)**

Word	Contents
<b>Group type 3</b>	<b>(present only if STATISTICS is initialized and database is audited)</b>
28	Total waiting time for normal audit write operations
29	Average waiting time for normal audit write operations

**Table Section 3–5. Structure Static Statistics (word 1 = <structure number>, word 2 = 0)**

Word	Contents
<b>Group type 7</b>	
1	Structure number
2-4	Structure name (first byte is length in binary)
5	Structure type: 2 = data set 5 = index set
6	Structure subtype (refer to the DATABASE/PROPERTIES symbolic file for more information)
7	Structure nesting level (1 = disjoint)
8	Structure block factor (in records for data sets, in key entries for index sets)
9	Structure physical block size in words (including integrity-checking words)
10	Structure area size in sectors
11	= 1 if structure is checksummed
12	= 1 if structure is addresschecked

Table Section 3–6. Structure Dynamic Statistics

Word	Contents
<b>Group type 8</b>	
1	Current number of random-access users
2	Current number of serial-access users
3	Current number of buffers allocated for structure
4	Current number of big buffers (for serial users) allocated for structure
5	Current number of all users (inquiry and update)
6	Current number of update-access users
<b>Group type 9 (present only if STATISTICS is SET and structure is a data set)</b>	
1	Number of FIND commands issued against data set
2	Number of CREATE/STORE commands issued against data set
3	Number of MODIFY/STORE commands issued against data set
4	Number of DELETE commands issued against data set
5	Number of times control information changed
<b>Group type 10 (present only if STATISTICS is SET and structure is an index set)</b>	
1	Number of FIND commands issued against index set
2	Number of INSERT commands used in index set
3	Number of key data changes in index set
4	Number of key deletions from index
<b>Group type 11 (present only if STATISTICS is SET)</b>	
1	Number of physical read operations against structure
2	Number of physical write operations against structure
3	Number of ticks spent waiting for read operations to complete
4	Number of ticks spent waiting for write operations to complete
5	Total amount of I/O time accumulated on file
6	Number of readahead operations issued against structure
7	Number of writeahead operations issued against structure

continued



**Table 3–6. Structure Dynamic Statistics (cont.)**

Word	Contents
<b>Group type 14</b>	<b>(present only if POPULATIONINCR is enabled and structure is a data set)</b>
1	POPULATIONINCR value
2	POPULATIONINCR timestamp [TIME (6)]
3	POPULATIONINCR current target for maximum AREAS
<b>Group type 15</b>	<b>(present only if POPULATIONWARN is enabled and structure is a data set)</b>
1	POPULATIONWARN value
2	POPULATIONWARN timestamp [TIME (6)]

### Procedure: Visible DBS Message (A[0] = 26)

Starting at A[1] is the Visible DBS message to be processed. The message must be terminated by 4"00".

The message is passed to the Visible DBS message processing routine and any response message is returned starting at A[1]. If the response contains multiple lines, 4"0D" is used to separate the multiple lines. The last line is terminated by 4"00".

#### Parameters

None

### Procedure: Return Displayed Messages (A[0] = 27)

This procedure returns the most recent displayable messages (does not include messages generated by the Visible DBS) in the array A, starting at the first character of word 0. If no messages exist, then seven nulls (48"00" ) are returned, starting at the first character of word 0. The most recent 23 messages are returned in the order most recent to least recent. Each message is terminated by 48"0D", and the final message of the group is terminated by a null (48"00").

#### Parameters

None

**Procedure: Return Software Version (A[0] = 28)**

This procedure returns the software version. Mark is in A[0].[47:16], cycle is in A[0].[31:16], and level is in A[0].[15:16].

**Parameters**

None

**Procedure: Clear TPS Address (A[0] = 29)**

This procedure resets the transaction address in the database control file to 0.

**Parameters**

None

**Procedure: Return Standard Data Set DATAEOFs (A[0] = 30)**

This procedure returns the DATAEOFs of the standard dataset.

**Parameters**

A[1] = Number of words in array

Starting at A[2] are pairs of words -- one pair per structure.

A[N] = As follows:

Field	Contents
[47:12]	Structure number
[35:16]	Partition number (Equals 0 if not partitioned)

### Procedure: DATAEOF Values (A[N+1])

UTILITY uses this DMINQ procedure to determine the DATAEOF values for standard data sets when performing an on-line dump. An entry is made in the array for each structure being dumped. The Accessroutines returns the DATAEOF value for the structure as it was two control points ago. UTILITY then checks only the checksums for those blocks of data that have an address less than or equal to DATAEOF.

#### Parameters

None

### Procedure: GETDATA with KEYCOMPARE (A[0] = 31)

When the KEYCOMPARE option is set for the database, the key is extracted from the desired record for the specified set and compared with the key value in A[3]. If equal, the record is moved to the user work area. If not equal or the set number provided is not a valid spanning set, an integrity error subcategory 1 is returned. Parameters

A[1] = AA of the desired record

A[2] = Structure number of the set used on the previous pathfinder call

A[3] = For key size equal to user key area after previous pathfinder call

### Procedure: Off-line Dump (A[0] = 32)

SYSTEM/DMUTILITY uses this DMINQ function to make sure there are no other updaters using the database when the off-line dump starts. If other updaters are using the database, then SYSTEM/DMUTILITY waits until they finish. SYSTEM/DMUTILITY then locks the database for the off-line dump.

SYSTEM/DMUTILITY also uses the DMINQ function to notify the database that the off-line dump is complete.

#### Parameters

A[1] = 0 indicates offline dump is beginning

A[1] = 1 indicates offline dump is complete

## **Procedure: Set Cursor (A[0] = 33)**

The Set Cursor function dynamically opens and/or switches to a different invocation of the structure. This offers multiple paths into the same structure without multiple invocations at compile time.

Set Cursor is always at the disjoint data set level and implicitly sets the cursor of all embedded and spanning structures. The DMINQ function can be called for any structure; however, it always locates the disjoint data set for the structure and calls Set Cursor on that disjoint data set. A Set Cursor performed on a DMSII set (as opposed to a data set) changes the cursor for that set only. No data set cursors are affected.

### **Example**

```
ENVINX := DMSIBINDEX (EMP);  
A[0] := 33;  
A[1] := CURSOR;  
DMINQ [ENVINX] (A);
```

### **Parameters**

A[1] = Cursor

## **Procedure: Set Current Path to Another Cursor's Path (A[0] = 34)**

The Set Current Cursor Path function equates the path of the current cursor to the path of another cursor. Thus, after a Set Cursor function, which opens or switches cursors, the current records and set paths can be equated to another cursor.

For data sets, the lock status of records is also equated. As with multiple compile-time invocations, lock and secure procedures performed by the same user in different invocations do not cause a deadlock.

### **Parameters**

A[1] = Other cursor

### DMINQ Arrays

Three arrays are defined for use with the DMINQ interface function: DMKEYAREA, DMWORKAREA, and DMSIBDESC. Note that these arrays can be used only in array reference assignment statements. They cannot be used as ordinary arrays. These arrays are described as follows:

- DMKEYAREA

A hexadecimal array that contains the user's key area for the one SIB invoked.

- DMWORKAREA [<structure environment index>]

A hexadecimal array that contains the user's work area for a particular structure.

The structure index is an arithmetic expression whose value equals the SIB index for that structure (see below).

- DMSIBDESC

A real array that contains the SIB description built by the database interface at compile time for the database invoked. The first N words of this array (0 to N-1) contain environment words describing the structure invoked. The index in the SIB description of this word is the value of DMSIBINDEX for that structure.

The array reference to which DMSIBDESC is assigned is marked read-only; data can be retrieved but not stored through that reference.

In a program that declares exactly one database, which is not DYNAMIC, DMSIBDESC is equivalent to <database id>.DMSIBDESC (where the <database id> is the identifier in the database declaration).

For more information on array references, see "Reference Variables Declaration and Assignment" in the section "DMALGOL Extensions to ALGOL Declarations and Statements."

## DMPROCOFF Statement

The DMPROCOFF statement establishes a Stuffed Indirect Reference Word (SIRW) to a procedure offset array in the database stack. This statement is used to enter user or interenvironment procedures into the Access routines.

**<DMPROCOFF statement>**

– DMPROCOFF – ( – <array name 1> – [ – <subscript> – ] – , – <array name 2> – ) ————|

### Explanation

The SIRW to array name 2 is stored at array name 1 [<subscript>]. The array <array name 2> should be set up using the SIBOFFSET function to contain offsets of procedures to be called using the DMPROCREF function.

The DMPROCOFF statement is used in conjunction with the DMPROCREF function for procedure reference assignment or procedure entry.

### Example

```
DMPROCOFF (PROCOFFS[STR], PROCOFFSETS);
```

In this example, an SIRW to PROCOFFSETS is stored at PROCOFFS[STR].

**Note:** *The DMPROCOFF statement is considered unsafe. Use of this construct in new code is not recommended.*

# DMPROCREF Function

The DMPROCREF function constructs a procedure Stuffed Indirect Reference Word (SIRW) from an array of procedure offset array SIRWs and a procedure offset index. It then returns the procedure SIRW.

### <DMPROCREF function>

– DMPROCREF – [ – <array name> – [ – <subscript> – ] – , – <arithmetic  
expression> ——— ]

### Explanation

The array name identifies an array of SIRWs built using the DMPROCOFF statement. The arithmetic expression identifies the procedure desired. The SIRW returned can be used for either procedure reference assignment or VIA procedure entry, as shown in the following example.

### Example

```
PROCEDURE REFERENCE STRSTATS(A);
ARRAY A[0];
BEGIN END;
.
ARRAY A[0:99];
.
STRSTATS := DMPROCREF[PROCOFFS[STR],STRSTATSOFFSET];
STRSTATS(A);
.
STRSTATS VIA DMPROCREF[PROCOFFS[STR],STRSTATSOFFSET] (A);
```

When the VIA procedure entry is used, the parameters are checked only against the procedure named in the statement, not against the procedure referenced by the SIRW constructed.

**Note:** *The DMPROCREF function is considered unsafe. Use of this construct in new code is not recommended; it is suggested that procedure reference arrays be used instead.*

## DMSCAUSE Statement

The DMSCAUSE statement is used for deadlock detection of Data Management System II (DMSII) functions.

**<DMCAUSE statement>**

– DMCAUSE – ( – <arithmetic expression> – ) —————|

### Explanation

The DMSCAUSE statement calls the MCP procedure DMSCAUSE and passes a single real parameter indicated by the <arithmetic expression> construct. The effect of the DMSCAUSE statement call is dependent on this arithmetic expression as follows:

- Arithmetic expression < 0  
Indicates that the calling program has left the transaction state. The program is delinked from the transaction state linkage chain.
- Arithmetic expression = 0
- Indicates that a syncpoint has been completed. All programs waiting for a syncpoint for this database are resumed.
- Arithmetic expression > 0

Indicates that a record for which other users are waiting has been freed. The parameter is the stack number of the previous owner. All programs waiting on that stack number are resumed.

At run time, the operating system restricts the DMSCAUSE statement to use only by the Accessroutines.

The DMSCAUSE statement is used with the DMSWAIT function.

**Note:**    *The DMSCAUSE statement is considered unsafe.*



### DMSFREE Statement

The DMSFREE statement calls the MCP procedure DMSFREE.

**<DMSFREE statement>**

– DMSFREE —————|

#### **Explanation**

The call to the MCP procedure DMSFREE causes all records locked by this process to be freed in every database visible to the process.

**Note:**    *The DMSFREE statement is considered unsafe.*

## DMSFREEZE Function

The DMSFREEZE function calls the MCP procedure DMSFREEZE. The DMSFREEZE function is invoked to indicate to the operating system when a database stack has been built and users can be attached.

<DMSFREEZE function>

– DMSFREEZE —————|

### Explanation

The DMSFREEZE function is a Boolean function that returns FALSE if the freeze is successful. It returns a value of TRUE in the low order bit and an exception type in [19:16] if the freeze fails.

The exception types returned if the freeze fails are as follows:

- The environment calling DMSFREEZE does not have an Software Control Word (SCW) within it.
- There are no stacks waiting to attach to the database.
- The caller is already an active frozen database.
- The caller is not a database stack initiated by DMSOPEN.

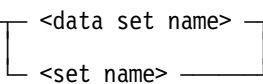

The DMSFREEZE function can be executed only by a code file initiated by the operating system as the result of the MCP procedure DMSOPEN (that is, by an Accessroutine).

**Note:** *The DMSFREEZE function is considered unsafe.*

### DMSIBINDEX Intrinsic

The DMSIBINDEX intrinsic returns the Stuffed Indirect Reference Word (SIRW) offset of the structure indicated by the data set name or the set name.

**<DMSIBINDEX intrinsic>**

– DMSIBINDEX – (  ) 


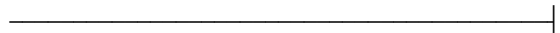
#### Explanation

The data set name is the external name of the data set to be invoked. The set name is the external name of the set to be invoked.

### DMSLOCALREF Statement

The DMSLOCALREF statement calls the MCP procedure DMSLOCALREF. This procedure is used to establish a reference to local Data Management System II (DMSII) buffers.

**<DMSLOCALREF statement>**

– DMSLOCALREF – ( – <direct array reference identifier> – ,   
→ <direct array identifier> – ) 

The <direct array identifier> is an array reference identifier that is declared to be DIRECT.

#### Explanation

The two arrays must be declared with the same number of dimensions.

**Note:** The DMSLOCALREF statement is considered unsafe.

## DMSUPDATEDISKHEADER Statement

This statement causes the disk header for the designated file to be updated in the directory.

**<DMSUPDATEDISKHEADER statement>**

– DMSUPDATEDISKHEADER – ( – <file designator> – ) —————|

### Explanation

For information on the file designator construct, see the *ALGOL Programming Reference Manual, Volume 1*.

**Note:** The DMSUPDATEDISKHEADER statement is considered unsafe.

## DMSWAIT Function

The DMSWAIT function calls the DMSWAIT procedure in the operating system.

**<DMSWAIT function>**

– DMSWAIT – ( – <arithmetic expression 1> – , – <arithmetic expression 2> →  
→ <arithmetic expression 3> – , – <array identifier> – ) —————|

### Explanation

The DMSWAIT function is a Boolean function with four parameters. The first three parameters are real arithmetic expressions, and the fourth is an array. The effect of the DMSWAIT function is dependent on the values of the second and third parameters as follows:

- Arithmetic expression 2 = 0, arithmetic expression 3 = 0.

Indicates that the calling program needs to wait for a syncpoint to complete on this database. The program is linked into a list of waiting programs for the database and is suspended. When a syncpoint is complete on this database (indicated by some other program doing a DMSCAUSE(0)), this program is resumed and returns a result of FALSE. If a deadlock is detected, this program is resumed and returns a result of BOOLEAN(1). If the program has specified a wait limit and the limit expires before the syncpoint occurs, the program is resumed and returns a value of BOOLEAN(3).

Parameter 1 is a control word obtained from the location specified by parameter 3 in the array given as parameter 4. If the value of the control word from the array changes before suspending the program, a value of FALSE is immediately returned.

- Arithmetic expression 2 = 0, arithmetic expression 3 = -1.

Indicates that the calling program has entered the transaction state. The program is linked into the transaction state linkage for the appropriate database. If the program is already in transaction state at the time of this call, a result of TRUE is returned; otherwise, a result of FALSE is returned.

- Arithmetic expression 2 = -1, arithmetic expression 3 = -1.

If the number of processes waiting for locked records in this database is less than the first parameter, a result of TRUE is returned; otherwise, a result of FALSE is returned.

- Arithmetic expression 2 > 0, arithmetic expression 3 > 0.

Indicates that the calling program needs to wait for a locked record. Parameter 2 is the stack number of the current owner. The calling program is linked into the list of programs that are waiting for the current owner of the record to release the record. The sleep count for this database is incremented by 1, and the program is suspended. When the current owner of the record frees it -- indicated by doing a DMSCAUSE (<owner's stack number>), this program is resumed and a result of FALSE returned. If a deadlock is detected, a result of BOOLEAN(1) is returned. If the program has specified a wait limit and the limit expires before the record is freed, the program is resumed and returns a value of BOOLEAN(3).

Parameter 1 is a control word obtained from the location specified by parameter 3 in the array given as parameter 4. If the value of the control word from the array changes before the program is suspended, a value of FALSE is immediately returned.

**Note:** The DMSWAIT function is considered unsafe.

## DMTRANSLOCK Statement

The DMTRANSLOCK statement performs transaction locking for Data Management System II (DMSII) jobs.

**<DMTRANSLOCK statement>**

– DMTRANSLOCK – ( – <formal array 1> – , – <formal array 2 – ) —————|

### Explanation

The formal array 1 construct refers to a transaction lock, and the formal array 2 construct is its new value.

**Note:** *The DMTRANSLOCK statement is considered unsafe. Use of this construct in new code is not recommended.*

## DSED FUNCTION

The DSED function is used to indicate whether or not a program has been terminated.

**<DSED function>**

– DSED —————|

### Explanation

The DSED function returns a Boolean result of TRUE if the program is terminated. If the program is not terminated, it returns a result of FALSE. This function can be used only in Accessroutines and other programs to which the operating system grants immunity from being terminated by the operator or by other external actions.

### DSWAIT and DSWAITANDRESET Functions

Accessroutines code cannot be discontinued using the DS system command. Therefore, if the normal ALGOL WAIT and WAITANDRESET statements are used in Accessroutines code, a user program is not reactivated when the DS command is used. The DSWAIT and DSWAITANDRESET functions provide a means of reactivating a program waiting in Accessroutines code and indicating that the program was discontinued while waiting. The Accessroutines can then take the appropriate action, and the user program will be terminated when it leaves the Accessroutines code. DSWAIT and DSWAITANDRESET constructs are similar to WAIT and WAITANDRESET, except that a result of 0 indicates the program was discontinued while waiting.

For a description of the WAIT and WAITANDRESET statements, see the section on statements in the *ALGOL Programming Reference Manual, Volume 1*.





process was discontinued while waiting. The Accessroutines can then take the appropriate action, and the user process is terminated when it leaves the Accessroutines code.

Although DSABLE cannot be specified if the INTERRUPTIBLE option is used, the INTERRUPTIBLE option includes the DSABLE features.

The result of the LOCK interlock function is zero (0) if the process is discontinued.

For a description of the LOCK interlock statement, see the *ALGOL Programming Reference Manual, Volume 1*.

## MEMORYDUMP Statement

The MEMORYDUMP statement causes a nonfatal system memory dump.

**<MEMORYDUMP statement>**

– MEMORYDUMP – ( – <string literal> – ) \_\_\_\_\_|

### Explanation

The string literal construct can contain up to 16 characters. Any text over 16 characters is truncated. For more information on this construct, refer to the section on language components in the *ALGOL Programming Reference Manual, Volume 1*.

### Example

```
MEMORYDUMP(" DUMP BY FAULT LOCKED" )
```

This statement results in a system dump with the cause displayed and logged as

```
DUMP BY FAULT LOCKED
```

**Note:** The MEMORYDUMP statement is considered unsafe.

## PURGEASM Statement

The PURGEASM statement purges various process registers and cache memory used by software running on a B 7900 system.

**<PURGEASM statement>**

– PURGEASM —————|

### Explanation

This construct has no parameters. It is valid only when TARGET is set to a B 7900 system.

*Note:* The PURGEASM statement is considered unsafe.

## READLOCKNOPURGE Function

The READLOCKNOPURGE function operates like the ALGOL READLOCK function with the following exception. When the target compiler control option is set for the B 7900 system, the READLOCK is performed without purging cache memory.

**<READLOCKNOPURGE function>**

– READLOCKNOPURGE – ( – <arithmetic expression> – , – <arithmetic variable> – )—————|

### Explanation

For an explanation of the parameters for this construct, see the READLOCK function in the *ALGOL Programming Reference Manual, Volume 1*.

*Note:* The READLOCKNOPURGE function is considered unsafe.

### SIBOFFSET Function

The SIBOFFSET function returns as its result the offset of a procedure in its environment.

**<SIBOFFSET function>**

– SIBOFFSET – ( – <procedure identifier> – ) —————|

#### **Explanation**

The SIBOFFSET function accepts a procedure identifier as its only parameter and returns a value at compile time.

#### **Example**

```
I := SIBOFFSET(PROC1);
```

The offset of the PROC1 is returned into the integer variable I.

## TPS TRANSACTION RECORD CONTROL ITEM ASSIGNMENT Statement

This statement assigns a Boolean or arithmetic expression to a transaction record control item.

**<TPS TRANSACTION RECORD CONTROL ITEM ASSIGNMENT statement>**

— <transaction record id> — <subscript> . — <control item> — := — <expression> —

### Explanation

The <transaction record id> construct identifies the transaction record variable that contains the data item to be referenced. The subscript is an arithmetic expression that identifies a particular element within a table.

For a complete list of control items, see the *DMSII Transaction Processing System (TPS) Programming Guide*.

**Note:** The *TPS TRANSACTION RECORD CONTROL ITEM ASSIGNMENT* statement is considered unsafe.

## TRY Statement

The TRY statement provides a general error/fault protection mechanism that allows a program to maintain normal flow of control.

**<try limited form>**

— [ —<error procedure> — : PROTECTED ] —  
→ <procedure invocation statement>  
→ <procedure reference statement>

### Explanation

The limited form of the TRY statement can be used when protection is needed for procedure calls. This form of TRY has a more limited syntax and provides more limited protection. Unlike the normal form, the protection afforded by the limited form does not begin until the procedure begins executing. For more information about the differences between the normal and limited forms of the TRY statement or expression, see the *ALGOL Programming Reference Manual, Volume 1*.

The “: PROTECTED” option provides additional protection against every type of fault, including “Operator DSed (Just DSed),” “Parent Process Terminated,” and “P-DS by another stack.”

The “: PROTECTED” clause offers protection against all forms and causes of process termination, with no limitations. All exceptions and limitations described for the TRY statement do not apply when “: PROTECTED” is specified.

### Caution

It is considered unsafe to use the “: PROTECTED” option.

Because “: PROTECTED” offers complete and unconditional protection against all forms of process termination, it is vital that code invoked to handle errors be as fast and efficient as possible and that the stack be expanded as little as possible. Otherwise, severe consequences can result, including processes hung in a state that cannot be discontinued using the DS command, or even total system failure.

## Section 4

# Compiler Control Options

Compiler control options provide a means to control many aspects of the compilation of a DMALGOL program. DMALGOL recognizes all the compiler control options available in ALGOL. The ALGOL compiler control options and the syntax for using them are described in the section on compiling programs in the *ALGOL Programming Reference Manual, Volume 1*.

DMALGOL recognizes two compiler control options in addition to those of ALGOL. These options are

### <warnunsafe option>

— WARNUNSAFE —————|

(Type: Boolean, Default: FALSE)

When TRUE, the WARNUNSAFE option causes a warning message to be generated for the first occurrence of each type of unsafe construct.

### <warnallunsafe option>

— WARNALLUNSAFE —————|

(Type: Boolean, Default: FALSE)

When TRUE, the WARNALLUNSAFE option causes a warning message to be generated for every occurrence of an unsafe construct.

If any unsafe constructs were used in your program, the following warning message is generated at the end of the compilation, regardless of the value of the WARNUNSAFE or WARNALLUNSAFE option:

```
**WARNING:  'UNSAFE' CONSTRUCTS ÛFOR SOME LEVELS OF INFOGUARD THIS FILE  
IS NOT EXECUTABLE UNLESS SL'ED OR XP'ED.
```

If a listing was generated to the LINE file and any unsafe constructs were used, the total number of unsafe constructs is shown in the trailer information. If either the WARNUNSAFE or the WARNALLUNSAFE option was TRUE at the end of the compilation, the number of occurrences of each type of unsafe construct is shown in the trailer information.



# Appendix A

## Understanding Railroad Diagrams

This appendix explains railroad diagrams, including the following concepts:

- Paths of a railroad diagram
- Constants and variables
- Constraints

The text describes the elements of the diagrams and provides examples.

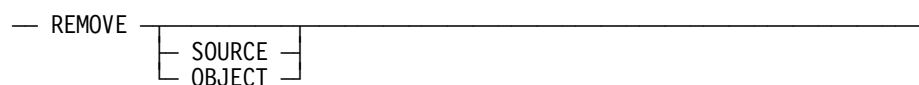
### Railroad Diagram Concepts

Railroad diagrams are diagrams that show you the standards for combining words and symbols into commands and statements. These diagrams consist of a series of paths that show the allowable structures of the command or statement.

#### Paths

Paths show the order in which the command or statement is constructed and are represented by horizontal and vertical lines. Many commands and statements have a number of options so the railroad diagram has a number of different paths you can take.

The following example has three paths:



The three paths in the previous example show the following three possible commands:

- REMOVE
- REMOVE SOURCE
- REMOVE OBJECT

A railroad diagram is as complex as a command or statement requires. Regardless of the level of complexity, all railroad diagrams are visual representations of commands and statements.



## Understanding Railroad Diagrams

---

Railroad diagrams are intended to show

- Mandatory items
- User-selected items
- Order in which the items must appear
- Number of times an item can be repeated
- Necessary punctuation

Follow the railroad diagrams to understand the correct syntax for commands and statements. The diagrams serve as quick references to the commands and statements.

The following table introduces the elements of a railroad diagram:

**Table Appendix A–1. Elements of a Railroad Diagram**

The diagram element...	Indicates an item that...
Constant	Must be entered in full or as a specific abbreviation
Variable	Represents data
Constraint	Controls progression through the diagram path

### Constants and Variables

A constant is an item that must be entered as it appears in the diagram, either in full or as an allowable abbreviation. If a constant is partially boldfaced, you can abbreviate the constant by

- Entering only the boldfaced letters
- Entering the boldfaced letters plus any of the remaining letters

If no part of the constant is boldfaced, the constant cannot be abbreviated.

Constants are never enclosed in angle brackets (<>) and are in uppercase letters.

A variable is an item that represents data. You can replace the variable with data that meets the requirements of the particular command or statement. When replacing a variable with data, you must follow the rules defined for the particular command or statement.

In railroad diagrams, variables are enclosed in angle brackets.

In the following example, BEGIN and END are constants, whereas <statement list> is a variable. The constant BEGIN can be abbreviated since it is partially boldfaced.

Valid abbreviations for BEGIN are

- BE
- BEG
- BEGI

### Constraints

Constraints are used in a railroad diagram to control progression through the diagram. Constraints consist of symbols and unique railroad diagram line paths. They include

- Vertical bars
- Percent signs
- Right arrows
- Required items
- User-selected items
- Loops
- Bridges

A description of each item follows.

### Vertical Bar

The vertical bar symbol (|) represents the end of a railroad diagram and indicates the command or statement can be followed by another command or statement.

— SECONDWORD — ( —<arithmetic expression>— ) —————|

### Percent Sign

The percent sign (%) represents the end of a railroad diagram and indicates the command or statement must be on a line by itself.

— STOP —————%

### Right Arrow

The right arrow symbol ( $\rightarrow$ )

- Is used when the railroad diagram is too long to fit on one line and must continue on the next
- Appears at the end of the first line, and again at the beginning of the next line

— SCALERIGHT — ( —<arithmetic expression>— , —————→  
→<arithmetic expression>— ) —————|

### Required Item

A required item can be

- A constant
- A variable
- Punctuation

If the path you are following contains a required item, you must enter the item in the command or statement; the required item cannot be omitted.

A required item appears on a horizontal line as a single entry or with other items. Required items can also exist on horizontal lines within alternate paths, or nested (lower-level) diagrams.

In the following example, the word EVENT is a required constant and <identifier> is a required variable:

— EVENT —<identifier>—————|

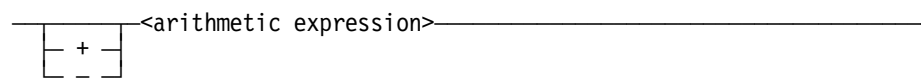
## User-Selected Item

A user-selected item can be

- A constant
- A variable
- Punctuation

User-selected items appear one below the other in a vertical list. You can choose any one of the items from the list. If the list also contains an empty path (solid line) above the other items, none of the choices are required.

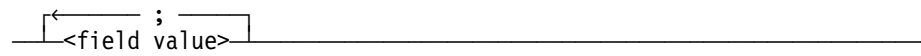
In the following railroad diagram, either the plus sign (+) or the minus sign (−) can be entered before the required variable <arithmetic expression>, or the symbols can be disregarded because the diagram also contains an empty path.



## Loop

A loop represents an item or a group of items that you can repeat. A loop can span all or part of a railroad diagram. It always consists of at least two horizontal lines, one below the other, connected on both sides by vertical lines. The top line is a right-to-left path that contains information about repeating the loop.

Some loops include a return character. A return character is a character—often a comma (,) or semicolon (;)—that is required before each repetition of a loop. If no return character is included, the items must be separated by one or more spaces.



### Bridge

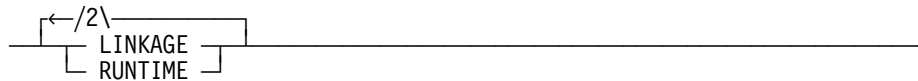
A loop can also include a bridge. A bridge is an integer enclosed in sloping lines (/ \) that

- Shows the maximum number of times the loop can be repeated
- Indicates the number of times you can cross that point in the diagram

The bridge can precede both the contents of the loop and the return character (if any) on the upper line of the loop.

Not all loops have bridges. Those that do not can be repeated any number of times until all valid entries have been used.

In the first bridge example, you can enter LINKAGE or RUNTIME no more than two times. In the second bridge example, you can enter LINKAGE or RUNTIME no more than three times.



In some bridges an asterisk (\*) follows the number. The asterisk means that you must cross that point in the diagram at least once. The maximum number of times that you can cross that point is indicated by the number in the bridge.



In the previous bridge example, you must enter LINKAGE at least once but no more than twice, and you can enter RUNTIME any number of times.

## Following the Paths of a Railroad Diagram

The paths of a railroad diagram lead you through the command or statement from beginning to end. Some railroad diagrams have only one path; others have several alternate paths that provide choices in the commands or statements.

The following railroad diagram indicates only one path that requires the constant LINKAGE and the variable <linkage mnemonic>:

```
— LINKAGE —<linkage mnemonic>—————|
```

Alternate paths are provided by

- Loops
- User-selected items
- A combination of loops and user-selected items

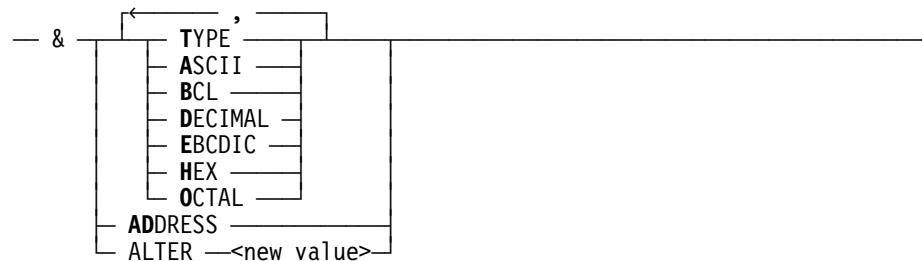
More complex railroad diagrams can consist of many alternate paths, or nested (lower-level) diagrams, that show a further level of detail.

For example, the following railroad diagram consists of a top path and two alternate paths. The top path includes

- An ampersand (&)
- Constants that are user-selected items

These constants are within a loop that can be repeated any number of times until all options have been selected.

The first alternative path requires the ampersand and the required constant ADDRESS. The second alternative path requires the ampersand followed by the required constant ALTER and the required variable <new value>.

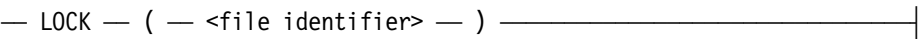


## Railroad Diagram Examples with Sample Input

The following examples show five railroad diagrams and possible command and statement constructions based on the paths of these diagrams.

**Example 1**

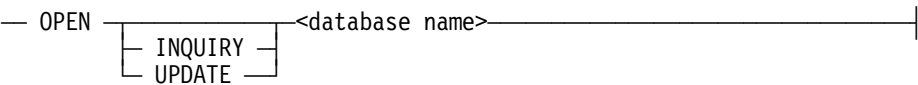
**<lock statement>**



Sample Input	Explanation
LOCK (FILE4)	LOCK is a constant and cannot be altered. Because no part of the word is boldfaced, the entire word must be entered.  The parentheses are required punctuation, and FILE4 is a sample file identifier.

**Example 2**

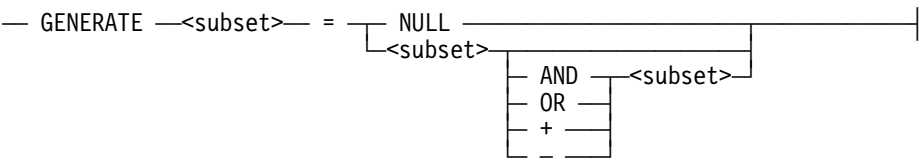
**<open statement>**



Sample Input	Explanation
OPEN DATABASE1	The constant OPEN is followed by the variable DATABASE1, which is a database name.  The railroad diagram shows two user-selected items, INQUIRY and UPDATE. However, because an empty path (solid line) is included, these entries are not required.
OPEN INQUIRY DATABASE1	The constant OPEN is followed by the user-selected constant INQUIRY and the variable DATABASE1.
OPEN UPDATE DATABASE1	The constant OPEN is followed by the user-selected constant UPDATE and the variable DATABASE1.

Example 3

<generate statement>

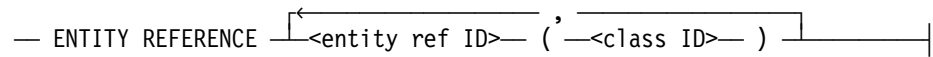


Sample Input	Explanation
GENERATE Z = NULL	The GENERATE constant is followed by the variable Z, an equal sign (=), and the user-selected constant NULL.
GENERATE Z = X	The GENERATE constant is followed by the variable Z, an equal sign, and the user-selected variable X.
GENERATE Z = X AND B	The GENERATE constant is followed by the variable Z, an equal sign, the user-selected variable X, the AND command (from the list of user-selected items in the nested path), and a third variable, B.
GENERATE Z = X + B	The GENERATE constant is followed by the variable Z, an equal sign, the user-selected variable X, the plus sign (from the list of user-selected items in the nested path), and a third variable, B.



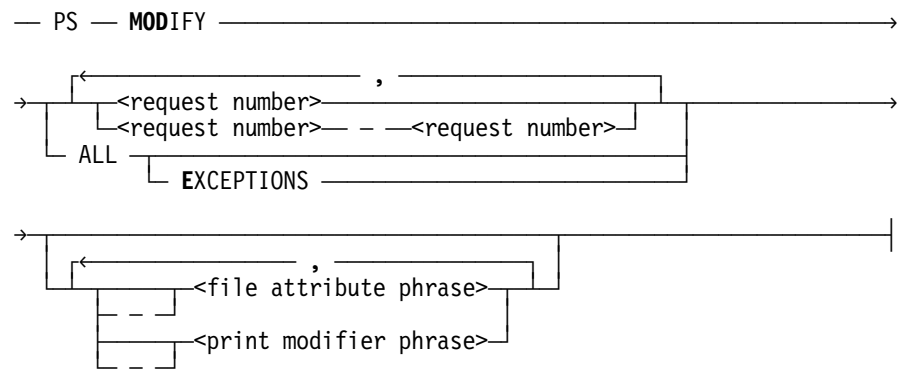
### Example 4

**<entity reference declaration>**



Sample Input	Explanation
ENTITY REFERENCE ADVISOR1 (INSTRUCTOR)	The required item ENTITY REFERENCE is followed by the variable ADVISOR1 and the variable INSTRUCTOR. The parentheses are required.
ENTITY REFERENCE ADVISOR1 (INSTRUCTOR), ADVISOR2 (ASST_INSTRUCTOR)	Because the diagram contains a loop, the pair of variables can be repeated any number of times.

Example 5



Sample Input	Explanation
PS MODIFY 11159	The constants PS and MODIFY are followed by the variable 11159, which is a request number.
PS MODIFY 11159,11160,11163	Because the diagram contains a loop, the variable 11159 can be followed by a comma, the variable 11160, another comma, and the final variable 11163.
PS MOD 11159–11161 DESTINATION = "LP7"	The constants PS and MODIFY are followed by the user-selected variables 11159–11161, which are request numbers, and the user-selected variable DESTINATION = "LP7", which is a file attribute phrase. Note that the constant MODIFY has been abbreviated to its minimum allowable form.
PS MOD ALL EXCEPTIONS	The constants PS and MODIFY are followed by the user-selected constants ALL and EXCEPTIONS.



# Appendix B

## Reserved Words

The table in this appendix lists reserved words that are unique to DMALGOL. ALGOL reserved words are also valid in DMALGOL programs. For a complete list of ALGOL reserved words, see the *ALGOL Reference Manual, Vol. 1*. See the *ALGOL Reference Manual, Vol. 2* for a complete list of the reserved words in Unisys Extended ALGOL.

The DMALGOL reserved words listed in this appendix are divided into three types. A reserved word of type 1 can never be declared as an identifier; that is, it has a predefined meaning that cannot be changed. A reserved word of type 2 can be redeclared as an identifier; it then loses its predefined meaning in the scope of that declaration. A reserved word of type 3 is context-sensitive. It can be redeclared as an identifier and, if it is used where the syntax calls for that reserved word, it carries the predefined meaning; otherwise it carries the user-declared meaning.

**Table B–1. DMALGOL Reserved Words**

Type 1 Reserved Words		
EXCEPTION		
PROPERTY		
PROTECTED		
Type 2 Reserved Words		
ALLOW	DMSLOCALREF	MEMORYDUMP
DISALLOW	DMSUPDATEDISKHEAD ER	NODE
DMINQ	DMSWAIT	PURGEASM
DMIO	DMTRANSLOCK	READLOCKNOPURGE
DMPROCOFF	DSED	SAVE
DMSCAUSE	DSWAIT	SIBOFFSET
DMSFREE	DSWAITANDRESET	DMSIBINDEX
DMSFREEZE	DYNAMIC	
DMSIBINDEX	ENVIRONMENT	

continued

## Reserved Words

---

**Table B–1. DMALGOL Reserved Words (cont.)**

Type 3 Reserved Words		
DMKEYAREA	DMPROCREF	DMSIBDESC
DMWORKAREA	VIA	

# Index

:

: PROTECTED clause, TRY statement, 3-37

## A

### Accessroutines

- direct communication with, 3-2
- DMINQ functions for, 3-2
- ENVIRONMENT declaration marked as, 3-33
- up-level array reference assignments in, 1-7

### Accessroutines

- code file
- description, 1-5

ALLOW statement, 3-1

<array designator>, 2-21

<array reference variable>, 2-21

### array values

#### DM INQUIRY interface

- A[0]=1, 3-4, 3-8
- A[0]=10, 3-5
- A[0]=11, 3-5
- A[0]=12, 3-5
- A[0]=13, 3-6
- A[0]=14, 3-6
- A[0]=16, 3-6
- A[0]=17, 3-6
- A[0]=18, 3-7
- A[0]=20, 3-7
- A[0]=22, 3-8
- A[0]=23, 3-8
- A[0]=24, 3-8
- A[0]=25, 3-9
- A[0]=26, 3-18
- A[0]=27, 3-18
- A[0]=28, 3-19
- A[0]=29, 3-19
- A[0]=30, 3-19
- A[0]=31, 3-20
- A[0]=32, 3-20

A[0]=33, 3-21

A[0]=34, 3-21

A[N+1], 3-20

### arrays

- DASDL, processing text directly from, 2-10
- DM INQUIRY, 3-22
- DMKEYAREA, 3-22
- DMSIBDESC, 3-22
- DMWORKAREA, 3-22
- in database declarations, 2-21
- INCLUDE statement for, 2-10
- node variable as index into, 2-11
- references to, 2-21

### arrays

- construct for declaring, 2-1

## C

clear TPS address (DM INQUIRY interface procedure), 3-19

### code files

- Accessroutines, description, 1-5
- unsafe
  - executing as a by-function library, 1-1
  - flagging the use of, 1-2

### compilation

- ONCE-ONLY for shared procedures, 1-7

### compilation

- listing, 2-12

### compiler control options

- WARNALLUNSAFE, 4-1
- WARNUNSAFE, 4-1

### compiler control options

- CTPROC, 2-8

### compile-time facility

- <compile-time variable declaration>, 2-11
- DEFINE statement, 2-8
- DISPLAY statement, 2-12
- ERROR statement, 2-12
- INCLUDE statement, 2-10
- INVOKE statement, 2-10
- LET statement, 2-17
- NODE declaration, 2-11

- PRINT statement, 2-12
- PROCEDURE declaration, 2-17
- PROPERTY declaration, 2-13
- protected exception procedure declaration, 2-15
- using, 1-4
- compile-time facility
  - CTPROC option in, 2-8
- concurrency control facility, 2-2
  - FREE statement in, 2-4
- constructs
  - DM INQUIRY function, 3-2
  - DMCAUSE statement, 3-25
  - DMPROCOFF statement, 3-23
  - DMPROCREF function, 3-24
  - DMSFREE statement, 3-26
  - DMSFREEZE function, 3-27
  - DMSIBINDEX intrinsic, 3-28
  - DMSUPDATEDISKHEADER statement, 3-29
  - DMTRANSLOCK statement, 3-31
  - DSED function, 3-31
  - DSWAIT function, 3-32
  - DSWAITANDRESET function, 3-32
  - environment declaration, 3-33
  - for invoking a procedure declared in a different environment, 1-6
  - MEMORYDUMP statement, 3-34
  - SIBOFFSET function, 3-36
  - TPS transaction record control item assignment, 3-37
- unsafe
  - array declaration, 2-1
  - definition, 1-1
  - DISALLOW statement, 3-1
  - DMCAUSE statement, 3-25
  - DMPROCOFF statement, 3-23
  - DMPROCREF function, 3-24
  - DMSFREE statement, 3-26
  - DMSFREEZE function, 3-27
  - DMSUPDATEDISKHEADER statement, 3-29
  - DMTRANSLOCK statement, 3-31
  - environment declaration, 3-33
  - MEMORYDUMP statement, 3-34
  - programs containing, 1-1
  - TPS transaction record control item assignment, 3-37
- create (DM INQUIRY interface procedure), 3-8
- CTPROC compiler control option, 2-8

## **D**

- DASDL array
  - processing text directly from the, 2-10
- data finder (DM INQUIRY interface procedure), 3-5
- Data Management System II
  - communicating with the Accessroutines, 3-2
  - concurrency control facility, 2-2
  - concurrency control facility
    - FREE statement, 2-4
  - detecting deadlock of functions, 3-25
  - indicating the use of a file by, 2-14
  - locking transactions for, 3-31
- <database id>, 2-21
- database stack (DBS)
  - declaring the contents of, 3-33
  - description, 1-5
- databases
  - array designators for database declaration, 2-21
  - declaring run-time invocable databases, 2-14
  - DYNAMIC DATABASE declaration, 2-14
  - environments and, 1-5
- DATAEOF values (DM INQUIRY interface procedure), 3-20
- date-timestamp for TPS (DM INQUIRY interface procedure), 3-7
- deadlock, detecting for DMSII functions, 3-25
- declaring
  - a node identifier, 2-11
  - a preprocessor define identifier, 2-8
  - a property identifier, 2-13
  - a protected exception procedure, 2-15
  - contents of the database stack (DBS), 3-33
  - messages, 2-16
  - run-time invocable databases, 2-14
- DEFINE statement, 2-8
- delete current (DM INQUIRY interface procedure), 3-8
- detecting deadlock of DMSII functions, 3-25
- <direct array reference identifier>, 3-28
- disabling external interrupts, 3-1
- DISALLOW statement, 3-1
- DISPLAY statement, 2-12
- DM INQUIRY
  - array values
    - A[0]=11, 3-5

## DM INQUIRY (DMINQ)

## array values

A[0]=1, 3-8  
 A[0]=10, 3-5  
 A[0]=12, 3-5  
 A[0]=13, 3-6  
 A[0]=14, 3-6  
 A[0]=17, 3-6  
 A[0]=18, 3-7  
 A[0]=20, 3-7  
 A[0]=22, 3-8  
 A[0]=23, 3-8  
 A[0]=24, 3-8  
 A[0]=25, 3-9  
 A[0]=26, 3-18  
 A[0]=27, 3-18  
 A[0]=28, 3-19  
 A[0]=29, 3-19  
 A[0]=30, 3-19  
 A[0]=31, 3-20  
 A[0]=32, 3-20  
 A[0]=33, 3-21  
 A[0]=34, 3-21  
 A[N+1], 3-20

corresponding array values and procedures  
(table), 3-3

## functions

create, 3-8

## group type layouts

for global static statistics, 3-11

## procedures

clear TPS address, 3-19  
 create, 3-8  
 data finder, 3-5  
 DATAEOF values, 3-20  
 date-timestamp for TPS, 3-7  
 delete current, 3-8  
 DMSREAD, 3-6  
 free current, 3-6  
 get link, 3-6  
 get statistics information, 3-9  
 get status of abort, 3-8  
 GETDATA, 3-5  
 GETDATA with KEYCOMPARE, 3-20  
 off-line dump, 3-20  
 reset TPS abort, 3-8  
 return displayed messages, 3-18  
 return last transaction address, 3-7  
 return software version, 3-19  
 return standard data set  
     DATAEOFs, 3-19  
 set current cursor path, 3-21  
 set cursor, 3-21

set to beginning, 3-5

store current, 3-6

visible DBS message, 3-18

## DM INQUIRY (DMINQ)

interface, 3-2

## DM INQUIRY (DMINQ)

procedures

pathfinder, 3-4

## DM INQUIRY (DMINQ)

array values

A[0]=1, 3-4

## DM INQUIRY (DMINQ)

functions

pathfinder, 3-4

## DM INQUIRY (DMINQ)

functions

data finder, 3-5

## DM INQUIRY (DMINQ)

functions

GETDATA, 3-5

## DM INQUIRY (DMINQ)

functions

DMSREAD, 3-6

## DM INQUIRY (DMINQ)

functions

get link, 3-6

## DM INQUIRY (DMINQ)

array values

A[0]=16, 3-6

## DM INQUIRY (DMINQ)

functions

store current, 3-6

## DM INQUIRY (DMINQ)

functions

free current, 3-6

## DM INQUIRY (DMINQ)

functions

clear TPS flag and date-timestamp, 3-7

## DM INQUIRY (DMINQ)

functions

return last transaction address, 3-7

## DM INQUIRY (DMINQ)

functions

delete current, 3-8

## DM INQUIRY (DMINQ)

functions

get statistics information, 3-9

## DM INQUIRY (DMINQ)

group type layouts

for group type 6, 3-11

## DM INQUIRY (DMINQ)

group type layouts

for group type 1, 3-12



- DM INQUIRY (DMINQ)
  - group type layouts
    - for global dynamic statistics, 3-12
- DM INQUIRY (DMINQ)
  - group type layouts
    - for group type 1, 3-14
- DM INQUIRY (DMINQ)
  - group type layouts
    - for global dynamic statistics, 3-14
- DM INQUIRY (DMINQ)
  - group type layouts
    - for group type 7, 3-15
- DM INQUIRY (DMINQ)
  - group type layouts
    - for structure static statistics, 3-15
- DM INQUIRY (DMINQ)
  - group type layouts
    - for group type 8, 3-16
- DM INQUIRY (DMINQ)
  - group type layouts
    - for structure dynamic statistics, 3-16
- DM INQUIRY (DMINQ)
  - group type layouts
    - for group type 10, 3-16
- DM INQUIRY (DMINQ)
  - group type layouts
    - for group type 10, 3-18
- DM INQUIRY (DMINQ)
  - group type layouts
    - for group type 10, 3-18
- DM INQUIRY (DMINQ)
  - functions
    - set cursor, 3-21
- DM INQUIRY (DMINQ)
  - arrays
    - DMKEYAREA, 3-22
- DM INQUIRY (DMINQ)
  - arrays
    - DMWORKAREA, 3-22
- DM INQUIRY (DMINQ)
  - arrays
    - DMSIBDESC, 3-22
- DMALGOL
  - extensions, 2-1
  - purpose, 1-1
  - using safely, 1-1
- DMALGOL
  - statements (See statements)
- DMCAUSE statement, 3-25
- DMINQ (See DM INQUIRY)
- DMIO file attribute, 2-14
- DMKEYAREA array, 3-22
- DMPROCOFF statement, 3-23

- DMPROCREF function, 3-24
- DMSFREE statement, 3-26
- DMSFREEZE function, 3-27
- DMSIB1, 2-21
- DMSIB2, 2-21
- DMSIBDESC array, 2-21, 3-22
- DMSIBINDEX intrinsic, 3-28
- DMSII (See Data Management System II)
- DMSREAD (DM INQUIRY interface
  - procedure), 3-6
- DMSUPDATEDISKHEADER statement, 3-29
- DMTRANSLOCK statement, 3-31
- DMWORKAREA array, 3-22
- DSED function, 3-31
- DSWAIT function, 3-32
- DSWAITANDRESET function, 3-32
- DYNAMIC DATABASE declaration, 2-14

## E

- enabling external interrupts, 3-1
- ENVIRONMENT declaration
  - purpose of, 1-5
  - syntax, 3-33
  - use in a program, 1-5
- environments
  - and databases, 1-5
  - nesting restriction for, 1-5
- ERROR statement, 2-12
- event references, 2-25
- examples
  - accessing the members of a node's
    - list, 1-5
  - accessing the text section of the DASDL
    - description file, 1-5
  - extracting a property from a node, 1-4
  - invoking a procedure declared in a different
    - environment, 1-6
- EXCEPTION PROCEDURE declaration
  - description of, 2-15
- executing
  - interrupt code, 3-1
- explicit structure lock, 2-5
- extensions
  - to ALGOL constructs, 2-1
  - unsafe, 1-2
- external interrupts
  - executing, 3-1
  - preventing execution of, 3-1

**F**

files  
     indicating the use of by DMSII, 2-14  
 free current (DM INQUIRY interface  
     procedure), 3-6  
 FREE statement, in DMSII concurrency  
     control facility, 2-4  
 functions  
     DMINQ interface, 3-2  
     DMPROCREF, 3-24  
     DMSFREEZE function, 3-27  
     DSED function, 3-31  
     DSWAIT function, 3-32  
     DSWAITANDRESET function, 3-32  
     SIBOFFSET, 3-36  
     unsafe, 1-2

**G**

get link (DM INQUIRY interface  
     procedure), 3-6  
 get statistics information (DM INQUIRY  
     interface procedure), 3-9  
 get status of abort (DM INQUIRY interface  
     procedure), 3-8  
 GETDATA (DM INQUIRY interface  
     procedure), 3-5  
 GETDATA with KEYCOMPARE (DM INQUIRY  
     interface procedure), 3-20  
 global dynamic statistics  
     group type layouts for, 3-12, 3-14  
 global static statistics  
     group type layouts for, 3-11  
 group type layouts for DM INQUIRY function  
     for global dynamic statistics, 3-12, 3-14  
     for global static statistics, 3-11  
     for group type 1, 3-12, 3-14  
     for group type 10, 3-16, 3-18  
     for group type 6, 3-11  
     for group type 7, 3-15  
     for group type 8, 3-16  
     for structure dynamic statistics, 3-16  
     for structure static statistics, 3-15

**I**

identifier  
     declaring as a preprocessor define, 2-8  
     node, 2-11

implicit structure lock, 2-5  
 INCLUDE statement, 2-10  
 INQUIRYSEMANTIC option, 2-19  
 interrupt code  
     executing, 3-1  
     preventing execution of, 3-1  
 intrinsics  
     DMSIBINDEX, 3-28  
 INVOKE statement, 2-10  
 invoking, a procedure declared in a different  
     environment, 1-6

**L**

layouts  
     for, 3-11  
     for DM INQUIRY functions  
         global dynamic statistics, 3-12, 3-14  
         group type 1, 3-12, 3-14  
         group type 10, 3-16, 3-18  
         group type 6, 3-11  
         group type 7, 3-15  
         group type 8, 3-16  
         structure dynamic statistics, 3-16  
         structure static statistics, 3-15  
 LET statement, 2-17  
 library, by-function, executing an unsafe code  
     file as a, 1-1  
 lock  
     explicit, 2-5  
     implicit, 2-5  
 locked records, releasing, 2-4  
 locking  
     transactions for DMSII jobs, 3-31

**M**

MAPPER option, 2-19  
 MEMORYDUMP statement, 3-34  
 message references, 2-16  
 MP (Mark Program) system commands, 1-1

**N**

nesting restriction for environments, 1-5  
 node  
     description, 1-4  
 node  
     identifier, 2-11

- NODE declaration, 2-11
- <node property construct>, 2-13
- NODE variables, 1-4
- nonlocal message references, 2-16

## O

- off-line dump (DM INQUIRY interface procedure), 3-20
- ONCE-ONLY compilation
  - definition, 1-7
  - LET statement, 2-17
  - PROCEDURE declaration, 2-17
  - syntax, 2-17
- OPEN statement extensions
  - INQUIRYSEMANTIC option, 2-19
  - MAPPER option, 2-19
  - PREVERIFY option, 2-19
  - RECONSTRUCT option, 2-19
  - RECOVER option, 2-19
  - REORG option, 2-19
  - UPDATESEMANTIC option, 2-19
  - UTILITY option, 2-19
- opening a database, 2-19
- options
  - compiler control
    - WARNALLUNSAFE, 4-1
    - WARNUNSAFE, 4-1
  - for the OPEN statement (*See* OPEN statement extensions)
- options
  - compiler control
    - CTRPROC, 2-8

## P

- parameters
  - DM INQUIRY interface
    - clear TPS address procedure, 3-19
    - create procedure, 3-8
    - data finder procedure, 3-5
    - DATAEOF values procedure, 3-20
    - date-timestamp for TPS procedure, 3-7
    - delete current procedure, 3-8
    - DMSREAD procedure, 3-6
    - free current procedure, 3-6
    - get link procedure, 3-6
    - get statistics information procedure, 3-9
    - get status of abort procedure, 3-8
    - GETDATA procedure, 3-5
    - GETDATA with KEYCOMPARE procedure, 3-20
    - off-line dump procedure, 3-20
    - pathfinder procedure, 3-4
    - reset TPS abort procedure, 3-8
    - return displayed messages procedure, 3-18
    - return last transaction address procedure, 3-7
    - return software version procedure, 3-19
    - return standard data set DATAEOFs procedure, 3-19
    - set current cursor path procedure, 3-21

- set cursor, 3-21
  - set to beginning, 3-5
  - store current, 3-6
  - visible DBS message, 3-18
- shared, ONCE-ONLY compilation of, 1-7
- PROPERTY
  - declaration, 2-13
  - <property specification>, 2-13
  - definition, 1-4
- <property specification>, 2-13
- PROTECTED clause, TRYstatement, 3-37
- protected exception procedure
  - declaration, 2-15
- purpose of DMALGOL, 1-1

## R

- railroad diagrams, explanation of, A-1
- RECONSTRUCT option, 2-19
- records
  - releasing locked or secured, 2-4
- RECOVER option, 2-19
- reference assignments, up-level, 1-7
- reference variables
  - array references, 2-21
  - file references, 2-24
  - procedure reference arrays, 2-22
  - purpose, 1-7
  - up-level, 2-21
- releasing locked or secured records, 2-4
- reserved words, B-1
- reset TPS abort (DM INQUIRY interface procedure), 3-8
- return
  - displayed messages (DM INQUIRY interface procedure), 3-18
  - last transaction address (DM INQUIRY interface procedure), 3-7
  - software version (DM INQUIRY interface procedure), 3-19
  - standard data set DATAEOFs (DM INQUIRY interface procedure), 3-19

## S

- safe use of DMALGOL, 1-1
- SAVE option, use in DMALGOL
  - Accessroutines, 2-1
- secured records, releasing, 2-4

- set
  - current cursor path (DM INQUIRY interface procedure), 3-21
  - cursor (DM INQUIRY interface procedure), 3-21
  - to beginning (DM INQUIRY interface procedure), 3-5
- shared procedure
  - ONCE-ONLY compilation of, 1-7
- SIBOFFSET function, 3-36
- SIRW (See stuffed indirect reference word)
- SL (Support Library) system command, 1-1
- stack image, building of by DMALGOL, 1-5
- statements
  - DMCAUSE, 3-25
  - DMPROCOFF, 3-23
  - DMSFREE, 3-26
  - DMSUPDATEDISKHEADER, 3-29
  - DMTRANSLOCK, 3-31
  - MEMORYDUMP, 3-34
  - TPS transaction record control item assignment, 3-37
- TRY, 3-37
- unsafe
  - DMCAUSE, 3-25
  - DMPROCOFF, 3-23
  - DMSFREE, 3-26
  - DMSUPDATEDISKHEADER, 3-29
  - DMTRANSLOCK, 3-31
  - MEMORYDUMP, 3-34
  - TPS transaction record control item assignment, 3-37
- statements (See *also* constructs), 3-32
- store current (DM INQUIRY interface procedure), 3-6
- structure dynamic statistics, group type layouts for, 3-16
- structure lock
  - explicit, 2-5
  - implicit, 2-5
- <structure number>, 1-6
- structure static statistics, group type layouts for, 3-15
- stuffed indirect reference word (SIRW)
  - function for constructing a procedure SIRW, 3-24
  - returning the SIRW offset, 3-28
  - statement for establishing, 3-23
- system commands
  - MP (Mark Program), 1-1
  - SL (Support Library), 1-1
- SYSTEM/DMUTILITY, use of off-line dump procedure, 3-20

**T**

- text section
  - of DASDL description file
    - accessing, 1-5
- text section, of DASDL description file
  - description of, 1-5
- text, processing directly from DASDL
  - arrays, 2-10
- TPS transaction record control item
  - assignment statement, 3-37
- transaction locking, 3-31
- <transaction record id>, 3-37
- <transaction variable id>, 2-21
- TRY statement,
  - PROTECTED clause, 3-37

**U**

- unsafe
  - code files
    - executing as a by-function library, 1-1
    - flagging the use of, 1-2
  - constructs
    - array declaration, 2-1
    - definition, 1-1
    - DISALLOW statement, 3-1
    - DMCAUSE statement, 3-25
    - DMIO file attribute, 2-14
    - DMPROCOFF statement, 3-23
    - DMPROCREF function, 3-24
    - DMSFREE statement, 3-26
    - DMSFREEZE function, 3-27
    - DMSUPDATEDISKHEADER
      - statement, 3-29
    - DMTRANSLOCK statement, 3-31
    - DYNAMIC DATABASE declaration, 2-14
    - INQUIRYSEMANTIC option, 2-19
    - MAPPER option, 2-19

- MEMORYDUMP statement, 3-34
- nonlocal message references, 2-16
- ONCE-ONLY compilation, 2-17
- OPEN statement, 2-19
- PROCEDURE declaration, 2-17
- PROCEDURE REFERENCE ARRAY
  - declaration, 2-22
- programs containing, 1-1
- TPS transaction record control item
  - assignment, 3-37
- UPDATESEMANTIC option, 2-19
- up-level event references, 2-25
- up-level file reference
  - assignments, 2-24
- extensions, 1-2
- functions, 1-2
- statements, 1-2
- UPDATESEMANTIC option, 2-19
- up-level reference assignments, 1-7
- up-level reference variables, 2-21
- using DMALGOL safely, 1-1
- utilities, SYSTEM/DMUTILITY, use of off-line
  - dump procedure, 3-20
- UTILITY option, 2-19

**V**

- variables
  - reference, purpose of, 1-7
  - up-level, 2-21
- verification checking for event
  - references, 2-25
- VIA procedure entry, 3-24
- visible DBS message (DM INQUIRY interface
  - procedure), 3-18

**W**

- WARNALLUNSAFE, 4-1
- WARNUNSAFE, 4-1
- words, reserved, B-1



