# Unisys e-@ction Application Development Solutions

## DCALGOL

## Programming Reference Manual

**ClearPath Release MCP 7.0**

Printed in USA
November 2001
8600 0841–202

# Unisys e-@ction Application Development Solutions

## DCALGOL
## Programming Reference Manual

UNISYS

**ClearPath Release MCP 7.0**

Printed in USA
November 2001
8600 0841–202

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 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.

Correspondence regarding this publication can be e-mailed to **doc@unisys.com**.

Unisys e-@ction
Application Development
Solutions

DCALGOL

**Programming Reference
Manual**

**ClearPath Release
MCP 7.0**

8600 0841–202

Unisys e-@ction
Application
Development
Solutions

DCALGOL

**Programming
Reference
Manual**

**ClearPath
Release
MCP 7.0**

8600 0841–202

Bend here, peel upwards and apply to spine.

# Page Status

| Page | Issue |
|---|---|
| xii through xi | –202 |
| xii | Blank |
| xiii | –202 |
| xiv | Blank |
| 1–1 through 1–13 | –202 |
| 1–14 | Blank |
| 2–1 through 2–13 | –202 |
| 2–14 | Blank |
| 3–1 through 3–48 | –202 |
| 4–1 through 4–9 | –202 |
| 4–10 | Blank |
| 5–1 through 5–107 | –202 |
| 5–108 | Blank |
| 6–1 through 6–73 | –202 |
| 6–74 | Blank |
| A–1 through A–5 | –202 |
| A–6 | Blank |
| B–1 through B–3 | –202 |
| B–4 | Blank |
| C–1 through C–10 | –202 |
| D–1 through D–3 | –202 |
| D–4 | Blank |
| Index–1 through 10 | –202 |

# Page Status

# Contents

**Section 1. Declarations**

**Section 2. Statements**

**Section 3. Functions**

## Section 4. Attributes

## Section 5. DCWRITE Information

# Contents

# Figures

# Figures

# Tables

# Section 1
# Declarations

## About This Guide

### Purpose

The *DCALGOL Programming Reference Manual* provides information on the Data Communications ALGOL (DCALGOL) language. The manual provides information on all the constructs that have application in the control and implementation of a data communications (data comm) environment. It also documents constructs that are useful for interfacing or implementing specialized system programs, such as supervisor programs and performance monitoring tools.

As of SSR 44.2, the following physical hardware terms are valid only as general conceptual terms:

- Line Support Processor (LSP)
- Network Support Processor (NSP)
- Data Communications Data Link Processor (DCDLP)

This manual is intended to be used as a reference source, not as a task-based set of instructions.

### Scope

This manual applies to all Unisys computer systems. While DCALGOL is a high-level language that includes the full syntax of the ALGOL language, only those constructs that are directly connected with the writing of a message control system (MCS) to control a data comm environment are included in this manual. These constructs control the following activities:

- Creating messages
- Inserting messages into queues
- Removing messages from queues
- Combining, creating, and changing queues
- Controlling the data comm subsystem

This manual does not document the ALGOL language subset, the SYSTEMSTATUS or MAKEUSER procedures, or the private procedures CANDEFILEHANDLER, CANDEFILETTER, and ATTRIBSEARCHER. Refer to the appendix “Related Product Information,” for a list of manuals that address these topics.

## Audience

This manual is written for ALGOL-proficient data comm programmers who are familiar with the following:

- General strategies for creating an MCS
- Data comm subsystem
- Logical input/output (I/O) operations
- Message and file manipulations

Installation managers should be aware of the power of DCALGOL. Although the language has built-in user restrictions, inexperienced use of the DCALGOL language can cause system failure, data disruption, and operational problems. A program that uses DCALGOL constructs must have the following characteristics:

- Be privileged through use of the system command PP (Privileged Program)
- Be an MCS program
- Run under a privileged usercode or by a CONTROLCARD function

Installation managers can secure the DCALGOL compiler to prevent unauthorized use. Because the DCALGOL compiler is created by compiling the ALGOL symbolic file with the compiler generation option DCALGOL turned on, it might be wise to also protect the ALGOL symbolic file.

## Conventions

The following conventions are observed:

- Unless otherwise noted, all guides referred to in the text of this manual are for ClearPath systems.
- The manual uses railroad diagrams to depict language constructs. A railroad diagram is a way of graphically representing the syntax of a statement. It shows which items are required and which are optional, the order in which items should appear, how often you can repeat an item, and any punctuation required in a statement. For information on reading and using railroad diagrams, refer to the appendix “Understanding Railroad Diagrams.”

The standard arithmetic operators “+”, “-”, “*”, and “**” are used in this manual to signify addition, subtraction, multiplication, division, and exponentiation, respectively.

# EPILOG PROCEDURE Declaration

The EPILOG PROCEDURE declaration allows you to designate a procedure that must be executed before exiting the block in which the EPILOG PROCEDURE declaration is contained. If an EPILOG PROCEDURE declaration exists for a block, the EPILOG PROCEDURE is automatically executed before exiting the containing block. The EPILOG PROCEDURE is not required to be invoked before exiting the containing block. However, the program can explicitly invoke an EPILOG PROCEDURE during execution, if desired.

```
— EPILOG PROCEDURE — <epilog procedure identifier> — ; — <unlabeled statement>-|
```

```
<epilog procedure identifier>

—— <identifier>; ————————————————————————————————————————————————————————|
```

### Explanation

The following restrictions apply to EPILOG PROCEDUREs:

- No parameters are allowed.
- No bad GO TOs are allowed. That is, any attempt to exit the procedure with a GO TO to an outer block is flagged as a syntax error (if the compiler detects it) or a run-time error (if the compiler did not detect it).
- An EPILOG PROCEDURE cannot return a value.
- An EPILOG PROCEDURE cannot contain an EPILOG PROCEDURE declaration. A block or procedure cannot have more than one EPILOG PROCEDURE declaration (or an EPILOG PROCEDURE declaration and an EXCEPTION PROCEDURE declaration).
- An EPILOG PROCEDURE cannot be declared as a formal parameter.
- Certain restrictions are placed on programs that contain EPILOG PROCEDURE declarations. EPILOG PROCEDUREs cannot be declared as EXTERNAL. Only replacement binding can be used. A block or procedure with an EPILOG PROCEDURE declaration in its outer block cannot be used as the host code file when running BINDER. No procedure in a block that has an EPILOG PROCEDURE can be replaced by the BINDER.
- If a program that contains one or more EPILOG PROCEDURE declarations fails due to a fatal stack overflow, the EPILOG PROCEDURES is not executed.
- If the outer block (or procedure) of a program contains an EPILOG PROCEDURE declaration and the <statistics option> is TRUE, the EPILOG PROCEDURE is executed at block exit time before the statistics wrap-up code.
- If certain Data Management System (DMS) functions such as DATABASE OPEN or DATABASE CLOSE are called, it might not be possible to return to the EPILOG PROCEDURE if the executing task is discontinued.

Every procedure with critical locking code or critical block exit code of some type should have an EPILOG PROCEDURE declaration in it. All critical block exit code must be contained in the EPILOG PROCEDURE. Whenever the procedure is exited (either normally or abnormally), the EPILOG PROCEDURE is executed.

### Example

```
BEGIN
FILE OUT(KIND=DISK,MAXRECSIZE=14,AREASIZE=420,AREAS=5);
ARRAY A[0:13];
REAL I;
EPILOG PROCEDURE CLEANUP;
BEGIN
    %IF I=100 THEN PROGRAM TERMINATED NORMALLY
    %IF I<100 THEN PROGRAM TERMINATED ABNORMALLY
    REPLACE POINTER(A) BY " LAST RECORD, I=" ,I FOR 3 DIGITS;
    WRITE(OUT,14,A);
    LOCK(OUT,CRUNCH);
END CLEANUP;
    .
    .
    .
```

# EXCEPTION PROCEDURE Declaration

The EXCEPTION PROCEDURE declaration allows you to designate a procedure that must be automatically executed by the system whenever an abnormal exit occurs for the block in which the EXCEPTION PROCEDURE declaration is contained.

```
── EXCEPTION PROCEDURE ── <exception procedure identifier> ── ; ──────→

→─ <unlabeled statement> ─────────────────────────────────────────────┤
```

```
<exception procedure identifier>

── <identifier> ──────────────────────────────────────────────────────┤
```

## Explanation

An exception procedure is invoked when the block containing it terminates in any way other than a normal exit. It is not automatically invoked on a normal exit of the block. Abnormal exits include the following:

- A discontinue (DS) command
- A bad branch (GO TO) leading out of the block
- A branch to a global label in the block
- Any unhandled fault

An exception procedure can be invoked directly like any other procedure; it can be called, passed as a parameter (with limitations), and so on. The use of a procedure as an exception procedure is allowed only to the block that contains the procedure. A block or procedure cannot contain more than one exception procedure declaration (or an exception procedure declaration and an epilog procedure declaration).

The following restrictions apply to exception procedures:

- An exception procedure must be a named, untyped procedure without parameters, and therefore cannot return a value.
- An exception procedure cannot contain parameters.
- An exception procedure cannot be specified as a formal parameter. However, an exception procedure may be passed as an actual parameter, if the formal parameter is an untyped procedure without parameters.
- An exception procedure cannot contain a bad GO TO statement. That is, any attempt to exit the procedure via a GO TO to an outer block is flagged as a syntax error (if the compiler detects it) or a run-time error (if the compiler could not detect it).
- An exception procedure cannot contain an EXCEPTION PROCEDURE declaration or an EPILOG PROCEDURE declaration.
- An exception procedure cannot be declared as EXTERNAL. Only replacement binding can be used. A subprogram cannot be bound to a block or procedure with an exception procedure declaration in its outer block.

### Example

```
PROCEDURE P1;
  BEGIN
  REAL A, B;
FILE MYFILE (KIND=DISK);
  EXCEPTION PROCEDURE CLEANUP;
    BEGIN
    CLOSE (MYFILE, LOCK);
    END;  % OF EXCEPTION PROCEDURE CLEANUP

  IF MYFILE.AVAILABLE THEN
    BEGIN
    OPEN (MYFILE);
    CLEANUP;          %A DIRECT CALL TO THE EXCEPTION PROCEDURE
    END;
  A := 17* (B + 4);
  END;   % OF PROCEDURE P1.  THE PROCEDURE CLEANUP IS
         % INVOKED AUTOMATICALLY IF WE EXIT P1 ABNORMALLY.
```

# MESSAGE and MESSAGE ARRAY Declarations

A MESSAGE declaration or a MESSAGE ARRAY declaration defines one or more identifiers as messages or message arrays and defines their dimensions and bounds.

```
           ┌←─────────── , ──────────┐
── MESSAGE ─┴─ <message identifier> ─┴──────────────────────────────┤
```

```
<message identifier>

── <identifier> ────────────────────────────────────────────────────┤
```

```
<message array declaration>

── MESSAGE ARRAY ───────────────────────────────────────────────────→

 ┌←──────────────────────────── , ────────────────────────────┐
 │ ┌←────────────── , ──────────────┐                           │
→┴─┴─ <message array identifier> ──┴─ [ ── <bound pair list> ── ] ─┴──┤
```

```
<message array identifier>

── <identifier> ────────────────────────────────────────────────────┤
```

### Explanation

Essentially, messages are special forms of arrays and, with certain restrictions, can be used in the same manner. The dimension of a message or message array equals the number of given dimensions plus one “hidden” dimension with a lower bound of 0, which is used to index a word in the message. This hidden dimension is used only when working with the data inside the message. The number of words in the message is determined by the ALLOCATE statement and can be determined by the SIZE function.

The hidden dimension of a message can be referenced explicitly only as a specific element and not as a row that uses an asterisk (*) as a subscript in the low-order position.

For example, the following is a valid statement that declares MESSAGEID to be a message and assigns the value 1234 to word 5 in that message:

```
MESSAGEID[5] := 1234;
```

The following statement is not valid:

```
DCWRITE(MESSAGEID[*]);
```

The following statement is valid:

```
DCWRITE(MESSAGEID);
```

Each designated dimension can be referenced by an asterisk (*) or an arithmetic expression according to normal ALGOL rules. However, the hidden dimension cannot have an asterisk.

The following example is valid:

```
MESSAGE ARRAY MSGARYID[0:2,0:7];
 REAL REALNUM;
 REALNUM := MSGARYID[1,2,3];
```

The following example is not valid:

```
ALLOCATE(MSGARYID[1,2,*],6);
```

The following is a correct form that would allocate one message of six words:

```
ALLOCATE(MSGARYID[1,2],6);
```

The following is a correct form that would allocate eight messages of six words each:

```
ALLOCATE(MSGARYID[1,*],6);
```

Messages cannot be declared as formal parameters to a procedure, although a word of a message can be passed by a value to a procedure that expects an arithmetic actual parameter. Globally declared messages can be referenced from within nested blocks by using normal scope rules, but not from within nested procedures. Messages cannot be declared in the declaration list of a structure block type declaration or a connection block type declaration.

The application of messages is limited because they cannot be passed by name as parameters to procedures. Therefore, with respect to procedures, messages are explicitly local. By adhering to the defined syntax, array rows can be used anywhere messages can be used (except in the DCWRITE function, the NULL function, and the ALLOCATE statement). Because the entire message is copied into the new area instead of a pointer being passed to the original message, it is slower to use array rows than messages. However, the severe restrictions imposed on the use of messages frequently mandate the use of array rows.

Messages are organized and maintained by a set of operating system procedures. These procedures maintain a list of free space within the save memory pool and allocate this space to processes requesting space by making and passing a descriptor for an area. The procedures also accept space from processes that are finished with an area and then add the space to the available list.

### Examples

```
MESSAGE A,B,C;

MESSAGE ARRAY RSVP[0:5];

MESSAGE ARRAY FROM,TO[0:5], BACK,AHEAD[0:2,0:6];

MESSAGE ARRAY GREETINGS[0:5,0:5];
```

In the following examples, the hidden dimension is referenced explicitly and selects the word in the message:

```
REAL REALID;
MESSAGE MESSAGEID;
MESSAGE ARRAY MESSAGEARRAYID[0:3];
REALID := MESSAGEID[3];
REALID := MESSAGEARRAYID[2,3];
MESSAGEID[3] := 0;
MESSAGEARRAYID[2,3] := 0;
MESSAGEID[3].[4:5] := 0;
```

Invalid examples of the hidden dimension are as follows:

```
MESSAGEID := 0;

MESSAGEARRAYID[*] := 0;
```

You can use a message or message array element within a pointer expression. However, you cannot assign the pointer expression to another declared pointer. You cannot use a pointer variable to reference a message. The following statements are valid:

```
REPLACE POINTER(MESSAGEID) BY POINTERID FOR 6;

REPLACE POINTERID BY POINTER(MESSAGEID[2],8) FOR COUNT;

REPLACE POINTERID BY POINTER(MESSAGEARRAYID[3],8) FOR COUNT;
```

The following statements show invalid usage of pointers:

```
REPLACE POINTERID:POINTER(MESSAGEID[3]) BY "INVALID" ;

POINTERID := POINTER(MESSAGEID);
```

A word of a message can be passed by value to a procedure that expects an arithmetic actual parameter as shown in the following example:

```
PROCEDUREID (MESSAGEID[3]);
```

When the function causes a message to be removed, you cannot use the DCWRITE function and the REMOVE function as part of a statement that acts on that message.

The following examples show invalid usage of the DCWRITE function and the REMOVE function:

```
MESSAGEID[4] := DCWRITE(MESSAGEID);

MESSAGEID[2] := REMOVE(MESSAGEID,QUEUEID);
```

The following example program illustrates the basic forms of MESSAGE usage that also apply to MESSAGE ARRAYs. This example illustrates references only and not valid DCWRITE forms.

```
BEGIN
ARRAY A[0:29];                 % An extra real array.
MESSAGE MSG;                   % The GLOBAL MESSAGE variable.
POINTER MPTR;                  % A pointer to the real array.

PROCEDURE PPP(COMINGIN);       % A nested procedure with an
  VALUE COMINGIN;              %   arithmetic actual parameter.
  REAL COMINGIN;               %
     BEGIN
     MESSAGE PPPMSG;           % A locally declared MESSAGE.
     ALLOCATE(PPPMSG,10);
     A[25] := COMINGIN;
     COMINGIN := " HELLO[|P]';    % From within procedure PPP:
        BEGIN                  %   reference from within a
        POINTER APTR;          %   nested block is permitted
        PPPMSG[1] := COMINGIN; %   for PPP's locally declared
        A[25] := PPPMSG[0];    %   message PPPMSG, but not
        DCWRITE(PPPMSG);       %   permitted for the global
        END;                   %   message MSG.
     END;

BEGIN                        % From within the scope rules
                             %   of the outer block:
MESSAGE ARRAY INNERMSG[0:3];
ALLOCATE(MSG,30);            % Reference from within a nested
MPTR := POINTER(A[1],8);     %   block to the global message
MSG[1] := " *---->" ;          %   MSG is permitted.
A[0] := 0 & MSG[1] [47:48];
   BEGIN                     % Reference from within any further
   REAL AREAL;               %   nested blocks is also permitted
   ALLOCATE(INNERMSG[1],6);  % Reference to a block declared
   MSG[29].[15:16] := " 00" ;  %   message is valid.
   PPP(MSG[1]);              % A message element can be passed
   END;                      %   as a parameter by value.
REPLACE MPTR BY " WELCOME TO UNISYS       " ;
REPLACE POINTER(MSG) + 6 BY MPTR FOR 28 WORDS ; % Message used in
DCWRITE(MSG);                             % a pointer expression
REPLACE MPTR BY "  "  FOR 29 WORDS;
END;
END.
```

# QUEUE and QUEUE ARRAY Declarations

A QUEUE declaration or a QUEUE ARRAY declaration defines one or more identifiers as queues or queue arrays. The QUEUE ARRAY declaration also defines the subscript bounds of the dimensions of the queue array.

The QUEUE declaration syntax follows.

```
           ┌←───────── , ────────┐
── QUEUE ──┴─ <queue identifier> ┴──────────────────────────────┤
```

```
<queue identifier>

── <identifier> ────────────────────────────────────────────────┤
```

The QUEUE ARRAY declaration syntax follows.

```
── QUEUE ARRAY ─────────────────────────────────────────────────→

  ┌←──────────────────────────── , ───────────────────────────┐
  │ ┌←───────────── , ─────────────┐                          │
→─┴─┴─ <queue array identifier> ───┴─ [ ── <bound pair list> ── ] ─┴────┤
```

```
<queue array identifier>

── <identifier> ────────────────────────────────────────────────┤
```

### Explanation

A DCALGOL queue is a linked list of messages. A queue array is an indexable array having elements that are queues, not values. A queue array is not segmented. Queues and queue arrays can be declared as parameters to procedures and can be passed by name only. If a procedure is invoked by a RUN statement, queues and queue arrays cannot be passed as actual parameters.

Each queue (a simple queue or a single element of a queue array) has a queue reference word. If a queue is inactive, its queue reference word is a special data descriptor (with the index bit turned on and an invalid character size field of seven) that identifies this queue as inactive. When the queue is activated, the special data descriptor is replaced by a stuffed indirect reference word (SIRW) that points to a data descriptor that represents the physical queue in the DATACOM QUEUE stack.

The DATACOM QUEUE stack is a stack reserved for DCALGOL queues and contains a data descriptor for each active physical queue. Each such data descriptor addresses an array row in main memory. This array row, sometimes called a “hidden” message, contains information about the physical queue such as the number of messages in the queue, the amount of main memory used by the queue, the number of users of the queue, and so forth. Some of this information is available to you through the use of queue attributes.

A queue has no arithmetic properties. You cannot assign a value to it or use it as a value.

Within a DCALGOL program structure, a simple queue is represented by its queue reference word in the program working stack. A one-dimensional queue array is represented by a data descriptor in the program working stack. This data descriptor points to a set of queue reference words (one per queue array element). For higher dimensional queue arrays, intermediate sets of data descriptors are constructed. A subscripted queue (formed by subscripting a queue array) is represented by a queue reference word accessed through one or more data descriptors.

Queues can be passed as actual parameters to procedures by name only. When a queue is passed by name to a procedure, the formal parameter is represented by an SIRW that points to the original queue reference word. A queue can also be passed as a parameter to a library procedure.

The following conditions cause a queue to be activated implicitly:

- By inserting a message or an array row into it by using the INSERT statement or the COMBINE statement
- By using the ATTACH statement
- By invoking the DCWRITE function to direct the Data Comm Controller (DCC) to place messages in the queue

You can explicitly activate a queue by using the QACTIVE attribute.

When a queue becomes empty, it is not deactivated. A queue automatically becomes inactive (and its messages are removed) when no processes refer to it.

### Examples

```
QUEUE IN;

QUEUE IN,OUT,NEXT;

QUEUE ARRAY WAITING[0:5];

QUEUE ARRAY UP,DOWN[0:3,0:2];

QUEUE ARRAY LINEUP[0:5], CUP,PLATE,SPOON[0:50,0:33];
```

# QUEUE ARRAY REFERENCE Declaration

A QUEUE ARRAY REFERENCE is an array reference variable used only to reference queue arrays.

```
── QUEUE ARRAY REFERENCE ──────────────────────────────────────→

 ┌←─────────────────────────── , ───────────────────────┐
 │ ┌←────────────── , ──────────┐                        │
→┴─┴─ <queue array reference id> ─┴─ [ ── <lower bounds> ── ] ─┴───────┤

<queue array reference identifier>

── <identifier> ─────────────────────────────────────────────────┤
```

### Examples

```
QUEUE ARRAY REFERENCE ABC[3], XYZ[5];

QUEUE ARRAY REFERENCE ARY[2*5];
```

# Section 2
# **Statements**

This section describes DCALGOL statements.

## ALLOCATE Statement

The ALLOCATE statement causes an area of save memory <arithmetic expression> words in length to be reserved for a message.

```
── ALLOCATE ── ( ── <message group designator> ── , ─────────────────→

→─ <arithmetic expression> ── ) ────────────────────────────────────┤
```

```
<message group designator>

──┬─ <message designator> ──────────────────────────────┬───────────┤
  └─ <message array identifier> ─┬──────────────────────┤
                                 └─ <subarray selector> ─┘
```

```
<message designator>

──┬─ <message identifier> ─────────────────────────────────┬────────┤
  │                                      ┌←───── , ─────┐  │
  └─ <message array identifier> ── [ ──┴─ <subscript> ─┴─ ] ─┘
```

### Explanation

When a row or subarray of a message array is allocated, repeated allocations are made for each element of the array, reserving the number of words specified by the <arithmetic expression> for each element. If a message area referenced by a message group designator has already been allocated, the old message area is returned to the system before a new area is allocated to the message group. If the value of the arithmetic expression is less than or equal to zero, the message area, if any, is returned to the system, making the message null. If the arithmetic expression does not yield an integer value, it is rounded to an integer value.

The maximum size of a message that can be allocated is 2736 words. To determine the size of a message perform one of the following tasks:

- View the result of the Initialize Primary Queue (Type 0) DCWRITE request
- Use the SYSTEMSTATUS call to retrieve Software Configuration (Type 17) information

### Examples

```
ALLOCATE(MESSAGEID,SIZEINWORDS);

ALLOCATE(MESSAGEARRAYID[3],SIZEINWORDS);

ALLOCATE(MESSAGEARRAYID,SIZEINWORDS);
```

# ATTACH Statement

The ATTACH statement is used to attach a process to a queue.

```
── ATTACH ── ( ── <new queue> ── , ── <old queue> ── ) ──────────────┤
```

```
<new queue>

── <queue designator> ────────────────────────────────────────────────┤
```

```
<queue designator>

──┬─ <queue identifier> ───────────────────────────────┬──────────────┤
  │                              ┌←──────── , ────┐     │
  └─ <queue array name> ── [ ──┴─ <subscript> ──┴─ ] ─┘
```

```
<queue array name>

──┬─ <queue array identifier> ──────────┬─────────────────────────────┤
  └─ <queue array reference identifier> ─┘
```

### Explanation

The ATTACH statement creates a stuffed indirect reference word (SIRW) that points to the head-tail word referenced by the <old queue> construct and places the SIRW in the stack location referenced by the <new queue>construct.

The ATTACH statement activates the old queue if it is not active. If the new queue is active, the user count of the referenced queue is decreased by one and the new queue is made inactive (detached from the physical queue it previously referenced). (Refer to the queue attribute QUSERCOUNT under “Queue Attributes” in the “Attributes” section.)

The ATTACH statement causes the <new queue> construct to reference the same physical queue as the <old queue> construct, and in this way the new queue is activated. The user count of the physical queue, referenced by both the <new queue> and the <old queue>constructs, is increased by one.

### Examples

```
ATTACH(QUEUEID1,QUEUEID2);

ATTACH(QUEUEARRAYID1[3],QUEUEARRAYID2[3]);
```

# COMBINE Statement

The COMBINE statement combines two separate queues by changing the head-tail links of the queues. The messages and their links are unchanged (except for the link of the one-ended message).

```
── COMBINE ── ( ── <host queue> ── , ── <secondary queue> ──────────→

→─┬───────────────────┬─ ) ─────────────────────────────────────────┤
  └─ , ── <priority> ─┘
```

```
<host queue>

── <queue designator> ──────────────────────────────────────────────┤
```

```
<secondary queue>

── <queue designator> ──────────────────────────────────────────────┤
```

```
<priority>

── <Boolean expression> ────────────────────────────────────────────┤
```

### Explanation

The COMBINE statement empties the messages in the secondary queue into the host queue if it is active; if the host queue is not active, it is activated.

If the secondary queue is inactive, the process that is executing the COMBINE statement is terminated unless an ON statement has enabled the INACTIVEQUEUE interrupt. If the secondary queue is empty, nothing is combined; however, the host queue is activated if it is inactive.

If the priority has the value FALSE or is not designated, the messages from the secondary queue are placed at the tail of the host queue. If the priority has the value TRUE, the messages from the secondary queue are placed at the head of the host queue. When the queues are combined, both queues remain active, but the secondary queue is emptied.

### Examples

```
COMBINE(QUEUEID1,QUEID2);

COMBINE(QUEID1,QUEID2,TRUE);

COMBINE(RESULTQARRAY[3],SUBQUEUE,FALSE);
```

# DCKEYIN Statement

The DCKEYIN statement allows programs to send messages to the operating system and to receive the corresponding system responses. To use this statement, a program must be running under a privileged usercode or a systemuser usercode, or be running as a privileged program.

```
── DCKEYIN ── ( ── <pointer expression> ── , ──────────────────────────→

→─ <noncharacter array row> ── ) ─────────────────────────────────────┤
```

<noncharacter array row>

An array row whose identifier is declared with a <type> construct.

### Explanation

The first parameter must be an EBCDIC pointer expression that points to the first character of the message to be sent. The following rules apply to these messages:

- You can send any system command, such as PD or RES.
- You can send WFL statements that the operator could enter from the ODT, such as RUN and REMOVE.
- You cannot send primitive commands such as ??RJ.
- The message must be terminated by one or more null characters (4“00”).
- The maximum size for the message is 256 words (1536 bytes). If this size is exceeded, the system displays an error message at runtime and terminates the task.

The second parameter is an array row to be used by the DCKEYIN statement to return the response to the system message. The response is a series of logical lines each of which is terminated by a null character (4“00”). The last line of the response is terminated by an ETX character (4“03”).

The array row can have a minimum size of four words and a maximum size of 1,048,571 words. If the array row is less than four words long, DCKEYIN does not send the message to the system. If the array row exceeds the maximum size, the system displays an error message at run time and terminates the task. If the array row is too small to contain a response, the response is truncated.

All responses are truncated to 255 lines or 1,048,576 bytes, depending on circumstances.

If the DCKEYIN statement causes the execution of a WFL job or a system command, the WFL job or the executor of the system command does not inherit the usercode or family substitution specification of the process that executed the DCKEYIN statement. For example, if a task is executing with a family substitution statement DISK = PACK ONLY, and the task uses the DCKEYIN statement to send the command PD = ON DISK, the response refers to the DISK family. As another example, the statement RUN OBJECT/PROG sent through the DCKEYIN statement starts a job without a usercode, no matter what the usercode is of the task that called the DCKEYIN statement.

The DCKEYIN statement calls the DCKEYIN procedure in the GENERALSUPPORT system library, which performs the following actions:

- Checks the parameters and the message
- Copies the message into the noncharacter array row
- Sends the message to the system and waits for a response from the system
- Exits

The DCKEYIN procedure does not send the message to the system if it detects the following errors:

- The pointer expression is uninitialized or is not initialized as an 8-bit pointer expression. This error returns the response NOT EIGHT BIT POINTER.
- The first nonblank character of the message is not an EBCDIC alphanumeric character, or is a lowercase letter. This error returns the response ILLEGAL INITIAL CHARACTER.

A segmented array fault can occur if the message does not terminate with a null character or if the noncharacter array row is too small to contain the copy of the input message.

If the noncharacter array row is too small to contain all the information requested, the response is truncated and terminated with an end of text (ETX) character.

### Example

This example demonstrates the use of a DCKEYIN statement to execute the system command PD OBJECT/= ON PACK.

Because the response to a DCKEYIN statement is limited to 255 lines, the response to a PD command submitted through DCKEYIN might not show all the files in the directory.

```
ARRAY SYSMSG [0:50];              % INPUT SYSTEM MESSAGE
ARRAY RESPONSE [0:10000];         % RESPONSE FROM SYSTEM

  POINTER PR;
FILE LINE (KIND = PRINTER);
  ARRAY PBUF [0:22];

REPLACE POINTER (SYSMSG) BY "PD OBJECT/= ON PACK", 48"00";
DCKEYIN (POINTER (SYSMSG), RESPONSE);

PR := POINTER (RESPONSE);
WHILE REAL (PR, 1) NEQ 48"03" DO   % LOOP UNTIL ETX CHARACTER
  BEGIN
    REPLACE POINTER (PBUF) BY " "  FOR 132;
    REPLACE POINTER (PBUF) BY PR:PR FOR 132 WHILE GTR 48"03";
    WRITE (LINE, 132, PBUF);
    IF PR LSS 48"03" THEN          % END OF LINE?
      PR := PR + 1;                % ADVANCE TO NEXT LINE
  END;
```

# FLUSH Statement

The FLUSH statement causes all messages in the queue to be discarded. The queue remains active. Messages that have been flushed from the queue no longer exist in the system.

```
── FLUSH ── ( ── <queue designator> ── ) ──────────────────────────────┤
```

### Explanation

If the queue designated by the <queue designator> construct is empty, the FLUSH statement is ignored. The process executing the FLUSH statement is not terminated if the queue designated by the <queue designator> construct is not active.

### Examples

```
FLUSH(QUEUEID);

FLUSH(QUEUEARRAYID[3]);
```

# INSERT Statement

The INSERT statement causes a message to be linked into the queue referenced by the <queue designator>construct.

```
── INSERT ── ( ── <insert source part> ── , ── <queue designator> ────→

→─┬──────────────────┬─ ) ─────────────────────────────────────────┤
  └─ , ── <priority> ─┘
```

```
<insert source part>
──┬─ <message designator> ───────────────────────┬────────────────────┤
  └─ <noncharacter array row> ── , ── <use size> ─┘
```

```
<use size>

──┬─ * ──────┬────────────────────────────────────────────────────────┤
  └─ <size> ─┘
```

```
<size>

── <arithmetic expression> ───────────────────────────────────────────┤
```

### Explanation

If the insert source part is a message designator, the message area is linked into the queue, which leaves the message designator null.

If the insert source part is a noncharacter array row and the use size is an asterisk (*), a message is allocated that has a length equal to the length of the noncharacter array row. If the use size includes the <size>construct, the length in words of the allocated message area is specified by the <size> cosntruct. Therefore, if the use size is declared as an asterisk, the noncharacter array row is copied into the allocated message area; otherwise, the number of words of the noncharacter array row specified by the <size> construct and starting with the first word are copied. If the declared size is not greater than 0, a fatal run-time MESSAGE SIZE ERROR results. If the size construct does not yield an integer value, it is rounded to an integer value. The message is linked into the queue, leaving the original noncharacter array row intact.

If the referenced queue is inactive, an implicit activation is performed on the queue. Inserting a null message in an inactive queue makes the queue active (but empty), while inserting a null message in an active queue is ignored.

If the priority has the value FALSE or is not designated, the message is linked to the tail of the queue; otherwise, the message is linked to the head of the queue. Linking a message to the tail of a queue preserves the time order of the messages in that queue. Linking a message to the head of the queue ensures that the message is the next one removed from that queue, unless another message is linked to the head in the meantime.

### Examples

```
INSERT(MESSAGEID,QUEUEID);

INSERT(MESSAGEID,QUEUEID,PRIORITYBIT);

INSERT(MESSAGEARRAYID[1],QUEUEARRAYID[1]);

INSERT(ARRAYID[*],3,QUEUEID);

INSERT(ARRAYID[*],*,QUEUEARRAYID[3],TRUE);
```

# ON Statement

The ON statement is used to enable or disable an interrupt for one or more fault conditions.

DCALGOL accepts the same syntax for the ON statement as ALGOL does, but DCALGOL allows the use of the following additional fault name:

```
INACTIVEQUEUE
```

### Explanation

The ON statement is documented in the *ALGOL Programming Reference Manual, Volume 1: Basic Implementation.* The fault that is enabled or disabled by designating INACTIVEQUEUE in the fault list is an attempt to use an inactive queue. The fault number value for INACTIVEQUEUE is 6. Refer to the “FLUSH Statement” in this section for information on what can cause an INACTIVEQUEUE interrupt. On ASD machines, a process can never receive an INACTIVEQUEUE interrupt.

### Examples

```
ON INACTIVEQUEUE, GO TO ERROREXIT; %ENABLE INTERRUPT

ON INACTIVEQUEUE,
     BEGIN
     WRITE(FILEID,<" I AM IN TROUBLE" >);
     GO TO EXITLABEL;
     END;

ON INACTIVEQUEUE;                  %DISABLE INTERRUPT
```

# RESIDENT Statement

The RESIDENT statement designates that certain data and code are to be resident at all times in primary storage instead of being overlayed to secondary storage.

```
── RESIDENT ── ( ── <Boolean expression> ── , ── <resident list> ── ) ─┤
```

```
<resident list>
  ┌←────────────────── , ────────────────────┐
──┴─┬─ * ─────────────────────────────────────┴──────────────────────────┤
    ├─ <procedure identifier> ─────────────┤
    ├─ <epilog procedure identifier> ──────┤
    ├─ <nondirect array name> ─────────────┤
    ├─ <nondirect array row> ──────────────┤
    ├─ <nondirect subscripted variable> ───┤
    ├─ <truthset identifier> ──────────────┤
    ├─ <translatetable identifier> ────────┤
    ├─ <format identifier> ────────────────┤
    └─ <switch file identifier> ───────────┘
```

```
<nondirect array name>

──┬─ <array identifier> ───────────────────────┬──────────────────────────┤
  ├─ <nondirect array reference identifier> ───┤
  └─ <value array identifier> ─────────────────┘
```

<nondirect array reference identifier>
An array reference identifier that is not declared to be DIRECT.

```
<nondirect array row>

──┬─ <one-dimensional nondirect array name> ──┬──────────────────────────┤
  └─ <nondirect array name> ── <row selector> ┘
```

<one-dimensional nondirect array name>
A nondirect array name whose identifier is declared with one dimension.

```
<nondirect subscripted variable>

                                ┌←─────── , ───┐
── <nondirect array name> ── [ ─┴─ <subscript> ─┴─ ] ──────────────────┤
```

### Explanation

If the Boolean expression has the value TRUE, the listed elements are made present in primary storage and are marked as resident so that they reside in primary storage whenever the program is active (until deallocated). If the Boolean expression has the value FALSE, any listed elements marked as resident are unmarked and, thereafter, are treated as normal, overlayable information.

When the resident element is an asterisk (*), the code segment that contains the RESIDENT statement is affected.

For a procedure identifier or epilog procedure identifier, the code segment that contains the entry point to that procedure is affected. Segments that contain other blocks within that procedure are unaffected. Large procedures can have their entry point in a separate stack-building segment; in this case, the segment that contains the actual body of the procedure is unaffected.

For a nondirect array name or nondirect array row, the entire array or array row is affected and includes all rows (if multidimensional) or segments (if segmented).

For a nondirect subscripted variable, only the row or segment that contains the element is affected.

A truthset, translate table, format, or switch file is implemented as an array; the appropriate identifier as a member of the resident list affects that array. Several truthsets, translate tables, or formats can reside in the same array.

Redundant setting or resetting of residency is ignored.

If all the requested elements cannot be made present in primary storage, the system issues an operator message and might wait for a response.

When an item named in a resident list is deallocated by a block exit or by an explicit DEALLOCATE statement, storage is released and residency is forgotten. Residency is also lost if an array is resized.

Resident storage is not the same as save storage; that is, the information can be moved from one place to another within primary storage (or within the user's subspace) during execution of the program. This property makes resident storage less disruptive to the storage management algorithms because resident storage need not contribute to “checkerboarding.” However, the RESIDENT statement must be used judiciously and sparingly if overall system performance is not to be degraded by usurpation of resources. An appropriate use might be to keep the critical tables and code segments resident in a real-time application.

Resident storage is reported as save storage in the CU (Core Usage) system message.

### Examples

Given appropriate declarations, the following statement retains the procedure HANDLETRANSACTION and the current segment:

```
RESIDENT(TRUE,HANDLETRANSACTION,*);
```

The following statement retains or releases (according to KEEPIT) an array and two rows of a two-dimensional array:

```
RESIDENT(KEEPIT,HASHHEAD,INFO[0,*],INFO[1,*]);
```

The following statement releases all the rows of MATRIX and one row of NET:

```
RESIDENT(FALSE,MATRIX,NET[3,*]);
```

The following statement retains the first N segments of array WORKSPACE:

```
FOR I := 0 STEP 1 UNTIL N DO RESIDENT(TRUE,WORKSPACE[256*I]);
```

# STANDARDTODISPLAY Statement

The STANDARDTODISPLAY statement converts the standard form file title in the <standard location> construct to display form and stores the result, followed by a period (.), at the location specified by the display pointer. (Refer to the DISPLAYTOSTANDARD Function in the “Functions” section for a description of standard and display form file titles.)

```
── STANDARDTODISPLAY ── ( ── <standard location> ── , ────────────────→

→─ <display pointer> ── ) ─────────────────────────────────────────────┤
```

<standard location>

```
<standard location>

── <pointer expression> ───────────────────────────────────────────────┤
```

```
<display pointer>

── <pointer identifier> ───────────────────────────────────────────────┤
```

### Explanation

The display pointer is a call-by-name pointer parameter that is left updated to the character following the period that terminates the file title.

The compiler control option INSTALLATION 1 must be assigned to allow the compiler to access the required installation intrinsic.

### Example

```
STANDARDTODISPLAY(PTRSTANDARD,PTRDISPLAY);
```

Section 3
# Functions

This section describes the various DCALGOL functions. For information regarding the USERDATA, USERDATALOCATOR and USERDATABUILD functions refer to the *ALGOL Programming Reference Manual, Volume 1: Basic Implementation.*

## ATTACHSPOQ Function

When used in conjunction with a DCALGOL queue, the ATTACHSPOQ function allows a DCALGOL program to monitor much of the supervisory console message traffic. You must be a privileged user to use this function.

```
── ATTACHSPOQ ── ( ── <queue designator> ── ) ────────────────────────┤
```

### Explanation

The queue designator is the destination for copies of various console-related messages. The queue designator must designate an inactive DCALGOL queue. If the queue is successfully attached, the Boolean function returns the value TRUE. Furthermore, copies of messages generated by the procedure MESSER, messages sent to the CONTROLLER, and CONTROLLER responses are inserted in the queue designated by the <queue designator> construct and can be removed in the normal way. After it is attached for console-related messages, the queue can be detached only by exiting the block in which that queue is declared.

If the queue is already activated or if another queue is attached to receive console messages, the invocation of the ATTACHSPOQ function is unsuccessful, the value returned is FALSE, and the state of the indicated queue is unaffected.

Messages are received as EBCDIC characters starting in word 1. Word 0 of the message contains the time of day in 2.4-microsecond units. Messages can contain nonprinting characters that are used to control the console unit.

### Examples

```
IF ATTACHSPOQ(QUEUEIDENTIFIER) THEN
   GO TO SUCCESSFUL________
ELSE
   GO TO UNSUCCESSFUL;

IF ATTACHSPOQ(QUEUEID[4]) EQV BOO THEN
   GO TO EXPECTED;
```

# CHECKGUARDFILE Function

The CHECKGUARDFILE function checks access privileges based on whether the CHECKDECLARER option of the SECOPT (Security Options) system command is TRUE or FALSE. For information on the CHECKDECLARER option, refer to the *Security Administration Guide*. You should be familiar with the security checking scheme of this option before you invoke the CHECKGUARDFILE function.

```
— CHECKGUARDFILE — ( — <pointer expression> — ) ————————————————┤
```

### Explanation

The pointer expression is passed by value and must point to a file title of a guardfile. This title must be in display format, including any ON <familyname> clause. In addition, you must enter the complete file title, which includes the usercode. The system does not apply family substitution or usercode inheritance to search for the guard file.

The function returns a Boolean value. If all bits of the result are equal to 1, the guardfile is not available. Otherwise, the result is interpreted as two masks. The first mask describes actions relevant to files in general. The second mask describes actions relevant only to databases. If a particular bit in the first mask is turned on, access to the action associated with that bit is denied. If a particular bit in the second mask is turned on, access to the action associated with that bit is allowed.

The following bits describe actions relevant to files in general:

| Bit | Meaning if Bit is Set |
|---|---|
| 0 | Writing to the file is not allowed. |
| 1 | Reading the file is not allowed. |
| 2 | Executing the file is not allowed. |
| 41 | Writing to a file opened EXTEND (as defined by the ANSI74 COBOL standard) is not allowed. |
| 45 | No access is allowed. |

***Note:*** *If bit 45 is set (meaning no access is allowed), the other permission bits (0, 1, 2, and 41) are also set.*

The following bits describe actions relevant only to databases:

| Bit | Description |
|---|---|
| 4 | FIND |
| 5 | LOCK |
| 6 | OPEN INQUIRY |
| 7 | SECURE |
| 16 | ASSIGN |
| 17 | CREATESTORE |
| 18 | DELETE |
| 19 | GENERATE |
| 20 | INSERT |
| 21 | LOCKSTORE |
| 22 | REMOVE |
| 23 | OPEN UPDATE |
| 32 | CLOSELOCK |
| 33 | OPEN INITIALIZE |
| 34 | OPEN TEMPORARY |

### Examples

```
RSLT := CHECKGUARDFILE(PTRTOGUARDFILE);

IF REAL(CHECKGUARDFILE(USERGUARD)) IS REAL(NOT FALSE) THEN . . .
```

# CONTROLCARD Function

The CONTROLCARD function provides a mechanism for a process to initiate the Work Flow Language (WFL) compiler and to pass input records in the input originator.

```
── CONTROLCARD ── ( ── <input designator> ── , ──────────────────────→

→─ <arithmetic expression> ── ) ───────────────────────────────────────┤
```

<input designator>

```
──┬─ <nondirect file designator> ──────────────────────────┬────────────┤
  ├─ <queue designator> ───────────────────────────────────┤
  ├─ <one-dimensional real array identifier> ──────────────┤
  ├─ <one-dimensional real array reference identifier> ────┤
  ├─ <real array identifier> ── <row selector> ────────────┤
  └─ <real array reference identifier> ── <row selector> ──┘
```

<one-dimensional real array identifier>

An array identifier that is declared of type REAL and with one dimension.

<one-dimensional real array reference identifier>

An array reference identifier that is declared of type REAL and with one dimension.

<real array identifier>

An array identifier that is declared of type REAL.

<real array reference identifier>

An array reference identifier that is declared of type REAL.

### Explanation

The values and meanings associated with the arithmetic expression parameter are as follows:

| Field | Description |
|---|---|
| [46:01] | If messages from the calling process (for example, BOT, EOT, and DISPLAY) are passed to an MCS for display at the originating station, the job initiated by the call CONTROLCARD passes messages to the MCS. To suppress this feature, set this field to TRUE. |
| [44:01] | This is a START command (if TRUE). |
| [39:01] | If TRUE, causes error messages to be written to the file ERRORFILE. By default this file has a KIND of REMOTE, causing the messages to be written back to the appropriate terminal, provided that the STATION task attribute is nonzero. If the STATION task attribute is zero, the KIND must be equated to a value other than REMOTE. |
| [38:01] | This bit is applicable only to request type 4. If TRUE, the WFL job runs as a dependent task before the controlcard function returns control to the originating task. If FALSE, the WFL job is initiated as an independent process. Note that if the WFL job runs as a dependent task, references to MYJOB and inheritance of attributes normally inherited from a job (for example, WAITLIMIT and PRINTLIMIT) might behave differently, since they refer to the parent job of the task that called the controlcard function. |
| [37:01] | If TRUE, the job number of the compiled job and the disposition of the job are returned in the result of the CONTROLCARD function, unless bit 38 is also TRUE. |
| [34:01] | If TRUE, this job is compiled for syntax checking only. This field overrides the disposition in the job. |
| [15:08] | This field contains the character to be used as the invalid character. If the field is 4" 00" , a question mark (?) is assumed. |
| [07:08] | Request type 3 is ZIPped WITH file. Request type 4 is ZIPped WITH array. Request type 7 is ZIPped WITH queue. |

If request type 7 is used, control does not pass from the function until the following is encountered as an insert in the queue:

```
<invalid character> END JOB.
```

For this reason, a procedure that contains the function should be processed asynchronously.

Results are returned in the Boolean function value and in the TASKVALUE task attribute of the process calling the CONTROLCARD function.

The job number of the job compiled by WFL is returned in bits [24:24] of the TASKVALUE. If the WFL compilation had syntax errors, bit 0 of the TASKVALUE is also set.

Bits 37 and 38 of the arithmetic expression passed to CONTROLCARD determine the layout of the value returned by the function.

If bit 37 is set and bit 38 is not set, the following field values are returned:

| Field | Description |
|---|---|
| [46:01] | Set if a failure occurred that prevented a job from being compiled (for example, the file specified for a type 3 request did not have a FILEKIND of JOBSYMBOL). If this bit is set, the remainder of the result determines the type of failure. |
| [24:24] | The job number associated with the job that was compiled as a result of the CONTROLCARD call. The job number is present even if the compilation had syntax errors. |
| [00:01] | Set if the WFL compilation has syntax errors. |

If bit 38 is set or bit 37 is not set, the following values are returned:

| Value | Description |
|---|---|
| 2 | Bit 38 was set and the job was successfully compiled and initiated as a co-routine. |
| Other | An error that prevented the job from being compiled occurred (the value is zero if a syntax error occurred, and bit 0 of the taskvalue attribute is set). |

***Note:*** *Some WFL jobs and constructs cause the WFL compiler to change and restore the usercode under which the calling process is running. In some situations the usercode privileges cannot be restored accurately. If the WFL job that is being compiled is a single-statement START command, or contains a USER statement in the job heading, it is appropriate for the calling process to initiate a separate task to call CONTROLCARD.*

## WFL Card Image (Variant = 1)

The message is passed to the WFL compiler with a DCALGOL queue that has been used as the input designator in a CONTROLCARD function that uses the following construct:

```
INSERT(<message designator>, <queue designator>);
```

The format of this message is as follows:

| Word | Field | Value | Description |
|---|---|---|---|
| 0 | [47:08] | 21 | Type. |
| | [39:08] | 1 | Variant. |
| | [23:24] | | Logical Station Number (LSN) of the remote terminal. |
| 1 | | | Number of card images in the message. |
| 2 to end | | | Card image text in EBCDIC with each card image preceded by an 8-bit length byte. Bit [07:01] of this length byte is turned on if the first character of the card is invalid and if field [06:07] of the byte contains the length of the card image in units of characters. |

### Examples

```
CONTROLCARD(ARAY[*],4 & 1 [39:1] & 1 [38:1]);

CONTROLCARD(AFILE,3 & COMPOK [38:6:1]);

B := CONTROLCARD(QUEUETOG,NUMBER);
```

# DCERRANALYSIS Function

The DCERRANALYSIS function allows an MCS to interpret error result messages in the form of English text. This function can be used only in a data comm environment that is initialized at the time of the use of this function. Only an MCS that previously initialized a primary queue can use this function. The reason for this restriction is that the DCERRANALYSIS function uses an INTERROGATE STATION ENVIRONMENT DCWRITE.

```
── DCERRANALYSIS ── ( ── <single-precision array row> ── , ──────────→

→─ <pointer expression> ── , ── <Boolean expression> ── ) ──────────┤
```

<single-precision array row>

An array row whose identifier is declared to be of type INTEGER or REAL.

### Explanation

The first parameter is a single-precision array row that contains the error result message (six words). The second parameter is a pointer into an array where the text that results from the analysis of the error result message can be stored. The third parameter is a Boolean expression that indicates the request for complete analysis of the error result message (additional information such as NDLII flags and the original DCWRITE type). The Boolean result returned by the DCERRANALYSIS function contains the following information:

| Field | Description |
|---|---|
| [23:16] | Character count field |
| [03:01] | Error message invalid |
| [02:01] | Error message array too small |
| [01:01] | Text array too small |
| [00:01] | Analysis unsuccessful |

A successful analysis of an error result message returns a result with bit 0 equal to FALSE and the length of the text (in number of characters) stored in the character count field.

If the function is unable to successfully analyze an error result message (bit 0 equals TRUE in the result), the reason is given in bits 1 through 3. Bit 1 is TRUE if the array referenced by the pointer expression was too small to hold the analyzed text. Bit 2 is TRUE if the single-precision array row is less than six words long. Bit 3 is TRUE if the content of the error result message was contradictory or did not make sense (in this case, a brief indication of the nature of the difficulty encountered can still be stored as text and its length indicated by the character count field).

The text returned is divided into four parts in the following order:

1. The time of day the error occurred according to the data communications controller (DCC) timestamp.
2. The station name followed by the logical station number (LSN) in parentheses.
3. A description and diagnosis of the error.
4. Additional information, if the value of the Boolean expression is TRUE.

Contiguous entities in parts 3 and 4 are separated by semicolons.

An example of an error result message is as follows:

```
630000 00000A 010004 100010 000000 000000 380ED4 000000
000000 210000 000000 000000
```

Analysis of this error result message yields the following:

```
17:00:30 TD1314KV(10) TERMINATE ERROR: LASTFLAG=DISCONNECT,
FLAGS=STATION/LINE NOT RDY;
```

If a complete analysis is requested, the text returned would be as follows:

17:00:30 TD1314KV(10) TERMINATE ERROR: LASTFLAG=DISCONNECT,
FLAGS=STATION/LINE NOT RDY; LINE:NOT BUSY,READY, TOGS: 0=0 1=0;
DCWRITE TYPE=ENABLE INPUT ON STATION

### Example

```
IF NOT RSLT := DCERRANALYSIS(ERRMSG,PTR,FALSE) THEN . . .
```

# DCERRORLOGGER Function

The DCERRORLOGGER function allows an MCS to log data comm errors.

```
── DCERRORLOGGER ── ( ── <message group designator> ── , ── <size> ───→

→─ ) ─────────────────────────────────────────────────────────────────┤
```

### Explanation

The contents designated by the <message group designator> construct are entered in the system log for the number of words designated by the <size> parameter, prefixed by a 4-word header. If the <size> parameter does not yield an integer value, it is rounded to an integer value. The MAJOR type assigned to this entry is 5, which is the network support processor (NSP) maintenance record. The MINOR type has the value 4, which is the MCS result message record.

If the given log record cannot be entered in the system log, the DCERRORLOGGER function returns a negative result indicating the specific reason that the request was denied. The values returned by the DCERRORLOGGER function are as follows:

| Value | Description |
|---|---|
| 0 | The log entry was made successfully. |
| -1 | The <size> parameter was greater than the true size of the message group designated. |
| -2 | The caller is not a valid MCS or has not initiated the primary queue. |
| -3 | A disk error occurred during an attempt to log the entry. |

### Examples

```
R := DCERRORLOGGER(MA[*],6);

DCERRORLOGGER(MA,4);
```

# DCSYSTEMTABLES Function

The DCSYSTEMTABLES function enables an ALGOL or DCALGOL program to obtain information about the current data comm environment. The DCSYSTEMTABLES function has a real value.

```
── DCSYSTEMTABLES ── ( ── <arithmetic expression> ── , ───────────────→

→─ <noncharacter array row> ── ) ──────────────────────────────────────┤
```

### Explanation

The information selected by the arithmetic expression is copied into the noncharacter array row. If the value of the arithmetic expression is 3, 4, or 7, then the noncharacter array row is resized to contain the requested information; the previous contents of the noncharacter array row are lost. For all other values of arithmetic expression, the noncharacter array row is resized only if the size of the array is insufficient for the information to be returned.

The information selected when the value of the arithmetic expression is 0, 1, 2, or 4 is intended for use only by system software, and is subject to change.

The noncharacter array row is referred to in the following tables as NCAR. The information selected by the arithmetic expression is as follows:

| Value | Description |
|---|---|
| 0 | DCC tables. |
| 1 | NSP tables. |
| 2 | DCC station table |
| 3 | General information:<br>NCAR[0]:<br>Specifies the bit mask of initialized NSPs. For example, if NCAR[0].[3:1]=1, then relative NSP 3 has been initialized.<br>NCAR[1] through NCAR[3]:<br>Contains, in display form, the DATACOMINFO file prefix (including familyname) terminated by a period (.) The DATACOMINFO file prefix is truncated if it is greater than 18 characters in length.  See NCAR[6] through NCAR[10] for additional information.<br>NCAR[4]:<br>Specifies the bit mask of online NSPs.<br>NCAR[5]:<br>Specifies the bit mask of NSPs configured in the DATACOMINFO file. |

| Value | Description |
|---|---|
| | NCAR[6] through NCAR[10]: |
| | Contains, in display form, the DATACOMINFO file prefix (including familyname) terminated by a period (.). |
| | ***Note:*** *As indicated, the DATACOMINFO file prefix contained in NCAR[1] through NCAR[3] is truncated if it is greater than 18 characters in length. The DATACOMINFO file prefix contained in NCAR[6] through NCAR[10] is not truncated under any circumstances.* |
| 4 | DATACOMINFO |
| | File station record |
| 6 | Provides interpretive text for DCWRITE errors. NCAR[0] contains the DCWRITE error number as an integer. On return, the function value is 0 if no errors occurred. The first word of the array contains the text length in characters; the second and subsequent words contain an EBCDIC text string, followed by a null character. If the NCAR contains fewer than 20 words, it is resized. |
| 7 | If the first word of the array parameter contains 0: Provides the NSP and line support processor (LSP) hardware unit numbers for the entire network. The format of each word of the returned array is as follows: |
| | [47:16] |
| | The NSP hardware unit number. |
| | [31:16] |
| | The LSP hardware unit number. |
| | [15:01] |
| | If 1, and automatic initialization of the NSP is required, the operating system automatically initializes the NSP. If 0, the operating system has encountered an error while communicating with the NSP, and the NSP is not automatically initialized. |
| | [14:07] |
| | The relative number of the NSP. |
| | [07:04] |
| | The relative number of the LSP. |
| | [03:04] |
| | Always 0. |

| Value | Description |
|---|---|
| | If the first word of the array parameter is other than 0:<br>Provides the NSP and LSP hardware unit numbers for a specific DLS number designated by the value in the first word of the array parameter. The DLS number is made up of three components, which are separated by colons (:). The first component is the relative network support processor number (previously known as the data communications processor number). The second component is the line number. The third component is the relative station number.<br>The array row is resized, if necessary, to hold the information for all the LSPs on the network (one word per LSP). The format of the first word of the returned array row is as follows<br>[47:16]<br>The NSP hardware unit number.<br>[31:16]<br>The LSP hardware unit number, unless the relative LSP number submitted in the DLS number does not correspond to an LSP on that NSP, in which case this field is 0.<br>[15:16]<br>The DL part of the DLS number as given in the request (in field [23:16]). If the NSP is not available, the DLS bit (in field [15:01]) is changed to 0. |

When information is successfully obtained, the DCSYSTEMTABLES function returns a real value in the following format:

| Field | Description |
|---|---|
| [39:20] | Size (in words) of the NCAR parameter after it is resized by the DCSYSTEMTABLES function if applicable; otherwise, 0. |
| [19:20] | Memory address of the table specified by the <arithmetic expression> parameter if applicable; otherwise, 0. |

If the information is not successfully obtained, the value returned by the function is negative; that is, it contains a 1 in bit 46 and one of the following values in field [19:20]. Each value indicates a reason the information was not obtained.

| Value | Error Description |
|---|---|
| 1 | Data comm is not running, and the arithmetic expression is not equal to 3 or 6. |
| 2 | The value of the arithmetic expression is not valid input to the function. |
| 3 | An invalid LSN occurs in NCAR[0] (arithmetic expression equal to 4). |
| 4 | An invalid name occurs in NCAR (arithmetic expression equal to 4). |
| 5 | An unknown station exists (arithmetic expression equal to 4). |
| 6 | The relative NSP number is invalid. |
| 7 | No line information is available. |
| 8 | An invalid DLS number is designated. |
| 11 | A fault occurred while processing the request. |
| 13 | The information requested requires an array that is too large to be resized. The required size of the array is returned in field [39:20]. The request can be retried by using a segmented array of the indicated size for the NCAR parameter. |
| 20 | A disk error occurred while accessing the data comm files. |

When the arithmetic expression is equal to 4, you must designate the desired station either by placing the station name in EBCDIC starting in NCAR[0] or by designating the station LSN in NCAR[0].

### Example

```
T := DCSYSTEMTABLES(3,A[I,*]);
```

# DCWRITE Function

The DCWRITE function causes the message specified by the message designator to be passed to the data communications controller (DCC). The action taken by the DCC depends on the type and variant fields of the message. For detailed information, refer to the “DCWRITE Information” section.

```
── DCWRITE ── ( ── <message designator> ─┬──────────────────────────┬─→
                                         └─ , ── <queue designator> ─┘

→─ ) ──────────────────────────────────────────────────────────────────┤
```

### Explanation

The queue designator is required for certain DCC actions. The value returned is an error identification number. A zero indicates no error; other error values are given in the “DCWRITE Information” section.

A stack that is not marked as an MCS can call the DCWRITE function, provided that the procedure that calls the DCWRITE function is declared in a stack marked as an MCS.

### Examples

```
ERRORNO := DCWRITE(MESSAGEID);

ERRORNO := DCWRITE(MESSAGEARRAYID[3],QUEUEID);
```

# DISPLAYTOSTANDARD Function

The DISPLAYTOSTANDARD function is a Boolean function that converts a display form file construct to standard form.

```
── DISPLAYTOSTANDARD ─ ( ── <display location> ─ , ── <standard location>─→

→─ ) ──────────────────────────────────────────────────────────────────────┤
```

```
<display location>

── <pointer expression> ───────────────────────────────────────────────────┤
```

### Explanation

A display form file construct can be a file name, file title, file directory, directory name, or a directory title. Refer to the section in the *System Command Operations Reference Manual* titled "Basic Constructs" for the complete file construct syntax.

### Example

In the following example, "(SMITH)REPORT/JULY ON ACCOUNTS" is a file title that contains a usercode, two identifiers separated by a slash, and an ON <familyname> clause.

```
  "(SMITH)REPORT/JULY ON ACCOUNTS"
```

A standard form file construct, used internally by the operating system, has the following form:

```
── <total length> ── <qualification> ── <number of identifiers> ─────→

  ┌←─────────── /13\ ──────────┐
→─┴─ <id length> ── <identifier> ─┴──────────────────────────────────┤
```

The <total length> parameter is an 8-bit binary number that represents the total length in bytes of the standard form file construct and includes the <total length> parameter byte itself.

A qualification is a formatted, 8-bit byte that contains the following fields:

| Field | Value | Description |
|---|---|---|
| [06:01] | | A name in the display form file construct  was originally surrounded by quotation marks because it contained hyphens or underscores. |
| [02:01] | | Family name included. If this field is 1, the last identifier represents the familyname in the ON <familyname> clause. If this field is 0, no familyname is present. |
| [01:02] | | Usercode information. This field contains information about the presence of an asterisk (*) or a usercode. The possible values returned have the following meanings: |
| | 1 | The display form file construct is not preceded by an asterisk or usercode. |
| | 2 | The display form file construct is preceded by an asterisk. |
| | 3 | The display form file construct is preceded by a usercode. |

The <number of identifiers> parameter is an 8-bit binary number that represents the total number of identifiers in the file construct.

The <id length> parameter specifies the length, in bytes, of the identifier that immediately follows it.

The identifier is a usercode, a familyname, or one level of the file construct. In the file title (SMITH)REPORT/JULY ON ACCOUNTS, four identifiers are included: SMITH, REPORT, JULY, and ACCOUNTS. The DISPLAYTOSTANDARD function disallows blanks embedded within an identifier and ignores nonembedded blanks. A familyname must contain only uppercase alphanumeric characters. An identifier that represents one level of the file construct or a usercode can include hyphens and underscores without quotes. If other nonalphanumeric characters or lowercase characters are included, quotes are required. All identifiers and usercodes that are longer than 17 characters are truncated. If truncation occurs, the system issues a warning message and sets the warning bit [11:1].

A display form file construct must end with a period (.).

The result space in the array that follows the destination pointer and contains the resulting file construct must be at least 255 characters long.

You must assign the compiler control option INSTALLATION 1 to allow the compiler to access the required installation intrinsic.

The DISPLAYTOSTANDARD function returns a Boolean result that is FALSE if no errors occur during the conversion. Otherwise, it returns the Boolean result TRUE. If TRUE, bit 0 of the result is 1, and one of the following bits also has the value 1 to identify the error:

| Bit | Description |
|---|---|
| 47 | A fault occurred while scanning the display form file construct. |
| 10 | A name node was expected between slashes, or either a name or an equal sign was expected following the final slash. |
| 9 | More than 12 nodes occurred in the name portion of the file name. |
| 8 | A slash was expected between successive identifiers. |
| 7 | A usercode was expected after the left parenthesis. |
| 6 | No identifiers were found. |
| 5 | A nonalphanumeric character was found in the familyname. |
| 4 | The file construct did not terminate with a period(.). |
| 3 | An identifier that contains a nonalphanumeric character was not enclosed in quotation marks. |
| 2 | A null quoted string is illegal as an identifier. |
| 1 | A right parenthesis was expected after the usercode. |

***Note:*** *If the node name has more than 17 characters, only bit 11 is set to 1 and the node name is truncated to 17 characters.  This is not an error result.*

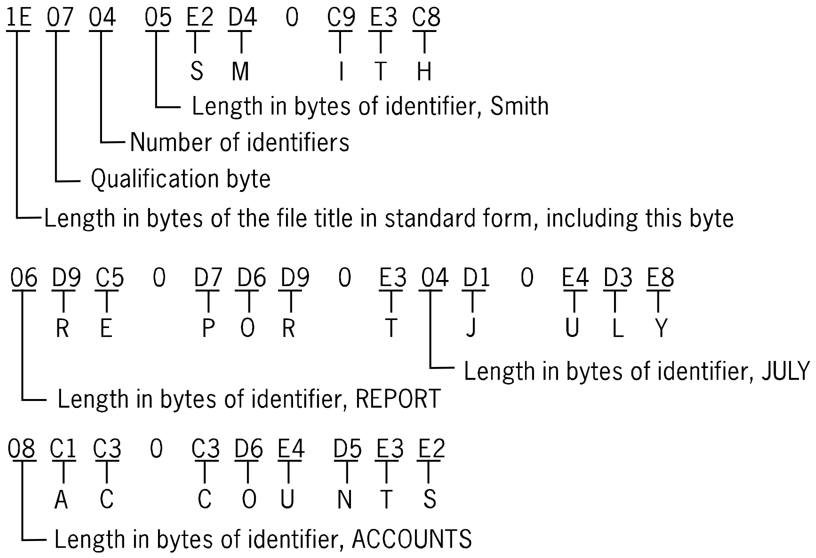
### Examples

The following example converts the file title (SMITH)REPORT/JULY ON ACCOUNTS to the standard form.

```
$SET INSTALLATION 1

. . .

REPLACE PTRDISPLAY BY "(SMITH)REPORT/JULY ON ACCOUNTS.";
ERRORFLAG := DISPLAYTOSTANDARD(PTRDISPLAY,PTRSTANDARD);
```


1E 07 04 05 E2 D4 0 C9 E3 C8
S M I T H
Length in bytes of identifier, Smith
Number of identifiers
Qualification byte
Length in bytes of the file title in standard form, including this byte
06 D9 C5 0 D7 D6 D9 0 E3 04 D1 0 E4 D3 E8
R E P O R T J U L Y
Length in bytes of identifier, JULY
Length in bytes of identifier, REPORT
08 C1 C3 0 C3 D6 E4 D5 E3 E2
A C C O U N T S
Length in bytes of identifier, ACCOUNTS

# GETSTATUS Function

The GETSTATUS function gathers designated system information and places it in the noncharacter array row.

```
── GETSTATUS ── ( ── <arithmetic expression> ── , ─────────────────────→

→─ <arithmetic expression> ── , ── <arithmetic expression> ── , ───────→

→─ <noncharacter array row> ── ) ──────────────────────────────────────┤
```

### Explanation

The information that you supply through four parameters designates the information that is to be returned by the program. The parameters, their values, and the information returned are described in the *GETSTATUS/ SETSTATUS Programming Reference Manual.*

### Example

```
BOOL := GETSTATUS(TYPE,SUBCLASS,MASK,ARRAYROW);
```

# INSERT_CLIENTQUEUE Function

The INSERT_CLIENTQUEUE function and the LINKLIB_CLIENTQUEUE function in conjunction can be used in place of two-way connection library linkages to eliminate the overhead of circular stack search graphs.

```
── INSERT_CLIENTQUEUE ── ( ──<noncharacter array row>── , ──<use size>─→

→─ , ─┬─<connection library instance designator>─┬─ ) ────────────────┤
      ├─<this intrinsic>─────────────────────────┤
      └─<connection parameter>───────────────────┘
```

### Explanation

The first two parameters specify the data, which is to be sent in a message to the client of the connection library instance specified by the third parameter.

The first two parameters have syntax and semantics identical to the first two parameters of the array row variant of the DCALGOL INSERT statement.

In the third parameter, <this intrinsic> refers to a connection library instance from within the connection control block, and <connection parameter> refers to a connection library instance through the use of a CONNECTION reference variable.

The following table displays the procedure values that are returned:

| Syntax Element | Value | Description |
|---|---|---|
| CLIENTQ_SUCCESS | 1 | The operation was successful. |
| CLIENTQ_NOQUEUE | 2 | The relevant library linkage was not established by a successful invocation of LINKLIB_CLIENTQUEUE. |
| CLIENTQ_QUEUEGONE | 3 | The queue supplied to LINKLIB_CLIENTQUEUE when the relevant library linkage was established has been deallocated, or it has been the subject of a RETURN_QUEUE_TOKEN invocation. |
| CLIENTQ_MSGINVALID | 4 | The length of the array row is less than the specified size, or the specified size is outside the permitted range of values. |

# LINKLIB_CLIENTQUEUE Function

The LINKLIB_CLIENTQUEUE function and the INSERT_CLIENTQUEUE function in conjunction can be used in place of two-way connection library linkages to eliminate the overhead of circular stack search graphs.

```
── LINKLIB_CLIENTQUEUE ── ( ──<queue designator>── , ──────────────────→

→─┬─<library identifier>──────────────┬─┬─<indirect linkage>───────┬──→
  ├─<connection library identifier>───┤ ├─<process family linkage>─┤
  ├─< connection library designator >─┤ └─<direct linkage>─────────┘
  ├─<this intrinsic>──────────────────┤
  └─<connection parameter>────────────┘

→─ ) ──────────────────────────────────────────────────────────────────┤
```

The queue designator specifies a queue by which the linked library might send messages to this client. The linked library must be a connection library.

The syntax and semantics of the remaining parameters are documented in the LINKLIBRARY description in the *ALGOL Programming Reference Manual, Volume 1: Basic Implementation*.

The values returned by the LINKLIB_CLIENTQUEUE function are the same as those returned by LINKLIBRARY, except for two additions and one semantic modification:

| Value | Description |
|---|---|
| 12 | Successful linkage was made to the library, but not all entry points were provided. Also, the queue specified by queue designator cannot be associated with the connection to the linked library, either because the linked library is not a connection library, or because of MCP resource limitations. |
| 11 | Successful linkage was made to the library, and all entry points were provided. Also, the queue specified by the queue designator cannot be associated with connection to the linked library, either because the linked library is not a connection library, or because of MCP resource limitations. |
| 0 | The program was already linked to the specified library at the time of the LINKLIBRARY call. Also, the queue specified by the queue designator was not associated with the connection to the linked library. |

The program, library, procedure, or block that declared the queue can terminate at any time without causing problems for the MCP. The MCP simply stops inserting INSERT_CLIENTQUEUE messages in the queue.

RETURN_QUEUE_TOKEN must not be called on the queue while messages are being inserted by INSERT_CLIENTQUEUE. (INSERT_CLIENTQUEUE starts to return the CLIENT_QUEUEGONE result.)

LINKLIB_CLIENTQUEUE leaves the queue with an anonymous token assigned to it: future invocations of GET_QUEUE_TOKEN on that queue return TOKEN_ALREADYDONE. (TOKEN_ALREADYDONE=2.)

For further information on RETURN_QUEUE_TOKEN, GET_QUEUE_TOKEN, and TOKEN_ALREADYDONE, refer to the *MCP Systems Interfaces Programming Reference Manual.*

The following examples illustrate the use of LINKLIB_CLIENTQUEUE and INSERT_CLIENTQUEUE to establish two-way communication between a library and its clients without the overhead associated with two-way connection libraries.

## Examples

LIB is an EXPORTING connection library, which acts as a server. Two client programs are also part of this example. CLIENT1 connects to LIB using traditional server library syntax and CLIENT2 connects to LIB using an IMPORTING connection library.

LIB exports multiple connections, one for each client. For example, if three CLIENT1 tasks and four CLIENT2 tasks are running, LIB has seven active connections.

LIB also illustrates the use of ISOLATED to prevent the DS of a client from interrupting execution of a LIB procedure that uses global locks.

### TEST/CLIENTQUEUE/LIB

```
BEGIN
  %----------------------------------------------------------------------
  % Declaration of some simple (and not very efficient) structures
  % to track which connections are in use.  This structure is
  % deliberately designed to require locking, to illustrate the use of
  % ISOLATED to protect the integrity of such a structure.

  INTEGER
      CLIENTS_ACTIVE;
  ARRAY
      CLIENTS_LIST[0:10];
  EVENT
      CLIENTS_INFO_LOCK;
  %----------------------------------------------------------------------
  TYPE CONNECTION BLOCK CB;
  BEGIN
      INTEGER
          CONNECTION_INDEX,
          NREQUEST,
          NRESPONSE;
      BOOLEAN PROCEDURE ADDCLIENT;
      BEGIN           [ ISOLATED ]
          %
          % This procedure uses ISOLATED to insure the integrety
          % of the structures protected by CLIENTS_INFO_LOCK.
          % If the stack running this procedure is DSED, ISOLATED
          % insures that either this procedure completes its work
          % or the entire library is terminated.
          PROCURE(CLIENTS_INFO_LOCK);
          CLIENTS_ACTIVE := *+1;
```

```
    IF SIZE(CLIENTS_LIST) LSS CLIENTS_ACTIVE THEN
        RESIZE(CLIENTS_LIST,CLIENTS_ACTIVE+50,RETAIN);
    CLIENTS_LIST[CLIENTS_ACTIVE-1] := CONNECTION_INDEX;
    LIBERATE(CLIENTS_INFO_LOCK);
END ADDCLIENT;
PROCEDURE REQUEST(A);
ARRAY A[0];
BEGIN
    % This procedure is to be invoked by the client when
    % it wants the library to do something, passing the
    % relevant data in this procedure's parameter(s).
    % For this example library, a message is displayed,
    % including text from the parameter.
    EBCDIC ARRAY
        MSG[0:200];
    NREQUEST := *+1;
    REPLACE MSG[0] BY
     "REQUEST ", NREQUEST FOR * DIGITS,
     " from connection ", CONNECTION_INDEX for * DIGITS,
     "  ", POINTER(A) FOR MIN(30,SIZE(A)*6) UNTIL EQL 48"00";
    DISPLAY(MSG[0])
END REQUEST;
PROCEDURE REMOVECLIENT;
BEGIN                  [ ISOLATED ]
    INTEGER
        MYINX,
        I;
    % This procedure uses ISOLATED to insure the integrety
    % of the structures protected by CLIENTS_INFO_LOCK.
    % If the stack running this procedure is DSED, ISOLATED
    % insures that either this procedure completes its work
    % or the entire library is terminated.
    %
    PROCURE(CLIENTS_INFO_LOCK);
    MYINX := -1;
    FOR I := 0 STEP 1 UNTIL CLIENTS_ACTIVE-1 DO
    BEGIN
        IF CLIENTS_LIST[I] EQL CONNECTION_INDEX THEN
            MYINX := I;
    END;
    IF MYINX GEQ 0 THEN
    BEGIN
        CLIENTS_ACTIVE := *-1;
        I := MYINX;
        WHILE I LSS CLIENTS_ACTIVE DO
        BEGIN
            CLIENTS_LIST[I] := CLIENTS_LIST[I+1];
            I := *+1;
        END;
    END;
    LIBERATE(CLIENTS_INFO_LOCK);
END REMOVECLIENT;
```

```
    PROCEDURE CB_CHANGE(CONN_INDEX, NEW_STATE, REASON, ACTOR, IMDSED);
    VALUE   CONN_INDEX, NEW_STATE, REASON, IMDSED;
    INTEGER CONN_INDEX, NEW_STATE, REASON;
    TASK    ACTOR;
    BOOLEAN IMDSED;
    BEGIN
        IF NEW_STATE EQL VALUE(LINKING) THEN
        BEGIN
            CONNECTION_INDEX := CONN_INDEX;
            NREQUEST := 0;
            NRESPONSE := 0;
            ADDCLIENT;
        END;
        IF NEW_STATE EQL VALUE(NOTLINKED) THEN
            REMOVECLIENT;
    END CB_CHANGE;
    EXPORT
        REQUEST;
END CB;
CB EXPORTING LIBRARY
    CL(INTERFACENAME = "INT.",
       CHANGE = CB_CHANGE);
%---------------------------------------------------------------------
REAL
    RSLT,
    SEED;
INTEGER
    I,
    CL_INX;
ARRAY
    A[0:100];
SEED := 123456789+TIME(11);
RSLT := READYCL(CL);
IF RSLT NEQ 0 THEN
BEGIN
    DISPLAY("READYCL failed, rslt = " CAT STRING(RSLT,*));
    MYSELF.STATUS := VALUE(TERMINATED);
END;
% A real library would presumably do something useful at this point.
%
% For this example, the library simply tells random clients
% to call GETRESPONSE, at random intervals.
WHILE TRUE DO
BEGIN
    WHEN(RANDOM(SEED)*3);         % Wait 1.5 seconds on average.
    PROCURE(CLIENTS_INFO_LOCK);
    IF CLIENTS_ACTIVE GTR 0 THEN
    BEGIN
        I := MIN(INTEGERT(RANDOM(SEED)*CLIENTS_ACTIVE),
                 CLIENTS_ACTIVE-1);
        CL_INX := CLIENTS_LIST[I];
        CL[CL_INX].NRESPONSE := *+1;
```

```
          REPLACE POINTER(A) BY
           "RESPONSE ", CL[CL_INX].NRESPONSE FOR * DIGITS,
           " for connection ", CL_INX for * DIGITS,
           48"00";
          RSLT := INSERT_CLIENTQUEUE(A,10,CL[CL_INX]);
          IF RSLT NEQ 1 THEN
          BEGIN
              DISPLAY("INSERT_CLIENTQUEUE failed, rslt = " CAT
                      STRING(RSLT,*));
              MYSELF.STATUS := VALUE(TERMINATED);
          END;
      END;
      LIBERATE(CLIENTS_INFO_LOCK);
  END;
END.
```

### TEST/CLIENTQUEUE/CLIENT1

```
BEGIN

  % This client illustrates use of library TEST/CLIENTQUEUE/LIB.
  %
  % This program uses normal server library syntax to link
  % to the EXPORTING connection library in TEST/CLIENTQUEUE/LIB.
  %-----------------------------------------------------------------------
  LIBRARY
      CLIENTQLIB(LIBACCESS=BYTITLE,
                 INTERFACENAME = "INT.",
                 TITLE="OBJECT/TEST/CLIENTQUEUE/LIB.");
  PROCEDURE REQUEST(A);
  ARRAY A[0];
  LIBRARY CLIENTQLIB;
  %-----------------------------------------------------------------------
  DEFINE
      TICKSPERSECOND = INTEGER(1/(2.4@-6)) #;
  REAL
      RSLT,
      SEED,
      T14_DONE,
      T14_NEXTREQUEST;
  QUEUE
      CALLBACKQ;
  ARRAY
      A[0:100];
  INTEGER
      N;
  SEED := 123456789+TIME(11);
  RSLT := LINKLIB_CLIENTQUEUE(CALLBACKQ,CLIENTQLIB,WAITFORFILE);
  IF RSLT NEQ 1 THEN
  BEGIN
      DISPLAY("LINKLIBRARY failed, rslt = " CAT STRING(RSLT,*));
      MYSELF.STATUS := VALUE(TERMINATED);
  END;
  % A real program would presumably do something useful at this point.
  %
  % For this example, the program simply runs for about a
  % minute, occasionally calling REQUEST and displaying any
  % messages which appear in CALLBACKQ.
  T14_DONE := TIME(14)+(30+60*RANDOM(SEED))*TICKSPERSECOND;
  T14_NEXTREQUEST := TIME(14)+10*RANDOM(SEED)*TICKSPERSECOND;
  WHILE TIME(14) LSS T14_DONE DO
  BEGIN
      CASE WAIT((MAX(0.1,(T14_NEXTREQUEST-TIME(14))/TICKSPERSECOND)),
                CALLBACKQ.QINSERTEVENT) OF
      BEGIN
      1:
          REPLACE POINTER(A) BY
           "MIX = ", MYSELF.MIXNUMBER FOR * DIGITS, 48"00";
          REQUEST(A);
          T14_NEXTREQUEST := TIME(14)+10*RANDOM(SEED)*TICKSPERSECOND;
      2:
          N := REMOVE(A,CALLBACKQ);
          DISPLAY(POINTER(A));
      END;
  END;
END.
```

### TEST/CLIENTQUEUE/CLIENT2

```
BEGIN

  % This client illustrates use of library TEST/CLIENTQUEUE/LIB.
  %
  % This program uses IMPORTING connection library syntax to link
  % to the EXPORTING connection library in TEST/CLIENTQUEUE/LIB.
  %--------------------------------------------------------------------
  TYPE CONNECTION BLOCK CB;
  BEGIN
      PROCEDURE REQUEST(A);
      ARRAY A[0];
      IMPORTED;
  END CB;
  CB SINGLE IMPORTING LIBRARY
      CLIENTQLIB(LIBACCESS=BYTITLE,
                 INTERFACENAME = "INT.",
                 TITLE="OBJECT/TEST/CLIENTQUEUE/LIB.");
  %--------------------------------------------------------------------
  DEFINE
      TICKSPERSECOND = INTEGER(1/(2.4@-6)) #;
  REAL
      RSLT,
      SEED,
      T14_DONE,
      T14_NEXTREQUEST;
  QUEUE
      CALLBACKQ;
  ARRAY
      A[0:100];
  INTEGER
      N;
  SEED := 123456789+TIME(11);
  RSLT := LINKLIB_CLIENTQUEUE(CALLBACKQ,CLIENTQLIB,WAITFORFILE);
  IF RSLT NEQ 1 THEN
  BEGIN
      DISPLAY("LINKLIBRARY failed, rslt = " CAT STRING(RSLT,*));
      MYSELF.STATUS := VALUE(TERMINATED);
  END;
  % A real program would presumably do something useful at this point.
  %
  % For this example, the program simply runs for about a
  % minute, occasionally calling REQUEST and displaying any
  % messages which appear in CALLBACKQ.
  T14_DONE := TIME(14)+(30+60*RANDOM(SEED))*TICKSPERSECOND;
  T14_NEXTREQUEST := TIME(14)+10*RANDOM(SEED)*TICKSPERSECOND;
  WHILE TIME(14) LSS T14_DONE DO
  BEGIN
      CASE WAIT((MAX(0.1,(T14_NEXTREQUEST-TIME(14))/TICKSPERSECOND)),
                CALLBACKQ.QINSERTEVENT) OF
      BEGIN
      1:
          REPLACE POINTER(A) BY
           "MIX = ", MYSELF.MIXNUMBER FOR * DIGITS, 48"00";
          CLIENTQLIB.REQUEST(A);
          T14_NEXTREQUEST := TIME(14)+10*RANDOM(SEED)*TICKSPERSECOND;
      2:
          N := REMOVE(A,CALLBACKQ);
          DISPLAY(POINTER(A));
      END;
  END;
END.
```

# MAKEUSERCODE Function

The MAKEUSERCODE function uses the data in the noncharacter array row parameter to create a usercode or to create a usercode and a password. The Boolean value returned by the MAKEUSERCODE function is TRUE if the designated usercode (and password, if designated) is created; otherwise, it is FALSE.

If a security administrator is authorized at your site, only a process running under the security administrator usercode can use the MAKEUSERCODE function; otherwise, only a process with privileged-user status can use this function.

You cannot use the MAKEUSERCODE function on a password-generating system.

```
── MAKEUSERCODE ── ( ── <noncharacter array row> ── ) ──────────────┤
```

## Explanation

The data in the noncharacter array row must have the following structure:

- The data must consist of an EBCDIC character string.
- The data must be in one of the three following forms:

  ```
  <usercode>
  <usercode> / <password>
  (<usercode>)<password>
  ```

- The data must terminate with a period.
- Leading blanks, trailing blanks, and blanks separating pairs of items in the case of usercode and password are acceptable.
- The usercode and password can be 17 or fewer characters, which exclude quotation marks when the usercode or password is enclosed in quotation marks.
- Blanks are significant inside a usercode or a password that is enclosed in quotation marks.
- Any of the 256 EBCDIC characters can be used (except the quotation marks) inside a usercode or password that is enclosed in quotation marks.
- If the usercode is ELEPHANT and the password is BIG-ONE, then any one of the following REPLACE statements is appropriate to fill the noncharacter array row:

  ```
  P := POINTER (ARRAYID);
  REPLACE P BY " ELEPHANT/" ," " BIG-ONE" ," " ." ;
  REPLACE P BY " ELEPHANT/" " " BIG-ONE" " " ." ;
  REPLACE P BY " ELEPHANT/ " " " BIG-ONE" " " ." ;
  ```

## Examples

```
IF MAKEUSERCODE(A[*]) THEN GO TO NEXTONE;

MAKEUSERCODE(ARRAYID);
```

# MCSLOGGER Function

The MCSLOGGER function allows an MCS to make entries into the system log primarily for the purpose of billing and monitoring station users. The noncharacter array row contains the information to be logged. The array format is described in the "Understanding the System Log" section in the *System Log Programming Reference Manual*.

```
── MCSLOGGER ── ( ── <noncharacter array row> ── ) ─────────────────┤
```

### Explanation

With the exception of log-on entries, the contents of the passed array are not modified and the first word of the noncharacter array row must contain a valid job number.

For log on, the MCSLOGGER function returns a unique job number in the first word of the noncharacter array row. This job number must be used for all future logging for this user.

If the given entry could not be entered in the log, the MCSLOGGER function returns a negative result to indicate the specific reason that the request was denied. The MCSLOGGER function returns the following values:

| Value | Description |
|---|---|
| 0 | The entry was logged successfully. |
| -1 | Either the noncharacter array row is too small to contain the information (length fields in link words were in error) or the information required more than 256 words. |
| -2 | The caller is not a valid MCS or has not initialized its primary queue. |
| -3 | A disk parity error occurred while the system was entering the records in the log. |
| -4 | Either the MAJOR type of entry was not 4 (MCS record), or the MINOR type was invalid (less than or equal to 0 or greater than 4). |
| -5 | The entry was not logged on, and the first word of the array did not contain a valid job number. |
| -6 | A job file cannot be made. |
| -7 | The fixed part of the log entry was too small for the entry type. |
| -8 | There were bad links pointing outside the variable part of the log entry. |
| -9 | Either the user code, access code, or charge code has a bad length or an illegal number of names. |

### Examples

```
MCSLOGGER(TEXT[A,B,*]);
RESULT := MCSLOGGER(JOBINFO[J,*]);
MCSLOGGER(BIGARRAY);
```

# NULL Function

The NULL function returns a Boolean value of TRUE if the message designator does not have any allocated area or if the queue designator has not been activated. Otherwise, it returns a value of FALSE.

```
── NULL ── ( ─┬─ <message designator> ─┬─ ) ───────────────────────────┤
              └─ <queue designator> ───┘
```

### Examples

```
BOOLEANID := NULL(MESSAGEID);

BOOLEANID := NULL(MESSAGEARRAYID[3]);

BOOLEANID := NULL(QUEUEARRAYID[3]);

BOOLEANID := NULL(QUEUEID);
```

# QUEUEINFO Function

The QUEUEINFO function returns an integer value in response to the information requested about the queue designator by the second parameter, <arithmetic expression>.

```
── QUEUEINFO ── ( ── <queue designator> ── , ──────────────────────────→

→─ <arithmetic expression> ── ) ──────────────────────────────────────┤
```

### Explanation

The values that are used in the arithmetic expression and the meanings of these values are as follows:

| Value | Description |
|---|---|
| -1 | Returns 0 and causes a memory dump. |
| 0 | Returns 0. |
| 1 | Returns the number of messages in the queue. |
| 2 | Returns the number of processes attached to the queue. |
| 3 | Returns the size, in words, of the message at the head of the queue. |
| 4 | Returns the total number of words being used by all the messages in the queue. |

All other values return -1.

A value of 3 (the size of the message at the head of the queue) should not be used to determine if truncation occurs with a remove to a noncharacter array row (or with any similar function) unless care is taken to ensure that no other processes could possibly remove that message between the time the size is checked and the time the message is removed.

### Examples

```
ANSWER := QUEUEINFO(QUEUEID,1);

ANSWER := QUEUEINFO(QUEUEARRAYID[3],NUMBER);
```

# REMOVE Function

The REMOVE function takes the message at the head of the queue, delinks it from that queue, and either inserts a descriptor that points to that message in the stack location referenced by the message designator or copies the message into the designated noncharacter array row. The value returned by the REMOVE function is the length of the message, in words.

Because the queue must be active to remove a message from it, be careful when you remove a message from a queue. If doubt exists, perform a test with the NULL function (refer to the “NULL Function” in this section). If you attempt to remove a message from an inactive queue, a run-time error occurs and INACTIVEQUEUE terminates the process unless an enabled ON statement exists for this fault.

```
── REMOVE ── ( ─┬─ <message designator> ─────┬─ , ────────────────────→
                └─ <noncharacter array row> ─┘

→─ <queue designator> ── ) ───────────────────────────────────────────┤
```

### Explanation

The REMOVE function delinks a message from the specified queue and causes the message to be either referenced by the message designator or copied into the noncharacter array row. If the latter occurs, the message area is returned to the system. A value of 0 is returned if no messages are in the queue. Otherwise, the length (in words) of the message removed is returned. If the message designator already references a message, that message area is returned to the system before the removal. If the noncharacter array row is smaller than the message removed, the message is truncated and the length of the noncharacter array row is returned. If the queue designator is not active, the program is terminated unless an ON statement has enabled the INACTIVEQUEUE interrupt.

### Examples

```
MESSAGESIZE := REMOVE(MESSAGEID,QUEUEID);

MESSAGESIZE := REMOVE(MESSAGEARRAYID[1],
                      QUEUEARRAYID[1]);

MESSAGESIZE := REMOVE(ARRAYID[*],QUEUEID);
```

# SETSTATUS Function

The SETSTATUS function controls a variety of operating system functions. The user is notified of invalid applications of the SETSTATUS function by error information coded into certain fields of the noncharacter array row. The required parameters, their values, and the information returned are described in the *GETSTATUS/ SETSTATUS Programming Reference Manual.*

```
── SETSTATUS ── ( ── <arithmetic expression> ── , ────────────────────→

→─ <arithmetic expression> ── , ── <arithmetic expression> ── , ──────→

→─ <noncharacter array row> ──────────────────────────────────────────┤
```

### Example

```
BOOL := SETSTATUS(TYPE,SUBTYPE,VAL,ARRAYROW);
```

# SETUPINTERCOM Function

The SETUPINTERCOM function allows an MCS to communicate with other MCSs or with the CONTROLLER. A message is sent to MCS number N by inserting a message in queue array reference identifier[N]. Received messages are found in the queue specified by the queue designator. The CONTROLLER implicitly has an MCS number of 0.

```
── SETUPINTERCOM ── ( ── <queue array reference identifier> ── , ──────→

→─ <queue designator> ─┬─────────────────────┬─ ) ──────────────────────┤
                       └─ , ── MLSCAPABLE ───┘
```

### Explanation

The SETUPINTERCOM function also allows an MCS to specify whether or not it is able to display messages sent by the operating system in multiple languages. Use the optional third parameter MLSCAPABLE to indicate this capability. “MLS” stands for multilingual system. The third parameter is a Type 2 reserved word.

If you include the MLSCAPABLE parameter, the operating system sends messages to the MCS in the format of message number and message parameters, if there are any. Because such a message is not in displayable form, the MCS should call the operating system procedure MCPMESSAGESEARCHER to format and translate the message, in the format described in “MESSAGE FROM CONTROLLER RESULT (Class=21)” in the “MCS Result Message Formats” section. For more information on the MultiLingual System, refer to the *MultiLingual System (MLS) Administration, Operations, and Programming Guide*. For further information regarding the MCPMESSAGESEARCHER procedure, refer to the *MCP System Interfaces Programming Reference Manual.*

If you do not include the MLSCAPABLE parameter, no special handling is done, and the operating system sends translated and formatted messages to the MCS as described in “MESSAGE FROM CONTROLLER RESULT (Class=21)” in the “MCS Result Message Formats” section.

If the request could not be completed, the SETUPINTERCOM function returns a negative result to indicate the specific reason for denying the request. The values returned by the SETUPINTERCOM function and their respective meanings are as follows:

| Value | Description |
|---|---|
| N>0 | The request completed without error. N is the MCS number of the program invoking this function. |
| -1 | The requesting program has not initialized its primary queue. |
| -2 | The <queue designator> parameter is already established as an intercom queue. |

The information that follows describes the interface message that uses the SETUPINTERCOM function to establish the communication link with the CONTROLLER.

You can cause a DCALGOL queue to pass to the operating system the designated message and the one that follows it by using the following construct:

```
INSERT(<message designator>,INTERCOMQUEUES[0]);
```

In the above construct, INTERCOMQUEUES is a queue array reference identifier. Before inserting a message in element 0 (the input queue of the CONTROLLER), you must establish the communication links by using the following DCALGOL construct:

```
SETUPINTERCOM(INTERCOMQUEUES,MYINPUTQ);
```

You can use one of two formats to send the message to the CONTROLLER:

- Format 1, which you use if you do not include the MLSCAPABLE parameter, or if you do not want to use the message attributes available in format 2. Format 1 is shown in Table 3–1.
- Format 2, which you use if you do include the MLSCAPABLE parameter, or if you want to use message attributes. Format 2 is shown in Table 3–2.

**Table 3–1. Format 1 for SETUPINTERCOM Function Messages**

| Word | Field | Value | Description |
|---|---|---|---|
| 0 | [47:08] | 21 | Type. |
| | [39:01] | 0 | Using format 1. |
| | [38:07] | | Variant field, as follows. Variants 7, 8, 9, and 10 are used only by Operator Display Terminal (ODT)-simulating MCSs: |
| | | 2 | Input request to the CONTROLLER for processing the message. |
| | | 4 | Request to the CONTROLLER to continue with the earlier request. Refer to Continuation Call later in this section. |
| | | 6 | Request to the CONTROLLER for connection to become an ODT-simulating MCS. Words 1, 2, 3, and 4 must have all bits equal to 1. |
| | | 7 | For an ODT-simulating MCS: request to the CONTROLLER for disconnection. Words 1, 2, 3, and 4 must have all bits equal to 1. |
| | | 8 | For an ODT-simulating MCS: requests to the CONTROLLER. The CONTROLLER discards all of the requests waiting for an answer from this dialogue and then proceeds as in Variant 2. |

**Table 3–1. Format 1 for SETUPINTERCOM Function Messages**

| Word | Field | Value | Description |
|---|---|---|---|
| | | 9 | For an ODT-simulating MCS: requests to the CONTROLLER. The CONTROLLER discards all of the requests waiting for an answer from this dialogue and then proceeds as in Variant 4. |
| | | 10 | For an ODT-simulating MCS: requests to the CONTROLLER. The CONTROLLER discards all of the requests waiting for an answer from this dialogue and then proceeds as in Variant 4. |
| | [31:08] | | Number of the originating MCS. |
| | [23:09] | | Used by the sending MCS for internal purposes. When a reply to this message is sent, this field is included in the reply. |
| | [14:15] | | LSN of the remote terminal. |
| 1 | [47:01] | 1 | If the reply is truncated at line width (see below). |
| | [46:01] | 0 | If the user is restricted to those commands marked FREE in the input to the table generation. |
| | | 1 | If the user can use any command. |
| | [45:01] | | ODT bit. When this bit is 1, the CONTROLLER handles the message as if it came from an ODT. |
| | [43:23] | | Dialogue number (for an ODT-simulating MCS). The dialogueCS number is untouched by the CONTROLLER and is sent back to the MCS for reference. |
| | [19:04] | 1 | No response to message. |
| | | 2 | Concise response, OK. |
| | | 3 | Expanded response. |
| | [15:08] | | Width of the line, where 2 < line width < 80 |
| | [07:08] | | Number of lines on a page, where 0 < lines <= 24 |
| 2-4 | | | Usercode. [2] . [47:08] contains the length of the usercode (in [2] . [46:06]) and a usercode check bit (in [2] . [47:01]). If the usercode check bit is set, the usercode is used to restrict mix-related commands, and for commands handled by external agents (such as Print System and DSS). If the usercode check bit is reset, the usercode is used to restrict PD commands, for WFL jobs, and for commands handled by external agents. If [2] . [47:08] equals 0, no usercode is used. |
| 5 | | | Length of input string in units of characters. |
| 6 to end | | | Input string. |

### Table 3–2. Format 2 for SETUPINTERCOM Function Messages

| Word | Field | Value | Description |
|---|---|---|---|
| 0 | [47:08] | 21 | Type. |
| | [39:01] | 1 | Using format 2. |
| | [38:07] | | Variant field, as follows (Variants 7, 8, 9, 10, 11, and 12 are used only by ODT-simulating MCSs): |
| | | 2 | Input request to the CONTROLLER for processing the message. |
| | | 4 | Request to the CONTROLLER to continue with the earlier request. Refer to Continuation Call, later in this section. |
| | | 6 | Request to the CONTROLLER for connection to become an ODT-simulating MCS. Words 1, 2, 3, and 4 must have all bits equal to 1. |
| | | 7 | For an ODT-simulating MCS: request to the CONTROLLER for disconnection. Words 1, 2, 3, and 4 must have all bits equal to 1. |
| | | 8 | For an ODT-simulating MCS: requests to the CONTROLLER. The CONTROLLER discards all of the requests waiting for an answer from this dialogue and then proceeds as in Variant 2. |
| | | 9 | For an ODT-simulating MCS: requests to the CONTROLLER. The CONTROLLER discards all of the requests waiting for an answer from this dialogue and then proceeds as in Variant 4. |
| | | 10 | For an ODT-simulating MCS: requests to the CONTROLLER. The CONTROLLER discards all of the requests waiting for an answer from this dialogue and then proceeds as in Variant 4. |
| | | 11 | For an ODT-simulating MCS: requests to the CONTROLLER. Words 1, 2, 3, and 4 must have all bits equal to 1. On receiving this request, the operating system starts sending security-related messages, such as security violations or modifications to security settings. |
| | | 12 | For an ODT-simulating MCS: requests to the CONTROLLER. Words 1, 2, 3, and 4 must have all bits equal to 1. On receiving this request, the operating system stops sending security related messages. |
| | [31:08] | | Number of originating MCS. |
| | [23:09] | | Used by the sending MCS for internal purposes. When a reply to this message is sent, this field is included in the reply. |
| | [14:15] | | LSN of remote terminal. |

### Table 3–2. Format 2 for SETUPINTERCOM Function Messages

| Word | Field | Value | Description |
|---|---|---|---|
| 1 | [47:01] | 1 | If the reply is to be truncated at line width. |
|  | [46:01] | 0 | If the user is to be restricted to those commands marked FREE in the input to the table generation. |
|  |  | 1 | If the user can use any command. |
|  | [45:01] |  | ODT bit. When this bit is 1, the CONTROLLER handles the message exactly as if it came from an ODT. |
|  | [43:23] |  | Dialogue number (for an ODT-simulating MCS). The dialogue number is untouched by the CONTROLLER and sent back to the MCS for reference. |
|  | [19:04] | 1 | No response to the message. |
|  |  | 2 | Concise response, OK. |
|  |  | 3 | Expanded response. |
|  | [15:08] |  | Width of the line, where 2 < line width < 80. |
|  | [07:08] |  | Number of lines on a page, where 0 < lines <= 24*80. |
| 2 |  |  | Not used. |
| 3 |  |  | Message component word. When turned on, the bits of this word indicate the existence of different message components, as shown in Table 3–3. |
|  | [47:01] | 1 | Indicates the existence of a special action component word. If bit 0 of the corresponding special action word is turned on, the usercode is used to restrict mix-related commands, and for commands handled by external agents (such as Print System and DSS). If this bit is reset or the special action component word is not present, the usercode is used to restrict PD commands, for WFL jobs, and for commands handled by external agents. |
| 4 to end |  |  | For each specified component, a component information word follows. |
|  |  |  | If bits 0, 1, 2, or 5 of word 3 are turned on, each corresponding component information word is composed of the following two fields: |
|  | [15:16] |  | The length of the component, in bytes. For example, the length of the command text. |
|  | [31:16] |  | The starting index, in words, of the component in the message array. |
|  |  |  | If bit 3 of word 3 is turned on, the corresponding session number component information word contains the session number of the requestor. This number is used for logging purposes. |
|  |  |  | If bit 4 of word 3 is turned on, the corresponding return LSN number component information word contains the value to be returned in the LSN number field. This value can be different from the LSN number of the input message. |

## Continuation Call

An input request to the CONTROLLER might supply only the requested number of lines of output, even though more output information is available. In this situation, the CONTROLLER terminates the queue of messages with a continuation message (variant 4). This message contains sufficient information to continue the input request. To continue the input request, an MCS receiving this message must place this message into the input queue of the CONTROLLER. The message must not be altered before being returned to the CONTROLLER.

Table 3–3 presents the values of the message component word, which is word 3 of format 2 of the SETUPINTERCOM function message.

**Table 3–3. Bits for Message Component Word**

| Bit | Component |
|---|---|
| 0 | Command text |
| 1 | Usercode |
| 2 | Language |
| 3 | Session number |
| 4 | Return LSN number |
| 5 | Convention |
| 47 | Special action |

## ODT-Simulating MCS

When MARC is executed, it is automatically established as an ODT-simulating MCS. Only one MCS in the system can be an ODT-simulating MCS. If you try to establish another ODT-simulating MCS when MARC is running, the request is discarded. If you have already established another MCS as the ODT-simulating MCS, and then you execute MARC, you cannot enter ODT commands from MARC. It is strongly suggested that you do not establish any other MCS as an ODT-simulating MCS, which reserves this capability for use by MARC. You can supply ODT features to any MCS by setting the ODT bit, which lets you enter ODT commands that are sent to the controller and handled as though they came from an ODT.

### Examples

```
R := SETUPINTERCOM(QUEUEARRAYREFID,QUEUEID);

R := SETUPINTERCOM(QUEUEARRAYREFID,QUEUEARRAY[2],MLSCAPABLE);
```

# SIZE Function

The SIZE function can be used to find out the length of a message array.

```
── SIZE ── ( ── <message group designator> ── ) ──────────────────────┤
```

### Explanation

If the message group designator contains a <message designator>construct as in the first and second examples below, the SIZE function returns the length of the message designator. If the message is null, the SIZE function returns 0. If the message group designator contains the construct <message array identifier> <subarray selector> as in the third example, the size of the dimension represented by the leftmost asterisk is returned. If the message group designator contains the construct <message array identifier> as in the fourth example, the size of the leftmost dimension is returned.

### Examples

```
LENGTH := SIZE(MESSAGEID);

LENGTH := SIZE(MESSAGEARRAYID[1,2,3]);

DIMENSIONSIZE := SIZE(MESSAGEARRAYID[1,*,*]);

DIMENSIONSIZE := SIZE(MESSAGEARRAYID);
```

# SYSTEMSTATUS Function

The SYSTEMSTATUS function gathers specified system information and places it in the noncharacter array row.

```
── SYSTEMSTATUS ── ( ── <noncharacter array row> ── , ───────────────→

→─ <arithmetic expression> ── , ── <arithmetic expression> ── ) ──────┤
```

### Explanation

The second and third parameters specify the system information that is to be supplied.

The Boolean value returned is TRUE if, for any reason, the information requested is not returned in the noncharacter array row. Field [11:08] of this value contains the coded reason for failure. If the information requested is returned, the value of the function is FALSE.

The SYSTEMSTATUS function is described in the *SYSTEMSTATUS Programming Reference Manual*.

### Examples

```
 IF SYSTEMSTATUS(A[*],X,Y) THEN
    GO TO TROUBLE;

SYSTEMSTATUS(B[S,*],N+2,Z);

IF SYSTEMSTATUS(ARRAYID,3,JOBNUMBER)
   AND SYSTEMSTATUS(B2[U,*],4,UNITNUMBER)
THEN
   GO TO BIGTROUBLE;
```

# USERDATAFREEZER Function

The USERDATAFREEZER function freezes the SYSTEM/USERDATAFILE against modification through the USERDATA operating system procedure. A description of the parameters, results, and error codes of the function is found in the “MAKEUSER” section of the *Security Administration Guide*.

```
── USERDATAFREEZER ── ( ── <Boolean expression> ── ) ────────────────┤
```

# WRITESPO Function

The WRITESPO function allows you to write data to an ODT. The function returns a Boolean result to indicate that the write has been initiated successfully (FALSE) or that the write could not be initiated (TRUE); if TRUE, the function result contains additional information.

```
── WRITESPO ── ( ── <unit number> ── , ── <character count> ── , ──────→

→─ <noncharacter direct array row> ── , ── <event designator> ── ) ───┤
```

```
<unit number>

── <arithmetic expression> ───────────────────────────────────────────┤
```

```
<character count>

── <arithmetic expression> ───────────────────────────────────────────┤
```

<noncharacter direct array row>

A direct array row whose identifier is declared with a <type> construct.

### Explanation

The function requires the following four parameters:

| Parameter | Description |
|---|---|
| <unit number> | This parameter designates the external unit number of the ODT to which the data is to be written. |
| <character count> | This parameter designates the number of words and characters of data to be transferred. The number of words to be transferred is designated in field [16:17]. The number of additional characters to be transferred is designated in field [19:03] and must be in the range 0 to 5, inclusive. |
| <noncharacter direct array row> | This parameter contains the data to be transferred to the ODT. If desired, the IOMASK of the array can be made equal to an appropriate mask. |
| <event designator> | This parameter designates the event to be caused when the I/O is finished. |

You must declare the noncharacter direct array row and the event designator in the same block.

The WRITESPO function scans the direct array row from the first character either to an end of text (ETX) character or for the number of words and characters indicated by the <character count> parameter. Any occurrence of the character sequence *ESC* is replaced by two blanks.

If there is not an ETX character within the number of characters specified by the <character count> parameter, an ETX character is appended after the final character that is to be transferred. If there is no room in the array to append the ETX, the last character to be transferred is overwritten with an ETX. Only the characters from the beginning of the array to the ETX are transferred to the ODT.

As mentioned previously, the WRITESPO function returns a Boolean result to indicate that the operation was initiated successfully (FALSE) or could not be initiated (TRUE). If the result is TRUE, field [11:08] of the result contains one of the following values:

| Value | Description |
|---|---|
| 34 | The <unit number> parameter did not designate an unreserved ODT. |
| 40 | The third parameter was not a direct array row. |
| 41 | The noncharacter direct array row already has an I/O in progress. |
| 42 | The event designator and the noncharacter direct array row were not declared in the same block. |

You can obtain additional information about a particular WRITESPO operation by interrogating the attributes of the direct array row passed as the third parameter. For details about direct I/O buffer attributes, refer to the *File Attributes Programming Reference Manual*.

### Example

The following statement writes 30 words plus 4 characters (184 characters) of data from the direct array OPMESSAGE to ODT 24. The event IOFIN is caused when the I/O is complete.

```
ERRFLAG := WRITESPO(24,30 & 4 [19:03],OPMESSAGE,IOFIN);
```

ERRFLAG is assigned the value TRUE if the initiation of the transfer is not successful.

# Section 4
# Attributes

This section discusses the two types of DCALGOL attributes:

- Queue attributes
- Task attributes

## Queue Attributes

The syntactical structure, use, and semantics of queue attributes are similar to those of task attributes and file attributes. Illegal attribute references (such as attempts to assign a read-only attribute) result in the display of a run-time error message on the console and entry of this message in the system log. However, the program does not terminate.

```
── <queue attributes> ── . ── <queue attribute name> ─────────────────┤
```

```
<queue attribute name>

──┬─ QACTIVE ───────┬──────────────────────────────────────────────┤
  ├─ QBLOCKSIZE ────┤
  ├─ QDISKERROR ────┤
  ├─ QHEADSIZE ─────┤
  ├─ QINSERTEVENT ──┤
  ├─ QMEMORYLIMIT ──┤
  ├─ QMEMORYSIZE ───┤
  ├─ QMESSAGECOUNT ─┤
  ├─ QREMOVEWAIT ───┤
  ├─ QROWSIZE ──────┤
  ├─ QSIZE ─────────┤
  ├─ QTANK ─────────┤
  └─ QUSERCOUNT ────┘
```

The following are descriptions of each of the queue attributes. Each description includes the data type of the attribute (integer, Boolean, or event) and whether it can be changed (read/write) or only queried (read-only).

## QACTIVE

*Boolean, read/write*

The QACTIVE attribute returns the current active state of a queue. Making this attribute equal to TRUE explicitly activates the queue. Making this attribute equal to FALSE deactivates the queue and flushes any messages currently in the queue. The QACTIVE attribute is initially FALSE.

## QBLOCKSIZE

*Integer, read/write*

The  QBLOCKSIZE attribute specifies the number of words per block in the tank file. The default value is 6000. The value of this attribute is rounded up to a multiple of 30, if necessary, with a minimum of 30 and a maximum of 65520. You cannot alter the QBLOCKSIZE value when the QTANK attribute is set to TRUE.

If you increase the value of QBLOCKSIZE, you can reduce the amount of disk I/O time for a queue. However, this action also increases memory requirements. Two buffers QBLOCKSIZE-words long are allocated for each queue. Therefore, by default, 600 words are required for buffer space.

If you use the COMBINE statement to combine two queues whose QBLOCKSIZEs differ, there is a danger that these two values can be undefined in the host queue if either queue is tanked and if the priority is equal to TRUE (combine to head of the host queue). For more information about the COMBINE statement, refer to “COMBINE Statement” in the “Statements” section.

## QDISKERROR

*Boolean, read/write*

When the QDISKERROR attribute is interrogated, it returns a Boolean value that indicates if any messages in the queue have been lost because of a tanking error.

In tanking messages to disk, the operating system usually recovers the data in a disk write error. If a write error occurs, the operating system stops the tanking and leaves the messages intact in memory.

However, if a read error occurs when detanking messages from disk, the operating system is unable to recover the tanked messages. Because the messages in the disk tank are linked, the entire disk tank of that queue must be flushed by the operating system. To indicate that messages have been lost, the operating system displays the following message:

```
<mixno> TANKING DISK ERROR
```

At the program level, the queue attribute QDISKERROR is given the value TRUE by the operating system for the above situation.

The program can interrogate or change the value of this attribute at any time. Making this value equal to TRUE or FALSE has no effect on the queue; the operating system makes this attribute equal to TRUE only when messages are lost because of a tanking disk error.

## QHEADSIZE

*Integer, read-only*

The QHEADSIZE attribute indicates the size of the first message in the queue. The value of the attribute is 0 if the queue is empty.

## QINSERTEVENT

*Event, read-only*

The QINSERTEVENT attribute is an event-valued attribute; therefore, it is an event designator and can be used wherever an <event designator> construct is used in ALGOL and DCALGOL.

This event is caused each time a message is inserted in a queue with the INSERT statement or COMBINE statement. The QINSERTEVENT attribute is turned off when the last message is removed from the queue by using the REMOVE function, COMBINE statement, or FLUSH statement.

The following examples illustrate two ways to use QINSERTEVENT. In each example, the assignment to L saves the message length.

The following example simply removes the next message from queue Q and waits if there is no message in the queue:

```
WHILE (L:=REMOVE(MSG,Q)) = 0 DO
   WAIT(Q.QINSERTEVENT);
```

In the next example, program execution is suspended until one of the following occurs:

- Five seconds have elapsed.
- There is a message in Q1.
- Event EV has happened.
- There is a message in Q2.

The ellipses (. . .) represent code to handle each contingency. The entire program fragment could be the body of a loop or the outer block of an MCS.

```
I := WAIT((5),Q1.QINSERTEVENT,EV,Q2.QINSERTEVENT);
CASE I OF
BEGIN
   1: % We have WAITed 5 seconds
      ...
```

```
    2: % There was a message in Q1
       WHILE (L:=REMOVE(MSG,Q1)) > 0 DO
       ...

    3: % Event EV has HAPPENED
       RESET(EV)
       ...

    4: % There was a message in Q2
       WHILE (L:=REMOVE(MSG,Q2)) > 0 DO
       ...

  END;
```

Note that there is an implied priority in the WAIT statement. For example, all messages are removed from Q1 before EV or Q2 is noticed. This priority is implicit in the WAIT statement.

## QMEMORYLIMIT

*Integer, read/write*

The QMEMORYLIMIT attribute defines the maximum value that the attribute QMEMORYSIZE can achieve before disk tanking is invoked. The default value is 16384 words. The maximum value of this attribute is (2**16)–1 or 65,535. Giving this attribute the value of 0 causes all messages to be tanked. You can change QMEMORYLIMIT at any time with the following results:

- If you increase the value, the system does not attempt to detank messages in order to raise QMEMORYSIZE to the new limit.
- If you decrease the value, the system tanks messages until QMEMORYSIZE is less than or equal to the new limit.

## QMEMORYSIZE

*Integer, read-only*

The QMEMORYSIZE attribute reflects the current size (in words) of the resident portion of a queue. This size is the sum of the sizes for each complete message area in the queue plus one word (a link word) for each message.

## QMESSAGECOUNT

*Integer, read-only*

The QMESSAGECOUNT attribute is the total number of messages in the queue and includes any that have been tanked.

## QREMOVEWAIT

*Boolean, read/write*

When equal to TRUE, the QREMOVEWAIT attribute causes a REMOVE function on an empty queue to wait until a message is inserted. Note that the same value of QREMOVEWAIT applies to all processes that see the same physical queue.

By making Q.QREMOVEWAIT equal to TRUE, the first example shown for QINSERTEVENT could be replaced by the following:

```
L := REMOVE(MSG,Q);
```

## QROWSIZE

*Integer, read/write*

The QROWSIZE attribute specifies the number of blocks in each row of the tank file. The default value is 5. You cannot change the QROWSIZE value if the QTANK attribute is set to TRUE.

Increasing QROWSIZE reduces the overhead involved in allocating tank space while increasing the space requirements. If the default values for QROWSIZE and QBLOCKSIZE are used, a tank row consists of 500 sectors.

Initially, two rows are allocated when tanking is invoked. Rows are allocated and deallocated as the size of the queue increases and decreases. All disk space is returned when the queue is no longer tanked. The tank file can contain a virtually unlimited number of rows.

If you use the COMBINE statement to combine two queues whose QROWSIZEs differ, there is a danger that these two values can be undefined in the host queue if either queue is tanked and if the priority is equal to TRUE (combine to head of the host queue). For more information on the COMBINE statement, refer to “COMBINE Statement” in the “Statements” section of this manual.

## QSIZE

*Integer, read-only*

The QSIZE attribute is the sum of the sizes of each message in the queue (both in memory and on disk). This size includes the message area only and not the message link words.

# QTANK

*Boolean, read/write*

The QTANK attribute is TRUE if a portion of the queue is currently resident on disk and FALSE if the entire queue is in memory. Making this attribute equal to TRUE causes all messages currently in memory to be tanked to disk. Making QTANK equal to FALSE is an error.

# QUSERCOUNT

*Integer, read/write*

The QUSERCOUNT attribute represents the number of independent users of a queue. QUSERCOUNT is increased by one when the queue is passed by value to a procedure and is decreased by one when exiting the procedure. QUSERCOUNT can also be altered by the ATTACH statement. For more information on the ATTACH statement refer to "ATTACH Statement" in the "Statements" section of this manual.

For example, assume that Q1 references a queue whose physical existence is referenced as QQ1. Assume also that Q2 references a queue (distinct from QQ1) whose physical existence is referenced as QQ2. Finally, assume that the statement ATTACH (Q1,Q2) is executed. After execution, both Q1 and Q2 reference the same physical queue, QQ2. The user count of the physical queue QQ1 is decreased by one, and the user count of the physical queue QQ2 is increased by one. Both Q1.QUSERCOUNT and Q2.QUSERCOUNT equal the user count of the physical queue QQ2.

### Example

```
IF (NOT OUTPUTQ.QACTIVE) OR (OUTPUTQ.QMESSAGECOUNT=0) THEN
    GO TO XIT;
CASE WAIT((2),Q.QINSERTEVENT,E)-1 OF
BEGIN
0: ;
1: S := REMOVE(MSG,Q);
2: GO TO XIT;
END;
```

# Task Attributes

The task attribute names that are associated with a task designator are described in alphabetical order in the following discussion. These task attribute names are pertinent to data comm and are a subset of task attributes. These task attributes, as well as others not pertinent to data comm, can be found in the *Task Attributes Programming Reference Manual*.

```
── <task designator> ── . ── <task attribute name> ────────────────────┤
```

```
<task attribute name>

──┬─ AUTOSWITCHTOMARC ─┬──────────────────────────────────────────────┤
  ├─ BACKUPFAMILY ─────┤
  ├─ DESTSTATION ──────┤
  ├─ DISPLAYONLYTOMCS ─┤
  ├─ INHERITMCSSTATUS ─┤
  ├─ MAXWAIT ──────────┤
  ├─ ORGUNIT ──────────┤
  ├─ SOURCEKIND ───────┤
  ├─ SOURCESTATION ────┤
  ├─ STATION ──────────┤
  └─ TANKING ──────────┘
```

The following are descriptions of each of the task attributes. Each description includes the data type of the attribute (integer, Boolean, real, pointer, or identifier) and whether it can be changed (read/write) or only queried (read-only).

## AUTOSWITCHTOMARC

*Boolean, read/write*

The AUTOSWITCHTOMARC attribute affects only processes that are initiated by a MARC session and that open a remote file. For these processes, this attribute specifies whether or not the task status screen is automatically displayed when the process terminates.

## BACKUPFAMILY

*Identifier, read/write*

The BACKUPFAMILY attribute specifies the family to which all print and punch backup files generated by the task are to be allocated.

## DESTSTATION

*Integer, read/write*

The DESTSTATION attribute allocates or returns the destination station. DESTSTATION can be allocated only by an MCS. When DESTSTATION is allocated, the destination MCS number also is made equal to the MCS number of the MCS currently controlling the destination station.

## DISPLAYONLYTOMCS

*Boolean, read/write*

The DISPLAYONLYTOMCS attribute controls the display of DISPLAY messages. If equal to TRUE, messages generated by this task are not displayed at the ODT.

## INHERITMCSSTATUS

*Boolean, read/write*

The INHERITMCSSTATUS attribute (if TRUE) causes a process to inherit the privileges of the MCS. This task attribute enables an MCS to initiate an external code file that performs some MCS functions.

## MAXWAIT

*Real, read/write*

The MAXWAIT attribute specifies the maximum number of seconds a task can wait on a specified system function.

## ORGUNIT

*Integer, read-only*

The ORGUNIT attribute records the logical station number (LSN) or physical unit number of the unit that initiated the process.

## SOURCEKIND

*Integer, read-only*

The SOURCEKIND attribute returns an integer value equal to the unit type (that is, the value of the KIND file attribute) associated with the unit that originated the task.

## SOURCESTATION

*Real, read/write*

The SOURCESTATION attribute allocates or returns the originating station (LSN). This attribute can be allocated only by an MCS.

## STATION

*Integer, read/write*

The STATION attribute stores the LSN of the station to be assigned any remote files used by the process. If STATION has a nonzero value, the TITLE file attribute does not affect selection of a station for remote files. If the STATION value is 0, the TITLE file attribute determines the station that is assigned a remote file.

## TANKING

*Mnemonic, read/write*

The TANKING attribute designates the default tanking mechanism for remote files used by the process. Tanking is a method that the system can use to temporarily store messages that a process writes to a remote file.

### Examples

```
REPLACE TSK.DESTNAME BY PB;

IF TSK.SOURCEKIND = VALUE(REMOTE) THEN
    MYLSN := TSK.SOURCESTATION;
```

# Section 5
# DCWRITE Information

This section is structured as follows:

- General DCWRITE information, which applies to all DCWRITE types
- Specific DCWRITE information, listed in order of DCWRITE type number

## General DCWRITE Information

A message control system (MCS) recognizes two general classes of messages:

- Messages constructed for use with the DCWRITE function by the MCS.
- Messages generated elsewhere within the data comm subsystem that eventually appear in one of the queues of the MCS.

Messages constructed by the MCS for use with the DCWRITE function adhere to a generally consistent format insofar as requirements for minimum message size and field locations for certain types of information. For example, the type field for DCWRITE messages is always located in MESSAGE [0].[47:08], but the value in this field varies depending on the type of DCWRITE function to be performed.

### DCWRITE Message Format

Messages constructed for use with the DCWRITE function are at least six words in length. (Exceptions to this requirement are noted where applicable.) The general format of messages used in conjunction with the DCWRITE function is presented in Table 5–1. The following acronyms are used in Table 5–1:

| Acronym | Meaning |
|---|---|
| DLS | A number made up of three components that are separated by colons (:) (the NSP number, LSP number, and the station number). |
| FRSN | File relative station number. |
| LSN | Logical station number. |
| LSP | Line support processor. |
| NSP | Network support processor. |
| RSN | Relative station number. |

**Table 5–1. DCWRITE Message Format (General)**

| Word | Field | Value | Description |
|---|---|---|---|
| [0] | [47:08] | | Type field. |
| | [39:16] | | Variant field. |
| | [23:24] | | LSN-FRSN-DLS number field |
| | [23:01] | 0 | [22:23] is an LSN or FRSN. If [22:23] is an FRSN, then [23:10] is a file number, and [13:14] is an RSN. |
| | | 1 | [22:23] is a DLS number: [22:07] is the relative NSP number, [15:08] is the line number (line number=relative LSP number * 16 + adapter number), and [07:08] is the station number. |
| [1] | [47:08] | | Priority of output (0-127); otherwise, 0. |
| | [39:08] to [32:01] | | TOGGLES: [39:01] is TOGGLE [8], [38:01] is TOGGLE [7], and so on until [32:01] is TOGGLE [1]. |
| | [31:32] | | Not used. |
| [2] | [47:08] | | Retry count field. |
| | [39:16] | | Text size field. |
| | [23:24] | | Not used. |
| [3] | [47:24] | | Not used. |
| | [23:08] | | Tally [0]. |
| | [15:08] | | Tally [1]. |
| | [07:08] | | Tally [2]. |
| [4] | [47:24] | | Message number field. |
| | [23:24] | | Not used. |
| [5] | [47:48] | | Not used (reserved for system use). |
| [6] | Text | | |
| [n] | Text | | |

In many respects, messages resemble arrays when the information within a message is manipulated. The ALLOCATE statement is used to obtain space for a message. The ALLOCATE statement requires two parameters: a message or a message array identifier, followed by an arithmetic expression that indicates the number of words the message spans. For example, the following statement allocates a 6-word message and places the data descriptor for it in the message variable MSG:

```
ALLOCATE(MSG,6);
```

After being allocated, a simple message variable can be referred to as a one-dimensional array. For example, the following construct illustrates a typical statement that appears in conjunction with messages being constructed for use with the DCWRITE function:

```
MSG[0].[47:08] := DCWRITETYPEATTACH;
```

The hidden dimension of a message has a lower bound of 0. All words in a message are equal to binary 0 when first allocated.

Interpretations of the fields within a message constructed for the DCWRITE function are described in the following discussion.

## Type Field (MSG[0].[47:08])

The type field contains one of a number of possible values to inform the DCWRITE function which operation to perform. Specific values are discussed with each DCWRITE type later in this section.

## Variant Field (MSG[0].[39:16])

The variant field is used for qualification, variations, or additional information with certain DCWRITE types (for example, carriage control in the WRITE DCWRITE function). The specific meaning of the variant field is discussed with each applicable DCWRITE type.

## LSN/FRSN/DLS Field (MSG[0].[23:24])

For most DCWRITE types, an LSN must be supplied to designate the station that is to be affected by the particular DCWRITE function. LSNs can be obtained through several mechanisms, one of which is the STATION ATTACH DCWRITE function. An extension allows this field to contain the DLS number of a station, indicated by MSG[0].[23:01] = 1. The format of the field in that case is as follows:

```
MSG[0].[22:07] = Relative NSP number
       [15:08] = Line number
       [07:08] = Station number
```

Some DCWRITE types require an FRSN instead of the customary LSN or DLS number. An FRSN is obtained for a station at file-open time.

## Priority Output Field (MSG[1].[47:08])

The priority of the message produced by the MCS is contained in this field. The MCS can produce 128 levels of priority, with 0 as the lowest priority and 127 the highest. If the priority level of the message is not zero, the DCC inserts the message into the station queue following messages of higher or equal priority and preceding messages of lower priority. In this way, the MCS messages are transmitted in a high-to-low priority sequence, rather than the order in which the messages were received by the DCC.

### TOGGLE and TALLY Fields (MSG[1].[39:08] and MSG[3].[23:24])

For certain DCWRITE types (WRITE and READ-ONCE ONLY), values can be supplied in these fields for an NDLII-written algorithm that initializes its TOGGLEs and TALLYs by using the INITIALIZE statement in the NDLII algorithm.

### Retry Count Field (MSG[2].[47:08])

The value in the retry count field (supplied by the MCS) is used to determine the total number of retries for any error conditions encountered by the NSP. If a value of 255 (decimal) is supplied for this field, the retry count designated for the station in the DATACOMINFO file is used.

### Text Size Field (MSG[2].[39:16])

The text size field specifies the number of bytes of meaningful text (beginning at MSG[6], six EBCDIC characters per word) in the message. For some DCWRITE types, this field is not applicable.

### Message Number Field (MSG[4].[47:24])

An MCS can elect to assign message numbers to all messages that it allocates in the system, thus providing some ability to audit the flow of its messages in the system. The software system preserves the integrity of this field and does not use it for other purposes.

If the MCS is participating in object job I/O for a station and the MCS receives an OBJECT JOB OUTPUT (Class = 3) message from the object job, the message number in this field in that message contains the FRSN. If the MCS chooses to forward this message to the station, it can choose not to alter the message number field so that if the MCS must later recall the message, it can determine the origin (for example, which file and which station in the file) of the recalled message.

### Text (Beginning at MSG[6])

Text is assumed to be EBCDIC characters (8-bit bytes) by the components of the data comm system and is left justified starting in word 6 of the message. Some DCWRITE types require no text. If translation for ASCII or other character sets is required for a given station, the translation is accomplished at the NSP.

## MCS Calls on DCWRITE

Depending on the specific type of DCWRITE function desired, one or two parameters are supplied to the DCWRITE function, as follows:

```
DCWRITE (<message designator>) or
DCWRITE (<message designator>,<queue designator>)
```

DCWRITE is a typed function and returns a value that indicates whether or not errors occurred during the performance of the desired function.

All calls on the DCWRITE function require that a nonnull message of at least six words be passed as the first (and possibly only) parameter. The exact minimum message size depends on the particular DCWRITE function to be performed.

Except where noted in specific DCWRITE calls, on exit from the DCWRITE function, the message passed as a parameter is null. The nonzero values returned from the DCWRITE function denote error situations that cannot be overlooked and can indicate programming errors in the calling MCS. In these error cases, the message that was passed as a parameter to the DCWRITE function remains unaltered and is nonnull on exit from the DCWRITE function.

In many cases, the task of the DCWRITE function can be properly accomplished only with the cooperation of the NDLII algorithm. For example, a call on the WRITE DCWRITE (DCWRITE Type = 33) function results in the transmission of the text only if the NDLII algorithm performs the proper output action. This cooperation is especially crucial when error conditions occur.

# Summary of DCWRITE

Table 5–2 shows a summary of the DCWRITE functions, and Table 5–3 lists the DCWRITE errors in a summary format. The possible values returned by the DCWRITE function are given in Table 5–4.

**Table 5–2. Summary of DCWRITE Types**

| Category | TypeNumber | Function |
|---|---|---|
| ENVIRONMENT | 0 | INITIALIZE PRIMARY QUEUE |
| | 1 | STATION ATTACH |
| | 2 | INTERROGATE MCS |
| | 3 | INTER-MCS COMMUNICATE |
| | 4 | INTERROGATE STATION ENVIRONMENT |
| | 5 | ATTACH SCHEDULE STATION |
| STATION | 32 | CHANGE CURRENT QUEUE |
| | 33 | WRITE |
| | 34 | READ-ONCE ONLY |
| | 35 | ENABLE INPUT |
| | 36 | DISABLE INPUT |
| | 37 | MAKE STATION READY/NOT READY |
| | 38 | SET APPLICATION NUMBER |
| | 39 | SET CHARACTERS |
| | 40 | SET TRANSMISSION NUMBER |
| | 41 | RECALL MESSAGE |
| | 42 | STATION DETACH |
| | 43 | SET/RESET LOGICALACK |
| | 44 | ACKNOWLEDGE |
| | 45 | TRANSFER STATION CONTROL |
| | 46 | WRITE AND RETURN |
| | 48 | NULL STATION REQUEST |
| | 49 | SET/RESET SEQUENCE MODE |
| | 53 | WRITE TO TRANSFERRED STATION |
| | 55 | SEND MCS RESULT MESSAGE |
| | 56 | SET PSEUDOSTATION ATTRIBUTES |
| ATTRIBUTE | 64 | STATION ASSIGNMENT TO FILE |
| | 65 | WRITE TO OBJECT JOB |

### Table 5–2. Summary of DCWRITE Types

| Category | TypeNumber | Function |
|---|---|---|
| LINE | 66 | STATION BREAK |
| | 67 | ADD STATION TO FILE |
| | 68 | CHANGE TERMINAL ATTRIBUTES |
| | 69 | SUBTRACT STATION FROM FILE |
| | 96 | MAKE LINE READY |
| | 97 | MAKE LINE NOT READY |
| | 98 | DIALOUT |
| | 99 | DISCONNECT |
| | 100 | ANSWER THE PHONE |
| | 101 | INTERROGATE SWITCHED STATUS |
| | 102 | SET/RESET AUTOANSWER |
| | 103 | SET/RESET LINE LOGS-TALLYS |
| | 104 | LINE INTERROGATE |
| | 105 | FORCE LINE NOT READY |
| RECONFIGURATION | 128 | SWAP LINES |
| | 129 | EXCHANGE LSPS |
| | 130 | MOVE/ADD/SUBTRACT STATION |
| | 131 | UPDATE LINE ATTRIBUTES |

### Table 5–3. Summary of DCWRITE Errors

| Value | Description |
|---|---|
| 064 | MESSAGE MAY NOT BE NULL |
| 065 | MUST INITIALIZE PRIMARY QUEUE |
| 066 | MCS NOT DEFINED IN NDLII |
| 067 | QUEUE PARAMETER REQUIRED |
| 068 | INVALID DCWRITE TYPE |
| 069 | STATION IS NOT YOURS |
| 070 | STATION IS ALREADY YOURS |
| 071 | INVALID LSN |
| 072 | TEXT SIZE > ACTUAL MESSAGE SIZE |
| 073 | MESSAGE TOO SMALL |

### Table 5–3. Summary of DCWRITE Errors

| Value | Description |
|---|---|
| 074 | PRIMARY QUEUE ALREADY INITIALIZED |
| 075 | DATACOM NOT INITIALIZED |
| 076 | UNKNOWN NSP |
| 077 | UNKNOWN LINE NUMBER |
| 078 | UNKNOWN STATION |
| 079 | BAD NAME FORMAT |
| 080 | INVALID FILE NUMBER |
| 081 | INVALID RELATIVE STATION NUMBER |
| 082 | STATION ALREADY IN FILE |
| 083 | STATION NOT ASSIGNED TO THE FILE |
| 084 | STATION MUST BE NOT READY |
| 085 | THIS LINE MAY NOT BE DIALED OUT |
| 086 | TYPE IS IN APPROPRIATE FOR LINE |
| 087 | NSP NOT INITIALIZED |
| 088 | STATION HAS NO LINE ASSIGNMENT |
| 089 | LINE CURRENTLY BEING CHANGED |
| 090 | STATION CURRENTLY BEING CHANGED |
| 091 | MCS DOES NOT CONTROL THE LINE |
| 092 | STATION/STATIONSET INFORMATION WAS NOT FOUND IN THE DATACOMINFO FILE |
| 093 | DESIGNATED NSP CANNOT BE EXCHANGED |
| 094 | IMPROPER LSP MASK SUPPLIED |
| 095 | LINES MUST BE IN SAME NSP |
| 096 | LINE CANNOT BE SWAPPED (MAXSTATIONS = 0) |
| 097 | INVALID DESTINATION (MAXSTATIONS=0) |
| 098 | NO MORE ROOM ON LINE |
| 099 | STATION NOT CAPABLE OF SEQUENCE MODE |
| 100 | SEQUENCE NUMBER SIZE MUST BE 1 THROUGH 8 |
| 101 | STATION MYUSE VALUE INVALID |
| 102 | MCS COULD NOT BE EXECUTED |
| 103 | MCS IS NOT RUNNING |
| 104 | INVALID APPLICATION NUMBER |
| 105 | MCS IS ALREADY IN EXECUTION |

### Table 5–3. Summary of DCWRITE Errors

| Value | Description |
|---|---|
| 106 | NSP IS NOT READY OR OFFLINE |
| 107 | NAME INDEX NOT WITHIN MESSAGE AREA |
| 108 | UNKNOWN TERMINAL NAME |
| 109 | DEFAULT TERMINALS NOT ALLOWED |
| 110 | UNKNOWN LINE NAME |
| 111 | STATION ADAPTER AND MODEM MISMATCHED |
| 112 | STATION ADAPTER AND TERMINAL MISMATCHED |
| 113 | STATION/LINE LINE CONTROL MISMATCHED |
| 114 | STATION/LINE ADAPTER TYPES MISMATCHED |
| 115 | LINE MODEM AND STATION ADAPTER MISMATCHED |
| 116 | ADAPTER TYPE OUT OF RANGE |
| 121 | STATION MUST BE ATTACHED |
| 122 | DLS REQUIRED (DLS.[23:01] = 1) |
| 124 | UNKNOWN LSP NUMBER |
| 125 | LINE ALREADY ATTACHED (MAINTENANCE) |
| 126 | LINE NOT ATTACHED (MAINTENANCE) |
| 137 | BOTH LINES MUST BE DIALOUT |
| 138 | SCHEDULE STATION CANNOT HAVE LINE |
| 139 | SCHEDULE STATION CANNOT BE TRANSFERRED |
| 140 | SCHEDULE STATION CANNOT BE ADDED TO FILE |
| 141 | IMPROPER VARIANT FIELD |
| 143 | STATION CANNOT BE ADDED TO FILE FOR SWAP JOB |
| 144 | INVALID OUTPUT TANKING SPECIFICATION |
| 170 | IOERROR ON DATACOMINFO FILE |
| 171 | DCRECON DSED AFTER IOERROR |
| 173 | DCRECON ERROR OCCURRED IN SUBTRACTING STATION |
| 174 | DCRECON ERROR OCCURRED IN ADDING STATION |
| 175 | LINE STATION ALREADY ASSIGNED |
| 176 | DCRECON ERROR OCCURRED IN SWAPPING LINES |
| 177 | NSP NOT FOUND FOR LSP |
| 178 | FILE HAS BEEN CLOSED DCWRITE=67 |
| 179 | OUTPUT PRIORITY MUST BE IN THE 0 TO 127 RANGE |
| 183 | NOT IN NETWORK MODE |

**Table 5–3. Summary of DCWRITE Errors**

| Value | Description |
|---|---|
| 184 | ATTACH NOT VALID FOR LOCAL STATION |
| 185 | CONNECT TO LOCAL HOST NOT VALID |
| 186 | INVALID BNA STATION |
| 187 | COULD NOT INITIATE STATION TRANSFER |
| 188 | CANNOT ISSUE ON A PSEUDOSTATION |
| 189 | ALLOWED ONLY ON A PSEUDOSTATION |
| 190 | STATION ALREADY HAS A FULLY-PARTICIPATING MCS |
| 191 | STATION DOES NOT HAVE FULLY-PARTICIPATING MCS |
| 192 | ALLOWED ONLY BY FULLY-PARTICIPATING MCS |
| 193 | NO PSEUDOSTATIONS AVAILABLE |
| 194 | SWITCHED LINE MAY NOT BE SWAPPED WITH PRIVATE LINE |
| 195 | BIT-ORIENTED LINE MAY NOT BE SWAPPED WITH CHARACTER-ORIENTED LINE |
| 196 | INCOMPATIBLE ORIGINAL DCWRITE TYPE AND RESULT CLASS |
| 197 | INVALID ODT UNIT NUMBER |
| 198 | ALLOWED ONLY BY THE ODT/DRIVER |
| 199 | INVALID PSEUDOSTATION NAME |
| 200 | DUPLICATE PSEUDOSTATION NAME |
| 201 | CANNOT SUBTRACT STATION FROM FILE |
| 202 | NO PATH TO LSP FROM DESTINATION NSP |
| 203 | DATACOMINFO FILE IN USE BY RECONFIGURATION |
| 204 | REQUEST WAS PREVIOUSLY DONE |
| 205 | STATION HAS BEEN PREPARED FOR DETACH |
| 206 | ONLY VALID FOR POSIX TTY FILES |
| 207 | UNKNOWN POSIX KEY CONTROL VARIANT |
| 208 | CANNOT TRANSFER LIMITED PSEUDO |
| 209 | FILE SIZE ON INITIALIZE PRIMARY QUEUE IS INVALID |
| 210 | STATION SIZE ON INITIALIZE PRIMARY QUEUE IS INVALID |
| 211 | FRSN VALUE GREATER THAN 24 BITS |
| 212 | MESSAGE EXCEEDS NSP MESSAGE SIZE |

**Table 5–4. Results from the DCWRITE Function**

| Value | Description |
|---|---|
| 0 | The function was accomplished (no apparent errors). |
| 64 | The message parameter was null (message must be nonnull). |
| 65 | The primary queue was not yet initialized (for example, INITIALIZE PRIMARY QUEUE DCWRITE not yet performed). |
| 66 | The calling MCS was not named in the DATACOMINFO file, or an invalid MCS number was supplied. |
| 67 | A queue parameter was required for this DCWRITE function, but none was supplied. |
| 68 | The type requested was not implemented (probably an incorrect value was supplied in the type field). |
| 69 | The station name, LSN, or DLS number supplied to the DCWRITE function is under control of a different MCS. |
| 70 | The station name, LSN, or DLS number supplied to the STATION ATTACH DCWRITE was already attached to the calling MCS. |
| 71 | The LSN supplied was invalid. |
| 72 | The value specified in MSG[2].[39:16] (text size) exceeded the message size. |
| 73 | The message parameter supplied did not meet the minimum size requirements for the specific DCWRITE function called. |
| 74 | The INITIALIZE PRIMARY QUEUE DCWRITE was previously called (only one call of this DCWRITE is allowed and necessary for the operation of the MCS). |
| 75 | The data comm subsystem was not yet initialized. |
| 76 | An invalid or unknown relative NSP number was specified in the DLS number. |
| 77 | An invalid or unknown line number was designated in the DLS number. |
| 78 | An invalid or unknown station number was specified in the DLS number. |
| 79 | An external standard-form identifier (for example, station name) had improper form or was too long. |
| 80 | An invalid file number was specified in the FRSN. (This result might occur if the file is valid but has CLOSED with retention.) |
| 81 | An invalid relative station number was specified in the FRSN. |
| 82 | The station supplied to the STATION ASSIGNMENT TO FILE DCWRITE was already assigned to the file. |
| 83 | An MCS attempted to write to an object job before the station was assigned to the file. |
| 84 | The type of DCWRITE attempted requires the station to be in the NOT READY state. |

**Table 5–4. Results from the DCWRITE Function**

| Value | Description |
|---|---|
| 85 | An attempt was made to dial out on a line for which no automatic calling unit was supplied. |
| 86 | An attempt was made to perform a switched-line function on a line that was not switched. |
| 87 | An attempt was made to perform a data comm function that resulted in a reference to an uninitialized NSP. |
| 88 | An attempt was made to perform a data comm function to/on a station line that had no current hardware (line) assignment. |
| 89 | An attempt was made to perform a data comm function to/on a line that was currently involved in reconfiguration. |
| 90 | An attempt was made to perform a data comm function to/on a station that was currently involved in reconfiguration. |
| 91 | An attempt was made to perform a data comm function to/on a line over which the requesting MCS had no control. |
| 92 | An attempt was made to perform a move/add/subtract station request when the station information did not exist in the DATACOMINFO file. |
| 93 | An attempt to perform LSP-exchanging was made where the indicated LSP was |
|  | was not known to be exchanged with any other NSP. |
| 94 | An attempt to perform LSP-exchanging was made where the LSP mask that was supplied was inappropriate (for example, the indicated LSPs were under control of the other NSP or did not exist). |
| 98 | An attempt to move a station was made in which the destination line had no additional table space in which to accommodate the station. |
| 99 | An attempt was made to set/reset sequence mode for a station that was not sequence-mode-capable. |
| 100 | The sequence number size supplied with set/reset sequence mode was less than one or greater than eight. |
| 101 | A request to assign a station to a file was denied because of improper use (for example, output-only device to the input file). |
| 102 | An MCS could not be initiated (either the program was not present in the disk directly or it was not a code file). |
| 103 | An attempt was made to communicate with an MCS that was not initiated/executing, and no automatic initiation was requested. |
| 104 | An invalid application number was supplied. |
| 105 | The MCS was already running. |
| 106 | The NSP designated as the recipient for an LSP exchange request was either NOT READY, not initiated, or offline. |
| 107 | The name index was outside the message area. |

### Table 5–4. Results from the DCWRITE Function

| Value | Description |
|---|---|
| 108 | The terminal name was unknown. |
| 109 | A default terminal name was not acceptable as a terminal name. |
| 110 | The line name is unknown. |
| 111 | The adapter and modem were incompatible. |
| 112 | The adapter and terminal were incompatible. |
| 113 | The station and line control procedure were incompatible. |
| 114 | The station and line adapter were incompatible. |
| 115 | The modem and line adapter were incompatible. |
| 116 | An invalid adapter number was supplied. |
| 121 | The station was not attached to the requesting MCS. |
| 122 | A DLS number was required (DLS.[23:01] = 1). |
| 124 | The LSP was not defined in the NSP. |
| 125 | A line was already attached. |
| 126 | A line was not attached. |
| 137 | Both lines must be dial-out capable. |
| 138 | The schedule station might not have had a line assignment. |
| 139 | The schedule station might not have been transferred. |
| 140 | The schedule station might not have been added to a file. |
| 141 | The variant field was improper for this DCWRITE. |
| 143 | An invalid attempt was made to add a station to a file of a SWAP task. |
| 144 | The output tanking value was invalid. |
| 170 | An I/O error occurred reading from or writing to the DATACOMINFO file. |
| 171 | DCRECON failed to recover after an I/O error, or failed to link to the DATACOMSUPPORT library. |
| 173 | A DCRECON error occurred in subtracting a station. |
| 174 | A DCRECON error occurred in adding a station. |
| 175 | The station is already assigned to a line. |
| 176 | A DCRECON error occurred in swapping lines. |
| 177 | The destination NSP was not found for the LSP. |
| 178 | The file has been closed (DCWRITE = 67). |
| 179 | Output priority must be in the 0 to 127 range. |
| 180 | This result is reserved. |
| 181 | This result is reserved. |

### Table 5–4. Results from the DCWRITE Function

| Value | Description |
|---|---|
| 182 | This result is reserved. |
| 183 | Not in network mode. |
| 184 | Attach not valid for the local station. |
| 185 | Connect to local host not valid. |
| 186 | Invalid BNA station. |
| 187 | The station transfer MCS could not be initiated. |
| 188 | The type of DCWRITE attempted cannot be performed on a pseudostation. |
| 189 | The type of DCWRITE attempted can be performed only on pseudostations. |
| 190 | The station already has a fully participating MCS. |
| 191 | The station does not have a fully participating MCS. |
| 192 | The type of DCWRITE attempted can be performed only by a fully participating MCS. |
| 193 | All pseudostations are in use. |
| 194 | A switched line cannot be swapped with a nonswitched line. |
| 195 | A bit-oriented line cannot be swapped with a character-oriented line. |
| 196 | The original DCWRITE type supplied in the message is incompatible with the result type. |
| 197 | The ODT unit number given is not a valid ODT unit number. |
| 198 | The transfer of a pseudostation to represent a system ODT unit is permitted only by COMS/ODT/DRIVER. |
| 199 | The pseudostation name is not properly formatted. |
| 200 | The pseudostation name is not unique. |
| 201 | The station could not be subtracted from the file specified in a SUBTRACT STATION FROM FILE DCWRITE (Type = 69). |
| 202 | An attempt was made to transfer a Line Support Processor (LSP) from one Network Support Processor (NSP) to another, but no path exists from the LSP to the destination NSP. |
| 203 | The DATACOMINFO file is being modified; this request cannot be honored at this time. The MCS can submit the request (Type = 1 or Type = 56) at a later time. |
| 204 | A second “STATION DETACH” (DCWRITE type = 42) with a “Prepare For Detach” indication (variant = 3) has been issued but a "Prepare For Detach" is already pending for that station.  The second request has been ignored. |
| 205 | Only an “INTERROGATE STATION ENVIRONMENT” (DCWRITE type = 4) or a “STATION DETACH” (DCWRITE type = 42) is allowed while a “STATION DETACH” with a "Prepare For Detach" (variant = 3) request is active. |
| 206 | Not applicable. |

**Table 5–4. Results from the DCWRITE Function**

| Value | Description |
|---|---|
| 207 | Not applicable. |
| 208 | A limited pseudostation cannot be transferred. |
| 212 | An attempt was made to send a message to an NSP based station with more than 9180 bytes of data. |

# Pseudostations and Fully Participating MCSs

The data comm subsystem supports a general implementation of pseudostations, and full participation of an MCS with transferred pseudostations. A pseudostation can be allocated to an MCS by the operating system and the MCS can then “participate” by interposing itself between a data comm program and the pseudostation. Full participation allows an MCS to participate in data comm functions, such as DCWRITE, performed on a transferred pseudostation.

The following sections describe the features of full participation and pseudostations in greater detail. The message formats of the DCWRITE types involved also detail the action or result of the data comm features.

## Pseudostations

A pseudostation is a virtual station that can be attached to, and controlled by, an MCS in a manner similar to a “real” station, that is, one declared in the DATACOMINFO file. However, unlike a real station, a pseudostation is not declared in the DATACOMINFO file, has no line assigned, and need not have a corresponding physical terminal on the local host.

To an application program, a pseudostation looks like any other station. An MCS can send messages to, and receive messages from, a pseudostation through the standard data comm interface procedures. Other programs can open a remote file to a pseudostation and use the READ and WRITE verbs in the various languages. In addition, one MCS can transfer control of a pseudostation to another MCS and can perform most DCWRITE functions that do not require physical line assignment.

However, a pseudostation differs from a station defined in the DATACOMINFO file in that it must have an MCS that is ultimately responsible for performing most DCWRITE functions. Usually, this MCS is the one that initially requested allocation of the pseudostation. The operating system controls the allocation and deallocation of pseudostations and knows which MCS is controlling each station at any given time. However, the operating system has no knowledge of any mapping from a pseudostation to a process, physical terminal, or other entity. Such mappings must be made and maintained solely by an MCS.

Pseudostations are useful for applications that require some form of virtual terminal capability, yet need to remain compatible with existing MCSs and application programs. A pseudostation can be allocated by one MCS and transferred to another, but the first MCS can still exercise control over the data comm functions (such as DCWRITE) performed on the station (see the description that follows on full participation). Thus, the corresponding station might be located on a different host computer, but only the allocating MCS would need to know that.

The MCP provides a second class of pseudostations that are referred to as limited pseudostations; they provide the functionality described previously, but are not transferable.

## MCS Participation in Data Comm Functions (Full Participation)

The data comm subsystem offers a feature called “participation” under which an MCS can interpose itself between a station and a process that has a dialogue established with that station. That process need not be informed that the MCS is actually participating in the dialogue. The MCS can simply provide editing, translation, or message-switching services in a transparent manner.

The full participation feature allows an MCS to participate in the remote-file operations and the data comm functions (such as DCWRITE) performed on a station after control of that station has been transferred to another MCS. In the following discussion of this feature, the MCS that transfers control of a station is referred to as the fully participating MCS, and the MCS that receives control of the transferred station is referred to as the controlling MCS.

Under full participation, as with remote-file participation, an MCS can provide editing, translation, or message-switching services for messages directed to a station. Full participation, however, currently applies only to pseudostations. An error result is returned to an MCS if it requests full participation when it transfers control of a station declared in the DATACOMINFO file to another MCS.

After full participation has been requested, the fully participating MCS receives notification when the MCS that controls the station performs certain DCWRITE functions on that station. The request messages from these DCWRITE functions are intercepted and placed in the primary queue of the fully participating MCS. That MCS can then perform some action that emulates the requested function or can route the request to another process that controls a physical terminal. In any event, the fully participating MCS must be capable of generating MCS result messages and forwarding them to the controlling MCS.

An MCS that elects to invoke full participation on a pseudostation must also be capable of remote-file participation. Control of a pseudostation can be transferred, with full participation requested, to any other MCS. The new controlling MCS can then start a task that opens a remote file to the station. If the new controlling MCS has not requested remote-file participation, the MCS that has requested full participation must also perform remote-file participation in order to successfully open a remote file. Because of the “virtual” nature of pseudostations, the data comm subsystem requires a destination to send messages written to the file and to fulfill the requests of the task to read from the file.

In the case just described, the fully participating MCS receives the OBJECT JOB OUTPUT (Class = 3) result messages that would otherwise have been given to an MCS that requested remote-file participation. In addition, the fully participating MCS is responsible for generating WRITE TO OBJECT JOB (Type = 65) DCWRITE messages to satisfy READ requests performed by the task that has opened the remote file.

# Specific DCWRITE Information

The individual DCWRITE types are discussed in the information that follows. At least one example is included with the description of each DCWRITE type.

The descriptions of the specific DCWRITE types are subdivided into the following topics:

- Required Parameters/Fields
- Explanation
- Examples

  For all examples, the following declarations are assumed:

```
MESSAGE MSG,MSG1;
QUEUE PRIMARYQ,CURRENTQ,INFOQUE;
INTEGER RESULT,MCSNR,LSN,NSPNR,LINENR,STANR,FRSN,APPLNO;
INTEGER FILENR,DL,LSPMASK,TESTNR;
```

## INITIALIZE PRIMARY QUEUE (DCWRITE Type = 0)

### Required Parameters/Fields

The message parameter and the queue parameter are required, along with the following:

| Word | Field | Value | Description |
|---|---|---|---|
| 0 | [47:08] | 0 | Type |
|  | [24:01] | 1 | Expanded Capacity MCS |

### Explanation

The primary queue is the first communication link established between an MCS and the data comm portion of the operating system. This communication link distinguishes an MCS from other DCALGOL programs. Therefore, when a DCALGOL program executes an INITIALIZE PRIMARY QUEUE DCWRITE function in an attempt to establish the desired link, several checks are made to guarantee the legitimacy of the DCALGOL program to become an MCS. These checks are as follows:

- The title of the DCALGOL program must be one of those referenced as the title of an MCS in the DATACOMINFO file.
- The DCALGOL program must not already be running under another job number as an MCS.
- Normally, if an MCS is not running when it is required, the MCS is initiated by the DCC; if neither the MCS nor the DCC is running, both are initiated if the AUTODC option is equal to TRUE. However, if the MCS is a procedure with formal parameters, the attempt by the DCC to initiate the MCS terminates abnormally with a parameter mismatch. Thus, the only way such MCSs can be initiated is through a run invocation by another program or a RUN control card where the parameters can be correctly supplied.

The INITIALIZE PRIMARY QUEUE DCWRITE function must be performed once (and only once) during the course of the existence of the MCS in the system job mix (preferably as one of the first things the MCS accomplishes). The first DCWRITE function performed by the MCS must be an INITIALIZE PRIMARY QUEUE DCWRITE function.

In an NDLII source program, the title of an MCS is associated with each station defined in the source program by designating the MCS station attribute. When data comm is initialized, the operating system assigns MCS numbers sequentially based on the order in which stations are examined in the DATACOMINFO file. The operating system examines each station starting with the first station on the first line of the first LSP of the first NSP. This process continues until all stations of every LSP and NSP have been examined. Stations not assigned to a line are then examined.

Always set the variant specified with the code line MSG[0] . [24:1]. If the variant is not set, some old format messages are returned that contain fields of insufficient size to hold the related data.

The minimum acceptable message size for this DCWRITE type is six words. The message is nonnull on return from the DCWRITE function. MSG[0] remains unaltered. The functional value returned for the DCWRITE function is 0 if the primary queue is initialized. In this case, MSG[1] contains the MCS number and MSG[2] contains the value of the maximum logical station number (LSN), as determined from the DATACOMINFO file in [23:24], and the number of schedule stations available for use in [47:08]. The level of the SOURCENDLII from which the DATACOMINFO file was originally created is in MSG[3]. This word is defined as follows:

```
MSG[3].[35:12] = Mark level
MSG[3].[23:12] = Cycle number
MSG[3].[11:12] = Patch number
```

MSG[4] contains the maximum number of pseudostations in the standard pseudostation pool. In addition, the name of the current data comm file prefix is placed in EBCDIC starting in MSG[6] in external standard form, and the length of the file prefix (in number of characters) is stored in the text size field (MSG[2].[39:16]). The queue designator supplied as the queue parameter is activated and references the primary queue. The DATACOMINFO TIMESTAMP attribute is returned as a real value in MSG[11] and is interpreted as follows:

```
MSG[11].[47:24] = Date as "MMDDYY"
MSG[11].[23:24] = Time as "HHMMSS"
```

The length of the longest data comm message allowed is returned in MSG[12]. Before determining the value of this word the program should first check that the message returned is at least 13 words long, because previous versions of the operating system returned a message 12 words long.

The field MSG[14] . [47:24] of the input message is modified to return the current maximum LSN allocated to a pseudostation in the expanded pool of station entries for pseudostations. The field MSG[14] . [23:24] of the input message is modified to return the current minimum LSN allocated to a pseudostation in the expanded pool of station entries for pseudostations.

Nonzero functional values are returned if the primary queue is not initialized.

| Value | Description |
|---|---|
| 64 | The message parameter is null. |
| 66 | The calling DCALGOL program is not named as an MCS in the current DATACOMINFO file. |
| 67 | The queue parameter is not supplied. |
| 73 | The message parameter supplied does not meet the minimum size required. |
| 74 | The INITIALIZE PRIMARY QUEUE DCWRITE was previously executed. |
| 75 | The data comm subsystem is not initialized (AUTODC is equal to FALSE or the data comm subsystem files are not on disk). |
| 105 | The MCS is already in execution. |

### Example

```
ALLOCATE(MSG,6);
MSG[0].[47:08] := 0;
MSG[0].[24:01] := 1;
RESULT := DCWRITE(MSG,PRIMARYQ);
MCSNR := MSG[1];
```

# STATION ATTACH (DCWRITE Type = 1)

### Required Parameters/Fields

A message parameter and the following are required:

| Word | Field | Value | Description |
|---|---|---|---|
| 0 | [47:08] | 1 | Type. |
| | [34:01] | 0 | Standard pseudostation. |
| | | 1 | Limited pseudostation. |
| | [32:01] | 0 | Reserved for use by SYSTEM/COMS. |
| | [31:01] | 0 | Regular station attach: The MCS provides either the DLS number, LSN, or station name. |
| | | 1 | Allocate pseudostation: The MCS gets the LSN and station name from the operating system. The DLS number does not apply to pseudostations. |
| | [29:06] | | Variant field (description follows): |
| | [29:01] | 0 | If a station is assigned to the file of an object job, all input from that station is sent directly to the file port and all output from the file port is sent directly to the station without intervention required by the MCS. |
| | | 1 | If a station is assigned to the file of an object job, all input from that station is sent directly to the file port and all output from the file port is sent to the current queue of the station. |
| | [28:01] | 0 | All STATION EVENT (Class = 1) messages, exclusive of the control message, are placed in the primary queue of the controlling MCS. |
| | | 1 | All STATION EVENT (Class = 1) messages, exclusive of the control message, are placed in the current queue of the station. |
| | [27:01] | 0 | All NOT READY results are placed in the primary queue. |
| | | 1 | All NOT READY results are placed in the current queue of the station. |
| | [26:01] | 0 | All control messages from the station are placed in the primary queue of the controlling MCS. |
| | | 1 | All control messages from the station are placed in the current queue of the station. Control messages are messages from a station that contains the control character of that station (as set forth in the DATACOMINFO file or allocated by the MCS). |

| Word | Field | Value | Description |
|---|---|---|---|
| | [25:01] | 0 | All ERROR RESULT (Class = 99) messages for the station are placed in the primary queue of the controlling MCS. |
| | | 1 | All ERROR RESULT (Class = 99) messages for the station are placed in the current queue of the station. |
| | [24:01] | 0 | All GOOD RESULTS (Class = 5) messages, except any for NULL STATION REQUEST (DCWRITE Type = 48), DCWRITE and all LINE CHANGE RESULT (Class = 9) messages for the station, are discarded. |
| | | 1 | All GOOD RESULTS (Class = 5) messages and all LINE STATUS CHANGE RESULT (Class = 9) messages for the station are placed in the current queue of the station. |
| | [23:24] | | LSN-FRSN-DLS field (description follows): |
| | [23:01] | 0 | If MSG[0].[22:23] is not 0, then MSG [0].[22:23] designates the LSN. |
| | | 1 | MSG[0].[22:23] contains the relative NSP, line, and station number (DLS number). Also, if MSG[0].[23:24] = 0, then the station name (as given in the DATACOMINFO file) in display form (for example, TTY/ONE) is contained in MSG[6], the text portion of the message. |

## Explanation

One of the mechanisms through which an MCS can gain control of a station is the STATION ATTACH DCWRITE function (refer also to “STATION EVENT Class = 1” and “FILE OPEN Class = 2” messages in the “MCS Result Message Formats” section). The MCS is allowed to use the physical location of the station (DLS number), the name of the station (as declared in the DATACOMINFO file of the installation), or its logical station number (LSN). In any case, the STATION ATTACH DCWRITE function verifies that the calling MCS is the one named in the DATACOMINFO file for that station or that control of the station has been passed to the MCS by another MCS through use of the TRANSFER STATION CONTROL (DCWRITE Type = 45) DCWRITE function.

When using a DLS number for attachment, the relative NSP number must appear in MSG[0].[22:07], the line number in MSG[0].[15:08], and the station number in MSG[0].[07:08]. MSG[0].[23:01] must equal 1 so that the DCWRITE routine understands that attachment is being performed with a DLS number.

Attachment by station name is accomplished by ensuring that MSG[0].[23:24] equals 0 and that the station name in external standard form is placed as text in the allocated message, starting at MSG[6]. Use of station attachment by name (or by LSN) removes the necessity for the MCS to know about the physical location of the station.

The station is not required to have a line assignment if attachment is performed by LSN or station name.

On exit from the DCWRITE routine, the message is nonnull and MSG[0].[22:23] contains the LSN if no errors occurred. The LSN must be used by the MCS for all further DCWRITE calls that affect or reference that station except for two cases in which the file relative station number (FRSN) is used. (Refer to “STATION ASSIGNMENT TO FILE (DCWRITE Type = 64)” and “WRITE TO OBJECT JOB (DCWRITE Type = 65)” in this section.) MSG[1].[23:24] contains the DLS number. MSG[1].[31:08] contains switched status for the line with which the station is associated. (Refer to the discussion of the “SWITCHED STATUS RESULT (Class = 7)” message in the “ MCS Result Message Formats” section). The remainder of the message has the format of an INTERROGATE STATION ENVIRONMENT (Class = 15) result and returns station, terminal, and line information, plus the station name.

If the station for which attachment is desired does not belong to the calling MCS or if the station is currently controlled by another MCS, the value returned by the DCWRITE intrinsic function is 69. At this point, the calling MCS must perform an INTER-MCS COMMUNICATE (DCWRITE Type = 3) DCWRITE function requesting the other MCS to relinquish or allow control of that station to the calling MCS. If the name or number of the controlling MCS is unknown, it can be obtained through use of the INTERROGATE STATION ENVIRONMENT (DCWRITE Type = 4) DCWRITE function.

GOOD INPUT RECEIVED (Class = 0) messages from a station go to the current queue of the station. Disposition of other kinds of results is controlled by the variant field specified in MSG[0].[29:06].

The STATION ATTACH DCWRITE function makes the current queue of the attached station point to the primary queue of the MCS. (Refer to “CHANGE CURRENT QUEUE (DCWRITE Type = 32)” in this section.)

The minimum message size required for this function is six words if attachment by DLS number or LSN is being performed; seven words (or some additional number of words sufficient to contain the station name) are required if attachment by station name is being performed.

The STATION ATTACH DCWRITE function can be used by an MCS to request allocation of a pseudostation from the pseudostation pool. This is designated by setting MSG[0].[31:01] to a value of 1.

If the pool of pseudostations is exhausted and the system is allowed to allocate additional pseudostations, this request can be delayed by up to 5 seconds to allow for the allocation of new pseudostations. To prevent this delay set MSG[0].[33:1] to a value of 1.

When an MCS requests allocation of a pseudostation (that is, MSG[0].[31:01] = 1), MSG[0].[23:24] is ignored. Usually, this field is used by an MCS to designate (by name, LSN, or DLS number) which station to attach. An MCS cannot determine the name or LSN of a pseudostation before it is allocated. Pseudostations do not have DLS numbers.

The type of pseudostation allocated can be further qualified. Normally, a standard pseudostation is allocated with MSG[0] . [34:01] = 0. However, a limited pseudostation can also be allocated by setting MSG[0] . [34:01] to a value of 1. A limited pseudostation cannot be requested if the variant MSG[0] . [24.1] was not set in the Initialize Primary Queue DCWRITE request—the field is ignored and a standard pseudostation is attached. Limited pseudostations are not transferable and always have LSNs greater than 16383 assigned to them.

Upon exiting the DCWRITE routine, if no errors occurred, MSG[0].[22:23] contains the LSN of the pseudostation that has been assigned to the requesting MCS. The rest of the message has the same format as the result message of a regular STATION ATTACH request (that is, it has the format of an INTERROGATE STATION ENVIRONMENT result message).

If all pseudostations in the pool have previously been allocated, a DCWRITE error (value = 193) is returned. If the expansion of the pseudostation pool requires more than 5 seconds or the MCS prohibits the delay, a DCWRITE error (value = 203) is returned.

The STATION ATTACH DCWRITE function is intercepted when performed on a pseudostation that has a fully participating MCS. The DCWRITE function must be performed by the MCS that currently controls the station. The intercepted messages are placed in the primary queue of the fully participating MCS.

### Examples

```
ALLOCATE(MSG,6);
MSG[0] := 0 & 1 [47:8] & 1 [23:1] & NSPNR [22:7]
            & LINENR [15:8] & STANR [7:8];
RESULT := DCWRITE(MSG);
LSN := MSG[0].[22:23];

ALLOCATE(MSG,8);
MSG[0] := 0 & 1 [47:8] & 63 [29:6];
REPLACE POINTER(MSG[6],8) BY "TTY/ONE.";
RESULT := DCWRITE(MSG);
LSN := MSG[0].[22:23];
```

# INTERROGATE MCS (DCWRITE Type = 2)

### Required Parameters/Fields

A message parameter and the following are required:

| Word | Field | Value | Description |
|---|---|---|---|
| 0 | [47:08] | 2 | Type. |
|  | [23:24] | not 0 | The field is assumed to be an MCS number (MCS numbers are values in the range 1 through N inclusive, where N depends on the number of unique MCSs identified in a given DATACOMINFO file. |
|  |  | 0 | The MCS name is assumed to start in MSG[6]. Six EBCDIC characters per word are in external standard form (for example, SYSTEM/CANDE). |

### Explanation

The INTERROGATE MCS DCWRITE function allows an MCS access to certain information concerning any other MCSs in the data comm environment.

The information returned is largely status information, that informs the interrogating MCS of, at least, the following:

- Whether the MCS in question was initiated or is in execution
- If being executed, the mix number of the MCS
- Whether or not the MCS was discontinued with the DS system command
- The MCS number
- The name of the MCS

The result of an MCS INTERROGATE function call appears in the message variable that was passed as a parameter to the DCWRITE intrinsic. The same general format as that for the INTERROGATE STATION ENVIRONMENT RESULT is used with an expanded message format so that the MCS can request additional information with minimal effort.

## Indexing

Figure 5–1 illustrates the path through index words to obtain additional MCS information.

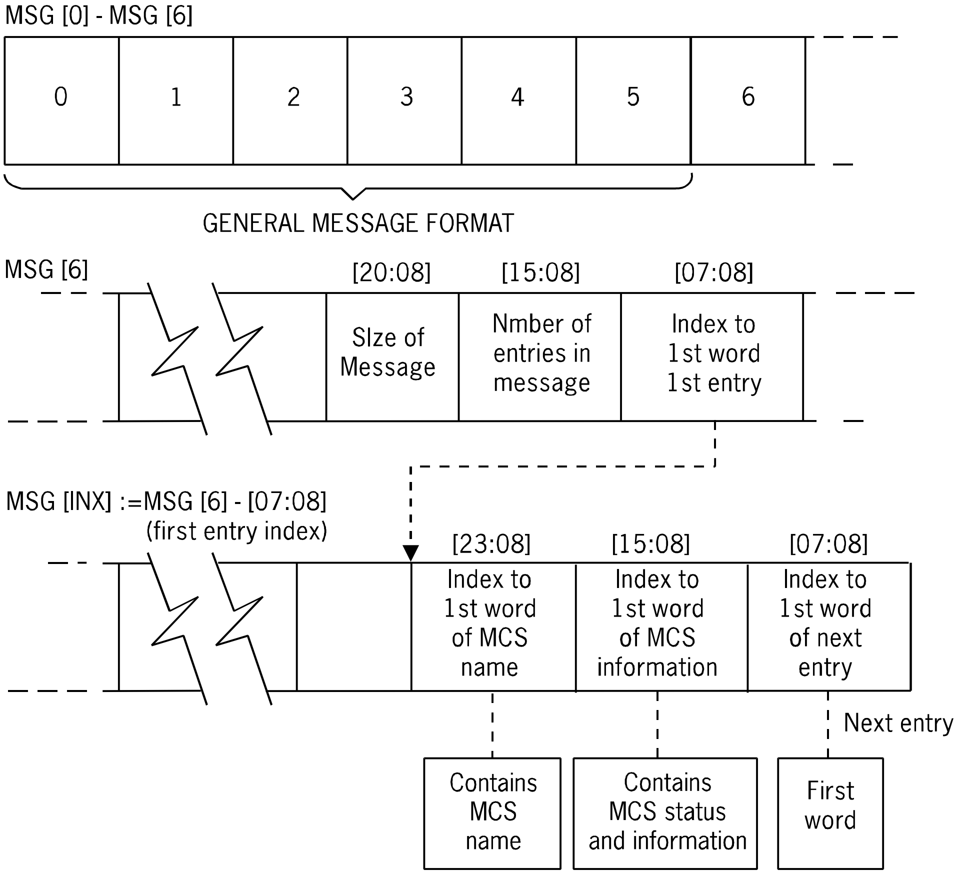

MSG [0] - MSG [6]
0
1
2
3
4
5
6
GENERAL MESSAGE FORMAT
MSG [6]
[20:08]
[15:08]
[07:08]
SIze of Message
Nmber of entries in message
Index to 1st word 1st entry
MSG [INX] :=MSG [6] - [07:08]
(first entry index)
[23:08]
[15:08]
[07:08]
Index to 1st word of MCS name
Index to 1st word of MCS information
Index to 1st word of next entry
Next entry
Contains MCS name
Contains MCS status and information
First word

**Figure 5–1. INTERROGATE MCS Index Diagram**

Word 6 of INTERROGATE MCS holds the index to the first word of the first entry of this message. This first word is referenced as the index word, MSG[INX], in the diagram and in the message format. MSG[INX] holds the indexes to several first words of MCS information, the name, or the index to the first word of the next entry. The bit fields of the words referenced by the indexes provide details of status and information requested. They are listed in the following message format:

| Word | Field | Description |
|---|---|---|
| 0-5 | | Appear as originally presented to the DCWRITE function |
| 6 | [23:08] | Contains the total size of this message (in words) |
| | [15:08] | Contains the number of entries in this message |
| | [07:08] | Contains the index to the first word of the first entry |

## MSG[INX] := MSG[6].[07:08]

This field is as follows:

| Word | Field | Description |
|---|---|---|
| [INX] | [47:08] | Contains the index to the first word of the next entry (0 indicates that this entry is the last entry) |
| | [23:08] | Contains the index to the first word of the MCS name (this entry) |
| | [15:08] | Contains the index to the first word of MCS information (this entry) |

## MSG[MSG[INX].[23:08]]

This field contains the first six EBCDIC characters (or less) of the MCS name in external standard form (carrying over into successive words, if necessary) and ends with a period.

## MSG[MSG[INX].[15:08]]

This field contains the first words of MCS information as follows:

| Field | Value | Description |
|---|---|---|
| Word n | | |
| [47:01] | 1 | The MCS has diagnostic capabilities. |
| [46:01] | 1 | The MCS is initiated/running. |
| [45:01] | 1 | The MCS was discontinued with the DS system command (abnormally terminated). |
| [44:01] | 1 | The MCS required message was displayed for this MCS. |
| [43:01] | 1 | DCRECON has a message for you. |
| [42:01] | 1 | MCS expanded capacity flag |
| [41:18] | | <reserved> |
| [23:08] | | MCS number |
| [15:16] | not 0 | MCS mix number |
| Word n+1 | | MCS mix number |
| Word n+2 | | MCS stack number |
| Word n+3 | | |
| [47:24] | | Number of bits representing file number |
| [23:24] | | Number of bits representing RSN |

If the variant MSG[0] . [24:1] was not set in the Initialize Primary Queue DCWRITE request, then the field MSG[[MSG[INX].[15:08]].[15.16] contains the mix number. Also, the MCS information words n+1 through n+3 are not returned.

If the LSN of the station being interrogated is greater than 16383 and the MCS neither owns or controls that station, error 71 is returned.

### Examples

The following example illustrates an INTERROGATE MCS in which the number of the MCS is used for the reference:

```
ALLOCATE(MSG,6);
MSG[0] := MCSNR & 2 [47:8];
RESULT := DCWRITE(MSG);
```

The following example illustrates an INTERROGATE MCS in which the name of the MCS is used for the reference:

```
ALLOCATE(MSG,9);
MSG[0] := 0 & 2 [47:8];
REPLACE POINTER(MSG[6],8) BY "SYSTEM/CANDE.";
RESULT := DCWRITE(MSG);
```

## INTER-MCS COMMUNICATE (DCWRITE Type = 3)

### Required Parameters/Fields

A message parameter and the following are required:

| Word | Field | Value | Description |
|---|---|---|---|
| 0 | [47:08] | 3 | Type. |
| | [39:16] | | Variant field (description follows): |
| | [25:01] | 0 | Do not override DSed status of target MCS. |
| | | 1 | Override the DSed status of the target MCS, if initiation was requested. |
| | [24:01] | 0 | No initiation is attempted. |
| | | 1 | The system automatically initiates the recipient MCS if it is not currently initiated or running and was not DSed. |
| | [23:24] | | MCS number of recipient MCS. |

### Explanation

The INTER-MCS COMMUNICATE DCWRITE function allows an MCS to establish a communication link to any other MCS.

The calling MCS need only supply the MCS number of the recipient MCS. The MCS number can be obtained through the invocation of the INTERROGATE MCS (DCWRITE Type = 2) DCWRITE function that supplies the name of the desired MCS.

The text portion of the message, if any, can contain any type of formatted information that, through prior arrangement and convention, conveys or represents meaningful information to the recipient MCS. No attempt is made to act on or otherwise interpret the textual portion of these messages by the data comm system.

The calling MCS can be set bit 24 to cause initiation of the recipient MCS in the event that the recipient is not initiated or running at the time. If the recipient MCS is not running due to having been DSed, the calling MCS can be set to bit 25 to override the DSed status of the recipient MCS and cause the initiation of the recipient MCS.

Nonzero functional values are returned if the message cannot be delivered to the primary queue of the recipient MCS.

| Value | Description |
|---|---|
| 102 | MCS could not be executed. |
| 103 | MCS is not running. |

If the codefile for the recipient MCS is not present, this DCWRITE causes a message, MCS REQUIRED, to be displayed and delivers the message into a pending queue for the MCS. A zero functional value is returned.

The recipient MCS receives the result of an INTER-MCS COMMUNICATE as a message in its primary queue. (Refer to the “INTER-MCS COMMUNICATE RESULT (Class = 13)” message in the “MCS Result Message Format” section for format information.)

### Example

```
ALLOCATE(MSG,8);
MSG[0] := MCSNR & 3 [47:8] & 1 [24:1];
REPLACE POINTER(MSG[6],8) BY "HERE I AM";
MSG[2].[39:16] := 10;
RESULT := DCWRITE(MSG);
```

# INTERROGATE STATION ENVIRONMENT (DCWRITE Type = 4)

### Required Parameters/Fields

The minimum message length is seven words. A message parameter and the following are required:

| Word | Field | Value | Description |
|---|---|---|---|
| 0 | [47:08] | 4 | Type. |
| | [39:16] | | Variant field (description follows): |
| | [31:01] | 1 | A terminal name is desired. |
| | [30:01] | 1 | NSP station information is desired. |
| | [29:01] | 1 | NSP line information is desired. |
| | [28:01] | 1 | A station name is desired. |
| | [27:01] | 1 | Line information is desired. |
| | [26:01] | 1 | Terminal information is desired. |
| | [25:01] | 1 | Station information is desired. |
| | [24:01] | 1 | Return information for all stations under control of calling MCS is desired. |
| | | 0 | Interrogate a specific station (see the LSN-FRSN-DLS field description that follows). |
| | [23:24] | | LSN-FRSN-DLS field (only if [24:01] = 0) (a description follows): |
| | [23:01] | 1 | MSG[0].[22:23] is assumed to be a DLS number. |
| | | 0 | If MSG[0].[22:23] is not 0, then MSG[0].[22:23] is an LSN. If MSG[0].[22:23] = 0, then MSG[6] is assumed to be the first word of a station name expressed in external display form (for example "TWXO.") with six EBCDIC characters per word. |

### Optional Parameters/Fields

The queue parameter is optional.

If only line information is desired, the DLS number (NSP in line) is the only required parameter; this information is as follows:

| Word | Field | Value | Description |
|---|---|---|---|
| 0 | [23:01] | 1 | Only line information is desired. |
| | [22:07] | | Relative NSP number. |
| | [15:08] | | Line number; the line number equals the relative LSP number times 16 plus the adapter number. |

In this case, either of the corresponding bits for NSP [29:01] and/or DCC [27:01] information can be turned on in the variant field; the information is returned.

### Explanation

The INTERROGATE STATION ENVIRONMENT DCWRITE function enables an MCS to gain access to certain information concerning the stations that are described in a data comm configuration.

The general flexibility of this function affords an MCS one or more of the following:

- Access to station information (logical attributes) such as the specified retry count, control character, and so forth, of the station.
- Access to terminal information (physical attributes) such as the terminal width, page size, terminal type (a screen device or not), and so forth, of the station.
- Access to line information concerning the line with which the station is associated (for example, whether or not the line is switched and, if it is, whether or not it is connected), and so forth.
- Access to the name of the station.
- Access to the above items either for a specific station or for all stations currently under control of the calling MCS.
- Access to station information that indicates whether or not a station is a pseudostation. It also indicates whether or not the station currently has a fully participating MCS.
- Optional designation of a queue into which the desired information is to be inserted.

The variant field (MSG[0].[39:16]) contains the MCS-specified options that determine the type or types of information to be returned. A special bit ([24:01]) is set aside to be used to designate whether the desired information (designated by the remaining bits in the variant field) is to be returned for one specific station or for all stations currently under control of the calling MCS.

The result information can be returned in the message parameter, placed in the primary queue of the MCS, or placed in the optional queue parameter. When results are placed in a queue, word 6 of the message parameter to the DCWRITE function equals the number of stations for which information was inserted in the queue. All result messages placed in the queue have the format of an INTERROGATE STATION ENVIRONMENT RESULT (Class = 15).

If MSG[0].[24:01] is 0 (interrogate a specific station), the result is returned by the message parameter that was supplied for the call on the DCWRITE function, unless a queue parameter was supplied for the call on the DCWRITE function. (The size of the message on return from the DCWRITE routine might not necessarily be the size of the originally supplied message.) If a queue parameter was supplied for the call on the DCWRITE function, the result is inserted in that queue, activating the queue if necessary. In the case of interrogation of a specific station, the calling MCS can be an MCS other than the controlling MCS.

If MSG[0].[24:01] is 1 (interrogate all stations for the MCS), the result is placed, as one or more messages (Class = 15), in the primary queue of the MCS unless a queue parameter was supplied for the call on the DCWRITE function. The result messages are inserted in the queue, which activates the queue if necessary.

If MSG[0].[39:15] is not equal to 0, the type of information is determined according to which bits are turned on. That is, if a given bit is turned on, the information associated with the meaning of the bit is returned for the interrogated station. These bits can be used in any combination desired. Requests that call for the return of station names should be judiciously invoked because at least one disk access is required for each station name retrieved.

If MSG[0].[39:15] is 0, then by default, only station information is returned. That is, making MSG[0].[39:15] equal to 0 is identically equivalent to making MSG[0].[25:01] equal to 1 and MSG[0].[39:14] equal to 0.

Additionally, if an MCS issues a blanket interrogation and has no station assigned, no Class = 15 messages are returned, and MSG[6] contains 0 to indicate this.

The INTERROGATE STATION ENVIRONMENT DCWRITE function executes in series rather than in parallel with the execution of the calling MCS. Frequent calls on the function, particularly for the purpose of performing blanket interrogations (MSG[0].[24:01] = 1) are not recommended and could significantly impair the total throughput capability of an MCS.

If the variant MSG[0].[24:01] was not set in the Initialize Primary Queue DCWRITE request, then the conditions under which error 71, INVALID LSN, occurs are expanded. If the LSN of the station being interrogated is greater than 16383 and the MCS neither owns or controls that station, error 71 is returned.

### Examples

The following example illustrates an interrogate invocation that returns the station, terminal, and line information for a specific station whose name is “A/B.”. The interrogate result appears in the message parameter that was passed to the DCWRITE intrinsic.

```
ALLOCATE(MSG,7);
MSG[0] := 0 & 4 [47:8] & 1 [25:1]
            & 1 [26:1] & 1 [27:1];
REPLACE POINTER(MSG[6],8) BY "A/B.";
RESULT := DCWRITE(MSG);
```

The following example illustrates a blanket interrogate invocation that returns the station, terminal, and line information as well as the name for each of the stations under the control of the calling MCS. Interrogation result messages are returned by a queue declared as INFOQUE.

```
ALLOCATE(MSG,7);
MSG[0] := 0 & 4 [47:8] & 31 [39:16];
RESULT := DCWRITE(MSG,INFOQUE);
```

# ATTACH SCHEDULE STATION (DCWRITE Type = 5)

### Required Parameters/Fields

A message parameter and the following are required:

| Word | Field | Value | Description |
|---|---|---|---|
| 0 | [47:08] | 5 | Type |

### Explanation

Schedule stations have no predefined MCS assignment and are available for use by any MCS. Before using a schedule station, the MCS must explicitly attach the station. Briefly, a schedule station is defined to have the following attributes:

- A schedule station has no line assignment.
- A schedule station has no predefined, dedicated MCS assignment.
- When used for a remote input (or I/O) file, the controlling MCS must participate in all I/O operations. Additionally, when an object job requests input from a schedule station, a message is sent to the controlling MCS to request that input be sent to the remote file.

After the call on the DCWRITE routine, MSG[0].[23:24] contains the LSN of the assigned schedule station. The LSN is 0 if no schedule stations are available, which indicates that all such stations are currently assigned.

Attachment of a schedule station implicitly turns on the MCS PARTICIPATES option (normally bit [39:01] in the variant field) for the station.

An attached schedule station can be detached (and thereby returned to the system for subsequent assignment to an MCS) by the STATION DETACH (DCWRITE Type = 42) DCWRITE function.

Any DCWRITE functions permissible for stations without a line assignment are allowed for schedule stations, with the following exceptions:

- A schedule station cannot be assigned to a line by a reconfiguration request. Attempting to do so results in DCWRITE error 138 (SCHEDULE STATION MAY NOT HAVE LINE ASSIGNMENT).
- A schedule station cannot be transferred to another MCS. Attempting to do so results in DCWRITE error 139 (SCHEDULE STATION MAY NOT BE TRANSFERRED).
- A schedule station cannot be added to a file by an ADD STATION TO FILE (DCWRITE Type = 67) DCWRITE function. Attempting to do so results in DCWRITE error 140 (SCHEDULE STATION MAY NOT BE ADDED TO A FILE).

When a STATION INTERROGATE (DCWRITE Type = 4) DCWRITE function is performed for a schedule station, the following pertinent information is returned, if requested:

## Station Information

| Information | Value |
|---|---|
| Enabled | TRUE |
| MCS number | Number of the current controlling MCS<br>(0 if not currently assigned to an MCS) |
| Width | 72 |

## Terminal Information

| Information | Value |
|---|---|
| Screen | FALSE |
| In/out | 3 (I/O) |
| Width | 72 |
| Maximum input | 72 |
| Page size | 0 |
| Maximum output | 72 |

In addition, the name of any interrogated schedule station is of the form SCHED#*nnn* where *nnn* is the sequential number of the station, beginning at 1. Because schedule stations are not explicitly declared in the DATACOMINFO file, attempting to interrogate a schedule station by name results in the station not being found.

When a schedule station is assigned to a remote file, the following restrictions apply:

- The controlling MCS must participate in I/O.
- A schedule station cannot be a member of a multistation file.
- Only one input or input/output file can be assigned to a schedule station at a time.

### Example

```
ALLOCATE(MSG,6);
MSG[0] := 0 & 5 [47:8];
RESULT := DCWRITE(MSG);
```

# CHANGE CURRENT QUEUE (DCWRITE Type = 32)

### Required Parameters/Fields

A message parameter, a queue parameter, and the following are required:

| Word | Field | Value | Description |
|---|---|---|---|
| 0 | [47:08] | 32 | Type. |
| | [29:06] | | Variant, as detailed in the STATION ATTACH (DCWRITE Type = 1) DCWRITE. |
| | [23:24] | | LSN or DLS number. |
| 2 | [39:16] | | If text is to be transmitted to the station, this field must contain a byte count. |
| 6 to end | | | Optional text. |

### Explanation

By default, all messages pertaining to a station are returned in the primary queue of the MCS. (The primary queue of the MCS is established through the INITIALIZE PRIMARY QUEUE (DCWRITE Type = 0) DCWRITE function.)

An MCS can elect (optionally) to have all station-related messages other than errors placed in a different queue through use of the CHANGE CURRENT QUEUE DCWRITE function.

By using the variant bits with this DCWRITE type, the controlling MCS is allowed to override the set of default options as outlined in the STATION ATTACH (DCWRITE Type = 1) DCWRITE function.

If MSG[2].[39:16] is not equal to 0, on completing the CHANGE CURRENT QUEUE task, the DCWRITE routine attempts to send the textual portion of the message, subject to any constraints imposed by the WRITE (DCWRITE Type =33) DCWRITE function. After the bit values in the variant field have been noted, the variant field is zeroed so that no conflict exists with the variant field of the WRITE (DCWRITE Type = 33) DCWRITE function, which this DCWRITE function now simulates. Also, the type field is changed to the value 33 before placing this message in the request queue of the appropriate NSP. Thus, in this case, the original DCWRITE type field (MSG[4].[23:24]) in either the GOOD RESULT (Class = 5) message or the ERROR RESULT (Class = 99) message contains the value 33 rather than the value 32. If no text is to be written to the station, the GOOD RESULT message returns the original DCWRITE type value of 32.

On exit from the DCWRITE routine (if all operations were successful and proper), the message is null. If the only objective of the MCS is to change default options (the station-related messages are to continue being placed into the primary queue or a previously established current queue), then the desired options can be turned on or off by naming the queue, which is now in use, as the queue parameter.

If the queue parameter that was passed is inactive, the CHANGE CURRENT QUEUE DCWRITE function activates the queue.

You must supply the LSN or DLS number for the station for this DCWRITE call.

The station specified by LSN does not require a line assignment; however, if text size is not zero and the station has no line assignment, error 88 is returned.

The CHANGE CURRENT QUEUE DCWRITE function is intercepted when performed on a pseudostation that has a fully participating MCS. The DCWRITE function must be performed by the MCS that currently controls the station. The intercepted messages are placed in the primary queue of the fully participating MCS.

### Example

```
ALLOCATE(MSG,13);
MSG[0] := LSN & 32 [47:8] & 1 [25:1];
REPLACE POINTER(MSG[6],8) BY
      "YOU ARE ON-LINE TO SYSTEM D.";
MSG[2].[39:16] := 37;
RESULT := DCWRITE(MSG,CURRENTQ);
```

# WRITE (DCWRITE Type = 33)

### Required Parameters/Fields

A message parameter and the following are required:

| Word | Field | Value | Description |
|---|---|---|---|
| 0 | [47:08] | 33 | Type. |
| | [39:16] | | Carriage control fields (description follows): |
| | [39:08] | | The channel number to skip to or the number of lines to skip (NDLII SKIPCOUNT). |
| | [31:01] | 1 | The tabulation to be done (NDLII TAB). |
| | [30:01] | 1 | Carriage control should be done before text is transmitted (NDLII MOTIONBEFORE). |
| | [29:01] | 1 | More blocks to follow this one (NDLII BLOCKED). |
| | [28:01] | 1 | The value stored in MSG[0].[39:08] is the number of vertical lines that should be skipped (NDLII SPACE). |
| | [27:01] | 1 | The value stored in MSG[0].[39:08] is the channel number to skip to (NDLII SKIPLINE). |
| | [26:01] | 1 | A new page is required for the output device (NDLII NEWPAGE). |
| | [25:01] | 1 | Carriage return is suppressed (NDLII NOCARRIAGERETURN). |
| | [24:01] | 1 | Line feed is suppressed (NDLII NOLINEFEED). |
| | [23:24] | | LSN or DLS number. |
| 1 | [46:07] | | Priority of output. |
| 2 | [39:16] | | Text size field. |
| 4 | [47:24] | | The message number field or, optionally, the file relative station number (FRSN). |
| 6 to end | | | Optional text (If text is to be transmitted to the station, a byte count must be supplied in the text size field MSG[2].[39:16]). |

### Explanation

The WRITE DCWRITE function allows output to be sent to the station indicated by the supplied LSN or DLS number. Carriage control is performed as specified by the variant-field bits if it is implemented in the NDLII algorithm and editor request of the station. If the text size field (MSG[2].[39:16]) is 0, only the carriage control is affected. If the text size field is greater than 0, the text is written to the station.

MSG[1].[46:07] contains the priority of the message. If this field is not zero, the DCC inserts the message into the station queue after any other messages of higher or equal priority but before messages of lower priority. In this way, an MCS causes messages to be transmitted in a different sequence from that given to the DCC. The MCS can produce 128 different levels of priority, with 0 the lowest priority and 127 the highest priority.

Field MSG[4].[47:24] is the message number field. An MCS can elect to assign message numbers to all messages that it allocates in the system, thus providing some ability to audit the flow of its messages in the system. The software system preserves the integrity of this field and does not use it for other purposes. If the MCS receives an OBJECT JOB OUTPUT (Class = 3) message, this field contains the FRSN. The MCS can elect to forward this message to the intended station by using the WRITE DCWRITE function without altering this field. In this case, the LSN or DLS number is still required. The FRSN that this field would contain could be useful to the MCS itself in identifying the file from which the message originated when the MCS is participating in I/O for the station and does a RECALL MESSAGE (DCWRITE Type = 41) DCWRITE function.

The WRITE DCWRITE function is intercepted when performed on a pseudostation that has a fully participating MCS. The DCWRITE function must be performed by the MCS that currently controls the station. The intercepted messages are placed in the primary queue of the fully participating MCS.

### Example

```
ALLOCATE(MSG,9);
MSG[0] := LSN & 33 [47:8] & 1 [26:1];
MSG[2].[39:16] := 14;
REPLACE POINTER(MSG[6],8) BY "THIS IS OUTPUT";
RESULT := DCWRITE(MSG);
```

# READ-ONCE ONLY (DCWRITE Type = 34)

### Required Parameters/Fields

The minimum message size for the READ-ONCE ONLY DCWRITE function is seven words. A message parameter and the following are required:

| Word | Field | Value | Description |
|---|---|---|---|
| 0 | [47:08] | 34 | Type. |
|  | [23:24] |  | LSN or DLS number. |
| 2 | [39:16] |  | Number of bytes of text area available for accumulating input. |
| 6 to 5+N |  |  | A number of words of text area, where N must satisfy N > (MSG[2].[39:16]+5) DIV 6. |

### Explanation

The READ-ONCE ONLY DCWRITE function allows an MCS to request the NSP to accept an input for the station indicated by the LSN or DLS number. The main purpose of this DCWRITE type is to ease the transition from NDL (DCP-based) systems to NDLII (NSP-based) systems.

The first spontaneous input from the station satisfies the READ-ONCE ONLY DCWRITE function. When accumulating input from the station for this particular read, the number of bytes indicated in MSG[2].[39:16] is collected and returned. If the station sends more characters than indicated in this size field, the extra characters are ignored. The station must be enabled for input to satisfy the READ-ONCE ONLY DCWRITE request. After this request is satisfied, subsequent inputs are handled as spontaneous inputs of varying sizes.

The READ-ONCE ONLY DCWRITE function is intercepted when performed on a pseudostation that has a fully participating MCS. The DCWRITE function must be performed by the MCS that currently controls the station. The intercepted messages are placed in the primary queue of the fully participating MCS.

### Example

```
ALLOCATE(MSG,21);
MSG[0] := LSN & 34 [47:8];
MSG[2].[39:16] := (21-6)*6;
RESULT := DCWRITE(MSG);
```

# ENABLE INPUT (DCWRITE Type = 35)

### Required Parameters/Fields

A message parameter and the following are required:

| Word | Field | Value | Description |
|---|---|---|---|
| 0 | [47:08] | 35 | Type. |
| | [39:16] | | Variant field (description follows): |
| | [31:08] | | Value to be stored in the NDLII variable STATION.FREQUENCY. Making this field equal to 255 causes STATION.FREQUENCY to have the default value designated in the DATACOMINFO file. |
| | [23:24] | | LSN or DLS number. |

### Explanation

The ENABLE INPUT DCWRITE function causes the NDLII variable STATION.ENABLED to equal TRUE, and the value of MSG[0].[31:8] to be stored into the NDLII variable STATION.FREQUENCY. The initial value of STATION.ENABLED is determined from the DATACOMINFO file. Thereafter, only the ENABLE INPUT DCWRITE function can turn on STATION.ENABLED and only the DISABLE INPUT (DCWRITE Type = 36) DCWRITE function can turn off STATION.ENABLED. Unless STATION.ENABLED is turned on or an outstanding read request exists (READ-ONCE ONLY (DCWRITE Type = 34) DCWRITE) at the head of the station request queue, no input from the station can be received.

When an NDLII algorithm performs a SENDHOST INPUT, the input message is either routed to the controlling MCS, or routed to an object job if the station is attached to a file, no errors are encountered, the input does not constitute a station event (for example, WRU character received), and the MCS is not participating in object job I/O. The NDLII algorithm has the responsibility for ensuring that input is received whenever the station is enabled and READY.

An NDLII algorithm can use STATION FREQUENCY in any way it is written. However, the intended use of the variable is to influence the rate at which a station is polled.

The ENABLE INPUT DCWRITE function is intercepted when performed on a pseudostation that has a fully participating MCS. The DCWRITE function must be performed by the MCS that currently controls the station. The intercepted messages are placed in the primary queue of the fully participating MCS.

### Example

```
ALLOCATE(MSG,6);
MSG[0] := LSN & 35 [47:8] & 1 [39:16];
RESULT := DCWRITE(MSG);
```

## DISABLE INPUT (DCWRITE Type = 36)

### Required Parameters/Fields

A message parameter and the following are required:

| Word | Field | Value | Description |
|---|---|---|---|
| 0 | [47:08] | 36 | Type. |
|  | [23:24] |  | LSN or DLS number. |

### Explanation

The DISABLE INPUT DCWRITE function is the opposite of the ENABLE INPUT DCWRITE function. Following a DISABLE INPUT request, if the station is a polled station, polling ceases. In any event, unless an outstanding READ request exists, no subsequent input messages are received from the station.

The DISABLE INPUT DCWRITE function is intercepted when performed on a pseudostation that has a fully participating MCS. The DCWRITE function must be performed by the MCS that currently controls the station. The intercepted messages are placed in the primary queue of the fully participating MCS.

### Example

```
ALLOCATE(MSG,6);
MSG[0] := LSN & 36 [47:8];
RESULT := DCWRITE(MSG);
```

## MAKE STATION READY/NOT READY (DCWRITE Type = 37)

### Required Parameters/Fields

A message parameter and the following are required:

| Word | Field | Value | Description |
|---|---|---|---|
| 0 | [47:08] | 37 | Type. |
|  | [39:16] | 0 | Make station NOT READY. |
|  |  | 1 | Make station READY. |
|  | [23:24] |  | LSN or DLS number. |

### Explanation

The MAKE STATION READY/NOT READY DCWRITE function allows the MCS to control the readiness of a station. For example, if a station is being polled because of a READ-ONCE ONLY or ENABLE INPUT DCWRITE function and the MCS desires to have polling or input temporarily suspended, it can issue a MAKE STATION READY/NOT READY request with MSG[0].[39:16] = 0. The station, if operating under an ENABLE INPUT DCWRITE function, ceases to be polled by the NSP until a MAKE STATION READY/NOT READY request with MSG[0].[39:16] = 1 is issued by the MCS.

Because of error recovery requirements, the MCS can be required to perform a MAKE STATION READY/NOT READY request. Refer to “ERROR RESULT (Class = 99)” in the “MCS Result Message Formats” section for messages that use the line/station format. These errors cause the station to become NOT READY.

For example, if the result message from the NSP implies a station error, the MCS must perform a MAKE STATION READY/NOT READY request with MSG[0].[39:16] = 1. This action is required because the NSP makes a station NOT READY if an irrecoverable error is encountered for a station. The execution of a SENDHOST ERROR statement in an NDLII-written algorithm constitutes an irrecoverable error situation. Making the station NOT READY allows the MCS to decide whether to abandon all retry attempts or to allow the station another chance.

If the LINE.BUSY variable is TRUE when a MAKE STATION NOT READY request is issued, the request is deferred until LINE.BUSY is given the value FALSE in the NDLII algorithm. The GOOD RESULT message is generated and delivered to the MCS when the NOT READY action is performed.

The MAKE STATION READY/NOT READY DCWRITE function is intercepted when performed on a pseudostation that has a fully participating MCS. The DCWRITE function must be performed by the MCS that currently controls the station. The intercepted messages are placed in the primary queue of the fully participating MCS.

### Example

```
ALLOCATE(MSG,6);
MSG[0] := LSN & 37 [47:8] & 1 [39:16];
RESULT := DCWRITE(MSG);
```

# SET APPLICATION NUMBER (DCWRITE Type = 38)

### Required Parameters/Fields

A message parameter and the following are required:

| Word | Field | Value | Description |
|---|---|---|---|
| 0 | [47:08] | 38 | Type. |
| | [39:16] | | Application number (must be one of the numbers designated for this section in the DATACOMINFO file). |
| | [23:24] | | LSN or DLS number. |

### Explanation

The SET APPLICATION NUMBER DCWRITE function allows the MCS to change the editor that the NSP algorithms invoke for editing input and output for a station indicated by the LSN or DLS number. The DCWRITE routine verifies that the application number specified by the MCS is a legal and otherwise meaningful value for the station.

The SET APPLICATION NUMBER DCWRITE function is intercepted when performed on a pseudostation that has a fully participating MCS. The DCWRITE function must be performed by the MCS that currently controls the station. The intercepted messages are placed in the primary queue of the fully participating MCS.

The SET APPLICATION NUMBER DCWRITE function causes a CHANGE STATION EDITOR request to be issued to the NSP that uses the editor designated for that application number in the application list. Application numbers are associated with editors in the application list for the station. In addition, if an EXTERNAL station variable called APPLICATION is declared in NDLII, a SET EXTERNAL VARIABLE request is issued to the NSP to make STATION.APPLICATION equal to the application number specified in the DCWRITE request.

### Example

```
ALLOCATE(MSG,6);
MSG[0] := LSN & 38 [47:8] & APPLNO [39:16];
RESULT := DCWRITE(MSG);
```

## SET CHARACTERS (DCWRITE Type = 39)

### Required Parameters/Fields

The minimum message size required for the SET CHARACTERS DCWRITE function is seven words. A message parameter and the following are required:

| Word | Field | Value | Description |
|---|---|---|---|
| 0 | [47:08] | 39 | Type. |
|  | [31:08] | 0 | The control character is turned on. |
|  |  | 1 | The end of message character is turned on. |
|  |  | 2 | The backspace character is turned on. |
|  |  | 3 | The line delete character is turned on. |
|  |  | 4 | The address characters are turned on. |
|  | [32:01] | 1 | The character is reset to the default from the DATACOMINFO file. |
|  | [23:24] |  | LSN or DLS number. |
| 6 |  |  | The character or characters, right-justified. If address characters are to be turned on, then MSG[6].[47:24] contains the receive address characters, right-justified, and MSG[6].[23:24] contains the transmit address characters, right-justified. |

### Explanation

The SET CHARACTERS DCWRITE function allows an MCS to change or restore certain characters used for polling, end of message, backspacing, and so forth. The NDLII algorithm must use these variables in order for them to have an effect. The control character is always dynamic, and the address characters, if any, are always changeable.

Rather than specifying a new value for the character, the MCS can request that the character be restored to its initial value in the DATACOMINFO file by making MSG[0].[32:01] equal to 1. In that case, MSG[6] does not contain the new value; rather, it is obtained by reading the DATACOMINFO file station record.

The SET CHARACTERS DCWRITE function is intercepted when performed on a pseudostation that has a fully participating MCS. The DCWRITE function must be performed by the MCS that currently controls the station. The intercepted messages are placed in the primary queue of the fully participating MCS.

### Example

```
ALLOCATE(MSG,7);
MSG[0] := LSN & 39 [47:8] & 1 [31:8];
MSG[6] := 4"0D";
RESULT := DCWRITE(MSG);
```

# SET TRANSMISSION NUMBER (DCWRITE Type = 40)

### Required Parameters/Fields

The minimum message size required for the SET TRANSMISSION NUMBER DCWRITE function is seven words. A message parameter and the following are required:

| Word | Field | Value | Description |
|---|---|---|---|
| 0 | [47:08] | 40 | Type. |
| | [25:01] | 1 | The transmit transmission number is turned on. |
| | [24:01] | 1 | The receive transmission number is turned on. |
| | [23:24] | | LSN or DLS number. |
| 6 | [47:24] | | Transmit transmission number value (if it is to be turned on). |
| | [23:24] | | Receive transmission number value (if it is to be turned on). |

### Explanation

The SET TRANSMISSION NUMBER DCWRITE function allows an MCS to reinitialize the transmit or receive transmission number (or both) for a particular station. The numbers must be represented as EBCDIC digits, and, for terminals with transmission numbers of one or two digits in length, the numbers must be placed, right-justified, in the appropriate fields.

The SET TRANSMISSION NUMBER DCWRITE function is intercepted when performed on a pseudostation that has a fully participating MCS. The DCWRITE function must be performed by the MCS that currently controls the station. The intercepted messages are placed in the primary queue of the fully participating MCS.

### Example

```
ALLOCATE(MSG,7);
MSG[0] := LSN & 40 [47:8] & 1 [24:1];
MSG[6] := "000";
RESULT := DCWRITE(MSG);
```

# RECALL MESSAGE (DCWRITE Type = 41)

### Required Parameters/Fields

The station or line must be in a NOT READY state. A DCWRITE result value of 84 is returned if the station is READY. A message parameter and the following are required:

| Word | Field | Value | Description |
|---|---|---|---|
| 0 | [47:08] | 38 | Type. |
|  | [39:16] |  | Application number (must be one of the numbers designated for this section in the DATACOMINFO file). |
|  | [23:24] |  | LSN or DLS number. |

### Explanation

The RECALL MESSAGE DCWRITE function allows an MCS to request the NSP to return either the first message or all messages in the station queue for a specific station (LSN or DLS number). The station or line must be in a NOT READY state at the time the request is made. Thus, the MCS must have explicitly requested that the station be made NOT READY (with the MAKE STATION READY/NOT READY DCWRITE function), or the station must be in a NOT READY state as the result of a previous station or line error. If the MCS desires to have only the first message in the station queue returned, then MSG[0].[39:16] must be equal to 0. If all messages in the station queue are to be returned, then MSG[0].[39:16] must be equal to 1.

If the MCS has established that it is to receive all GOOD RESULTS (Class = 5) messages through one of the following, then the MCS receives, in the current queue of the station, a GOOD RESULTS (Class = 5) message acknowledging the RECALL MESSAGE DCWRITE function:

- The variant field of the STATION ATTACH (DCWRITE Type = 1) DCWRITE function
- The CHANGE CURRENT QUEUE (DCWRITE Type = 32) DCWRITE function
- The STATION ASSIGNMENT TO FILE (DCWRITE Type = 64) DCWRITE function that it is to receive all GOOD RESULTS (Class = 5) messages

If no message was in the station queue, the GOOD RESULTS (Class = 5) message contains a value of 0 in MSG[0].[39:16]. If any message was to return (and is returned), the GOOD RESULTS (Class = 5) message contains a value of 1 in MSG[0].[39:16] and the message is returned in the current queue of the station as a RECALL MESSAGE (Class = 6) message. The original DCWRITE type and variant are found in MSG[4].[23:24] of each returned message.

The RECALL MESSAGE DCWRITE function is intercepted when performed on a pseudostation that has a fully participating MCS. The DCWRITE function must be performed by the MCS that currently controls the station. The intercepted messages are placed in the primary queue of the fully participating MCS.

### Example

```
ALLOCATE(MSG,6);
MSG[0] := LSN & 41 [47:8] & 1 [39:16];
RESULT := DCWRITE(MSG);
```

## STATION DETACH (DCWRITE Type = 42)

### Required Parameters/Fields

A message parameter and the following are required:

| Word | Field | Value | Description |
|---|---|---|---|
| 0 | [47:08] | 42 | Type |
| | [39:16] | 0 | Detach Station |
| | | 1 | Retract Station |
| | | 2 | Retract Station and Reset Name |
| | | 3 | Prepare for Detach |
| | [23:24] | | LSN or DLS number |

### Explanation

The STATION DETACH DCWRITE function allows an MCS to detach from a station to which it is currently attached. The STATION DETACH message can also be used by an MCS to detach a pseudo station.

Four variants of the STATION DETACH command are available:

- Detach Station
- Retract Station
- Retract Station and Reset Name
- Prepare for Detach

For each variant there are three cases. In each case, the calling message control system (MCS) is one of the following:

- Both the controlling and the originating MCS
- The controlling but not the originating MCS
- The originating but not the controlling MCS

The controlling MCS is the MCS associated with the primary queue of the station. The originating MCS is the MCS that either originally owned (as defined in the DATACOMINFO file) a physical station or originally allocated a pseudo station.

If an MCS requesting detachment is neither controlling nor originating, the DCWRITE error STATION NOT YOURS (69) is returned.

The following paragraphs explain the possible STATION DETACH variant:

## Standard Detach

The following paragraphs describe the possible cases for the standard Detach variant.

### Controlling and Originating

In this case, the station is marked unattached and no longer has a current or primary queue. If the station is a pseudo station, the pseudostation is deallocated and returned to the pool of available pseudo stations.

If the station is in use by a remote file when the detachment is requested, the DCWRITE error STATION ALREADY IN FILE (82) is returned.

### Controlling but not Originating

In this case, control of the station is returned to the originating MCS. The originating MCS is notified by the Transfer Station DCRESULT (class = 16).

If the station is in use by a remote file when the detachment is requested, the DCWRITE error STATION ALREADY IN FILE (82) is returned.

### Not Controlling but Originating

In this case, the station is marked unattached and no longer has a current or primary queue. If the station is a pseudo station, the pseudostation is deallocated and returned to the pool of available pseudo stations. The controlling MCS is notified by the NSP Terminated DCRESULT (class = 14) with the Detached by Fully-participating MCS variant (value = 1).

If the station is in use by a remote file when the detachment is requested, the remote file is closed and the next I/O operation to the station causes an end-of-file action.

## Retract Station

The following possible cases are for the Retract Detach variant.

### Controlling and Originating

In this case, the DCWRITE error STATION ALREADY YOURS (70) is returned.

### Controlling but not Originating

This case is the same as Standard Detach.

### Not Controlling but Originating

In this case, control of the station is returned to the originating MCS. The controlling MCS is notified by the network support processor (NSP) Terminated DCRESULT (class = 14) with the Detached by Fully-participating MCS variant (value = 1). The originating MCS is notified by the Transfer Station DCRESULT (class = 16).

If the station is in use by a remote file when the detachment is requested, the remote file is closed and the next I/O operation to the station causes an end-of-file action.

### Retract and Reset Pseudostation Name

The following paragraphs describe the possible cases for the Retract and Reset Pseudostation Name variant.

#### Controlling and Originating

Although this case returns an error, if the station is a pseudostation then the pseudostation name is reset to its initial value of PSEUDO0<XXXX>, where <XXXX> is the relative pseudo number of the pseudo station.

The DCWRITE error STATION ALREADY YOURS (70) is returned.

#### Controlling but not Originating

This case is the same as Standard Detach.

#### Not Controlling but Originating

In this case, control of the station is returned to the originating MCS. If the station is a pseudo station, the pseudostation name is reset to its initial value of PSEUDO0<XXXX>, where <XXXX> is the relative pseudo number of the pseudo station. The controlling MCS is notified by the NSP Terminated DCRESULT (class = 14) with the Detached by Fully-participating MCS variant (value = 1). The originating MCS is notified by the Transfer Station DCRESULT (class = 16).

If the station is in use by a remote file when the detachment is requested, the remote file is closed and the next I/O operation to the station causes an end-of-file action.

### Prepare for Detach

The Prepare for Detach variant of the Detach DCWRITE marks the station as detach pending. An MCS requesting detachment when the station is marked detach pending must use the Standard Detach variant (value = 0). If an MCS requests detachment with either the Retract Station variant (value = 1) or the Retract and Reset Pseudostation Name variant (value = 2) and the station is marked detach pending, the DCWRITE error IMPROPER VARIANT FIELD (141) is returned. If an MCS requests detachment with the Prepare for Detach variant (value = 3) and the station is marked detach pending, the DCWRITE error ALREADY DONE (204) is returned.

Only the Station Detach and Interrogate Station Environment DCWRITE functions are allowed for stations that are marked detach pending. All other DCWRITE functions fail with the DCWRITE error STATION IS DETACHING (205).

The following paragraphs describe the possible cases for the Prepare for Detach variant:

### Controlling and Originating

If the station is a pseudo station, the station name is reset to its initial value of PSEUDO0<XXXX>, where <XXXX> is the relative pseudo number of the pseudo station. The station is marked detach pending. The NSP Terminated DCRESULT (class = 14) with the Detached by Prepare variant (value = 3) is queued in the caller's primary queue.

If the station is in use by a remote file when the detachment is requested, the remote file is closed and the next I/O operation to the station causes an end-of-file action.

### Controlling but not Originating

In this case, the DCWRITE error IMPROPER VARIANT FIELD (141) is returned.   Not Controlling but Originating Control of the station is returned to the originating MCS. If the station is a pseudo station, the pseudostation name is reset to its initial value of PSEUDO0<XXXX>, where <XXXX> is the relative pseudo number of the pseudo station. The station is marked detach pending. The controlling MCS is notified by the NSP Terminated DCRESULT (class = 14) with the Detached by Fully-participating MCS variant (value = 1). The originating MCS is also notified by the NSP Terminated DCRESULT (class = 14) except the variant field is Detached by Prepare (value = 3).

If the station is in use by a remote file when the detachment is requested, the remote file is closed and the next I/O operation to the station causes an end-of-file action.

## Example

```
ALLOCATE(MSG,6);
MSG[0] := LSN & 42 [47:8];
RESULT := DCWRITE(MSG);
```

## SET/RESET LOGICALACK (DCWRITE Type = 43)

### Required Parameters/Fields

A message parameter and the following are required:

| Word | Field | Value | Description |
|---|---|---|---|
| 0 | [47:08] | 43 | Type. |
|  | [24:01] | 0 | Turn off LOGICALACK. |
|  |  | 1 | Turn on LOGICALACK. |
|  | [23:24] |  | LSN or DLS number. |

### Explanation

This DCWRITE type allows an MCS to modify the station variable INPUTACTION in the NSP, opting (or not) for the following action by the NSP. After input is successfully received from the station, the execution of an NDLII SENDHOST INPUT statement (if INPUTACTION = SENDANDWAIT) delivers that input to the system, with the TO BE ACKNOWLEDGED flag (MSG[1].[15:01]) turned on. This input is in the form of the GOOD INPUT RECEIVED (Class = 0) message. All further action is suspended on the line until an ACKNOWLEDGE (DCWRITE Type = 44) DCWRITE request is presented to the NSP.

The LOGICALACK capability requires MCS participation. Thus, if a station has INPUTACTION equal to SENDANDWAIT, and if the station is attached to an object job file, the MCS must participate in I/O for that object job file.

The SET/RESET LOGICALACK DCWRITE request is intercepted when performed on a pseudostation that has a fully participating MCS. The DCWRITE function must be performed by the MCS that currently controls the station. The intercepted messages are placed in the primary queue of the fully participating MCS.

### Example

```
ALLOCATE(MSG,6);
MSG[0] := LSN & 43 [47:8] & 1 [24:1];
RESULT := DCWRITE(MSG);
```

# ACKNOWLEDGE (DCWRITE Type = 44)

### Required Parameters/Fields

The line and station must be in a to-be-acknowledged state. A message parameter and the following are required:

| Word | Field | Value | Description |
|---|---|---|---|
| 0 | [47:08] | 44 | Type. |
| | [39:16] | | Variant field, not used. |
| | [23:24] | | LSN or DLS number. |

### Explanation

The ACKNOWLEDGE DCWRITE function causes the NSP to resume execution of the NDLII algorithm for the line and its station where execution had been suspended as the result of a SENDHOST INPUT statement with INPUTACTION SENDANDWAIT. If the line and station are not in a to-be-acknowledged state, an ERROR RESULT (Class = 99) message is generated with a value of 13 in the result byte index field (MSG[1].[47:08]).

The ACKNOWLEDGE DCWRITE function is intercepted when performed on a pseudostation that has a fully participating MCS. The DCWRITE function must be performed by the MCS that currently controls the station. The intercepted messages are placed in the primary queue of the fully participating MCS.

### Example

```
ALLOCATE(MSG,6);
MSG[0] := LSN & 44 [47:8];
RESULT := DCWRITE(MSG);
```

# TRANSFER STATION CONTROL (DCWRITE Type = 45)

### Required Parameters/Fields

The minimum message size is seven words. A message parameter and the following are required:

| Word | Field | Value | Description |
|---|---|---|---|
| 0 | [47:08] | 45 | Type. |
| | [39:16] | | Variant field, as follows: |
| | [31:01] | | BNA transfer bit. |
| | | 0 | The station is transferred to the designated MCS. |
| | | 1 | The station is transferred to the BNA station transfer MCS, SYSTEM/STATION/TRANSFER. The host name must be supplied and the MCS name is optional. Word 6.[47:12] supplies the index to the Station Transfer Index Control Word (INX = MSG[6].[47:12]). The host name and MCS name are indexed by MSG[INX].[11:12] and MSG[INX].[23:12] respectively. |
| | [30:01] | 0 | No information is to be passed in the station transfer control result to the MCS to which the station is to be transferred. |
| | | 1 | Information is to be passed in the station transfer control result to the MCS to which the station is to be transferred. |
| | [29:01] | 1 | The station being transferred is a pseudostation that represents an ODT. The ODT unit number must be present in MSG[6].[47:24]. Control is to be maintained over any output to that station. This transfer is allowed only by the MCS that is designated as the COMS/ODT/DRIVER. |
| | [28:01] | 0 | Full participation is not requested. |
| | | 1 | Full participation is requested: The MCS elects to receive intercepted DCWRITE requests as they are issued by the MCS to which control is being transferred. |
| | [27:01] | | Old BNA transfer bit. This is supported for migration only and will be deimplemented. |
| | | 0 | The station is transferred to the designated MCS. |
| | | 1 | The station is transferred to the BNA station transfer MCS, SYSTEM/STATION/TRANSFER. The host name must be supplied, and the MCS name is optional. The host name and MCS name are indexed by MSG[6].[07:08] and MSG[6].[15:08], respectively. |

| Word | Field | Value | Description |
|---|---|---|---|
| | [26:01] | 1 | Allows an MCS to transfer a station to another MCS and still maintain control over any output to that station. |
| | [25:01] | 1 | Allows an MCS to transfer a station under its control to the original controlling MCS defined in the DATACOMINFO file. |
| | [24:01] | 0 | No initiation is attempted for the recipient MCS. |
| 2 | [39:16] | | The size in bytes of the information to be passed (if word 0 field [30:01] = 1). |

## Explanation

The TRANSFER STATION CONTROL DCWRITE function allows an MCS to transfer any of the stations under its control to any other MCS.

The station to be transferred must be attached to its MCS at the time of its transfer. The recipient MCS need not be currently in execution for the transfer to take place. If the recipient MCS has been initiated or is in execution at the time of the transfer, the recipient receives a result message that serves to inform that MCS that it has inherited a new station. If the recipient MCS is not initiated or running at the time of the transfer, the station is marked as detached (not attached). Whether or not the recipient MCS is running or initiated at the time of the transfer, the current status of the station remains unaltered (if the station is NOT READY before the transfer, it remains NOT READY after the transfer).

The result message received by the recipient MCS (in its primary queue) has the same essential form as the INTERROGATE STATION ENVIRONMENT RESULT (Class = 15) message. (Refer to "TRANSFER STATION CONTROL RESULT (Class = 16)" in the "MCS Result Message Formats" section for details.)

A variant of this function allows the calling MCS to cause initiation of the recipient MCS in the event that the recipient is not initiated or running at the time of the transfer. The TRANSFER STATION CONTROL RESULT (Class = 16) message is placed in the primary queue of the recipient if the variant form is employed.

Another variant permits the MCS to transfer the station to its original controlling MCS specified in the DATACOMINFO file without knowing which MCS that is. However, if this variant is used to attempt a station transfer by the original controlling MCS specified in the DATACOMINFO file, then no action occurs.

In case of errors, the station remains assigned to the calling MCS.

Transferring a station from one MCS to another while maintaining partial control is allowed. This variant is invoked by turning on bit 26 in word 0 of the message. The number of the MCS making the request is stored in word 2 of the station table at PSEUDOMCSNRF ([41:06]). The number of the transferred MCS is stored in word 0 at STAMCSNRF ([37:08]).

If the station is transferred a second time by the new controlling MCS, the PSEUDOMCSNRF field contains the original controlling MCS number, allowing transfers to be made from one remote MCS to another without returning to the original controlling remote MCS each time. If the station is transferred back to the original controlling MCS, the PSEUDOMCSNRF is returned to its initial value of 0. Output that is directed to this station by either a program using a remote station or by the controlling MCS (the one to which the station was transferred, whose MCS number is contained in the STAMCSNRF field) is intercepted and delivered to the MCS whose number is contained in the PSEUDOMCSNRF field as an INTERCEPTED MESSAGE (Class = 29) result.

Transfer of station control to another MCS is not maintained over a halt/load or through a complete shut-down and subsequent reinitialization of data comm. In either case, the control of any stations that were transferred reverts to the MCSs originally designated in the DATACOMINFO file.

The TRANSFER STATION CONTROL DCWRITE function can also be used by an MCS to transfer control of a pseudostation to another MCS. At the time of the transfer, the transferring MCS can request full participation.

Under full participation, the first MCS (the fully participating MCS) can take part in the data comm functions performed on the pseudostation by the second MCS (the controlling MCS). Whenever the controlling MCS issues certain DCWRITE requests, the request messages are intercepted, transformed to INTERCEPTED MESSAGE (Class = 29) results, and placed in the primary queue of the fully participating MCS.

Full participation can be requested only for a pseudostation. If requested for an ordinary station, a DCWRITE error is returned.

Full participation can be requested by only one MCS for each pseudostation. If full participation is requested by an MCS but another MCS has already requested full participation, a DCWRITE error is returned.

If an attempt is made to transfer a limited pseudostation, a DCWRITE error is returned.

When control of a station is transferred to an MCS, that MCS receives a TRANSFER STATION CONTROL (Class = 16) result message in its primary queue. That message has the same form as the STATION ENVIRONMENT (Class=15) result message. Fields in the message can be used by the controlling MCS to determine whether or not the station is a pseudostation and whether or not the station has a fully participating MCS.

A check of the BNA transfer bits is performed to determine whether one of the following bits is on:Word 0.[27:01]This is supported for migration only and will be deimplemented. When this bit is on, a host name and an MCS name index is supplied in word 6.[7:08] and word 6.[15:08] respectively. A string value located at MSG[INX], where INX is the value of the host name or MCS name index, is terminated by a period (.).The host name must be supplied, and the MCS name is optional.Word 0.[31:01]When this bit is on, an index to a Station Transfer Index Control Word is supplied in word 6.[47:12] (INX = MSG[6].[47:12]). MSG[INX].[11:12] contains the index to the host name structure and MSG[INX].[23:24] contains the index to the MCS name structure.

The host name must be supplied, and the MCS name is optional. A value of 0 in the MCS name index field indicates that no MCS name was supplied.

## MSG[0].[31:01] = 0

If MSG[0].[31:01] = 0, the following information is valid:

| Word | Value | Description |
|---|---|---|
| 6 | [11:12] | MCS number of the recipient MCS. |
| 7 | | The first word of information to be passed (if word 0 field [30:01] = 1). |

## MSG[0].[31:01] = 1

If MSG[0].[31:01] = 1, the following information is valid:

| Word | Value | Description |
|---|---|---|
| 6 | [47:12] | INX: = Index to station transfer index control word. |
| MSG[INX] | [23:12] | Index to MCS name. |
| | | @[23:12].[47:16] - Length of MCS name in bytes. |
| | | @[23:12].[32:08] - First character of MCS name. |
| MSG[INX] | [11:12] | Index to host name. |
| | | @[11:12].[47:16] - Length of host name in bytes. |
| | | @[11:12].[32:08] - First character of host name. |
| 7 | | The first word of information to be passed (if word 0 field [30:01] = 1). |

## MSG[0].[27:01] = 0

If MSG[0].[27:01] = 0, the following information is valid:

| Word | Value | Description |
|---|---|---|
| 6 | [07:08] | MCS number of the recipient MCS. |
| 7 | | The first word of information to be passed (if word 0 field [30:01] = 1). |

## MSG[0].[27:01] = 1

If MSG[0].[27:01] = 1, then the setting of MSG[0].[30:01] is ignored, and the following information is valid:

| Word | Value | Description |
|---|---|---|
| 6 | [15:08] | Index to the program name. The program name must terminate with a period (.). |
|  | [07:08] | Index to the host name. The host name must terminate with a period (.). |

### Example

```
ALLOCATE(MSG,7);
MSG[0] := LSN & 45 [47:8] & 1 [24:1];
MSG[6] := MCSNR;
RESULT := DCWRITE(MSG);
```

# WRITE AND RETURN (DCWRITE Type = 46)

### Required Parameters/Fields

The required parameters are identical to those for the WRITE (DCWRITE Type = 33) DCWRITE function, except that Type (MSG[0].[47:08]) has the value 46.

### Explanation

The WRITE AND RETURN DCWRITE function is exactly the same as the WRITE (DCWRITE Type = 33) DCWRITE function except that the result message for the output request is unconditionally returned to the current queue of the MCS. This DCWRITE type can be used instead of the NULL STATION REQUEST (DCWRITE Type = 48) DCWRITE function to limit the number of output messages sent at one time. The format of the message parameter is identical to that for the WRITE DCWRITE function except that MSG[0].[47:8] = 46.

The priority of the message produced by the MCS is contained in this field. The MCS can produce 128 levels of priority, with 0 as the lowest priority and 127 the highest. If the priority level is not zero, the NSP inserts the message into the station queue following messages of higher or equal priority and preceding messages of lower priority. In this way, MCS messages are transmitted in a high-to-low priority sequence rather than the order in which the messages were received by the NSP.

The WRITE AND RETURN DCWRITE function is intercepted when performed on a pseudostation that has a fully participating MCS. The DCWRITE function must be performed by the MCS that currently controls the station. The intercepted messages are placed in the primary queue of the fully participating MCS.

### Example

```
ALLOCATE(MSG,8);
MSG[0] := LSN & 46 [47:8] & 1 [26:1];
MSG[2] := 0 & 8 [39:16];
REPLACE POINTER(MSG[6],8) BY "LAST ONE";
RESULT := DCWRITE(MSG);
```

# NULL STATION REQUEST (DCWRITE Type = 48)

### Required Parameters/Fields

A message parameter and the following are required:

| Word | Field | Value | Description |
|---|---|---|---|
| 0 | [47:08] | 48 | Type. |
|  | [39:16] | 0 | This message is to be returned to the current queue for the station. |
|  |  | 1 | This message is to be returned to the primary queue of the MCS. |
|  | [23:24] |  | LSN or DLS number. |

### Explanation

The NULL STATION REQUEST DCWRITE function causes a message to be placed in the appropriate station queue so that, on encountering this message, the DCC performs no other action than to return the message to the primary or current queue of the station as a GOOD RESULT (Class = 5) message with an original DCWRITE type of 48. This message is returned to the primary or current queue of the station regardless of whether or not the MCS has indicated that it is to receive Class = 5 and Class = 9 messages. This DCWRITE type allows the MCS to insert a marker to signal the end of a series of outputs or a batch within a series of outputs to a station.

The NULL STATION REQUEST DCWRITE request is intercepted when performed on a pseudostation that has a fully participating MCS. The DCWRITE function must be performed by the MCS that currently controls the station. The intercepted messages are placed in the primary queue of the fully participating MCS.

### Example

```
ALLOCATE(MSG,6);
MSG[0] := LSN & 48 [47:8] & 1 [39:16];
RESULT := DCWRITE(MSG);
```

## SET/RESET SEQUENCE MODE (DCWRITE Type = 49)

### Required Parameters/Fields

The minimum message size required for the SET/RESET SEQUENCE MODE DCWRITE function is eight words. A message parameter and the following are required:

| Word | Field | Value | Description |
|---|---|---|---|
| 0 | [47:08] | 49 | Type. |
|  | [39:16] | 0 | The sequence mode is to be turned off. |
|  |  | 1 | The sequence mode is to be turned on. |
|  | [23:24] |  | LSN or DLS number. |

## MSG[0].[39:16] = 1

If MSG[0].[39:16] = 1, the following information is valid:

| Word | Value | Description |
|---|---|---|
| 2 | [39:16] | Maximum number of digits allowed for sequence number 0 < MSG[2].[39:16] < 9. |
| 6 | [26:27] | Base (starting) sequence number value. |
| 7 | [26:27] | Increment value. |

### Explanation

The SET/RESET SEQUENCE MODE DCWRITE function allows an MCS to request the NSP to turn the SEQUENCE toggle on or off for use in NDLII algorithms and to provide the base (or starting) sequence number, increment value, and maximum number of digits allowed for the sequence number.

DCWRITE error 99 is returned if the station is not capable of performing automatic sequencing.

The SET/RESET SEQUENCE MODE DCWRITE function is intercepted when performed on a pseudostation that has a fully participating MCS. The DCWRITE function must be performed by the MCS that currently controls the station. The intercepted messages are placed in the primary queue of the fully participating MCS.

### Example

```
ALLOCATE(MSG,8);
MSG[0] := LSN & 49 [47:8] & 1 [39:16];
MSG[2].[39:16] := 8;
MSG[6] := 1000;
MSG[7] := 1000;
RESULT := DCWRITE(MSG);
```

## WRITE TO TRANSFERRED STATION (DCWRITE Type = 53)

### Required Parameters/Fields

A message parameter and the following are required:

| Word | Field | Value | Description |
|---|---|---|---|
| 0 | [47:08] | 53 | Type. |
| | [39:16] | 0 | A GOOD INPUT RECEIVED (Class = 0) message is forwarded to the MCS (STAMCSNRF) in control of the requested LSN. |
| | | 1 | A STATION EVENT (Class = 1) message is forwarded to the MCS (STAMCSNRF) in control of the requested LSN. |
| | | 2 | The result of a write from the transferred station is to be returned either to the controlling MCS or to logical I/O by a call on DCIOFINISH. |
| | | 3 | The message to DCWRITE is turned into a GOOD RESULTS (Class = 5) message. |
| | [23:24] | | LSN or DLS number. |
| 6 to end | | | If MSG[0].[39:16] = 2, these words contain the 6-word result message to be forwarded. |

### Explanation

The WRITE TO TRANSFERRED STATION DCWRITE function allows an MCS that has partially transferred control of a station to direct result messages to either the transferred station or its controlling MCS.

If the variant field (MSG[0].[39:16]) contains a value of 0 or 1, the message is forwarded with the MSG[0] word modified to reflect either a GOOD INPUT RECEIVED (Class = 0) or a STATION EVENT (Class = 1) result, a variant of 0, and the requested LSN in MSGUNITF.

If the variant field contains a value of 2, the message forwarded is in the text portion of the WRITE TO TRANSFERRED STATION (DCWRITE Type = 53) DCWRITE function message and is 6 words in length. If the original message was of type OBJECTJOBOUTPUT, this result is returned to logical I/O or it is returned to the MCS in control of the transferred station.

If the variant field contains a value of 3, the message is turned into a GOOD RESULTS message by changing the Type field. If the STATION ATTACH (DCWRITE Type = 1)

DCWRITE function that attached the station included a results request, or if the original DCWRITE request was a WRITE AND RETURN (DCWRITE Type = 46) DCWRITE function, then the message is placed in either the primary or current queue; otherwise, it is deallocated.

### Example

```
ALLOCATE(MSG,7);
MSG[0] := LSN & 53 [47:8] & 1 [39:16];
RESULT := DCWRITE(MSG);
```

# SEND MCS RESULT MESSAGE (DCWRITE Type = 55)

### Required Parameters/Fields

The minimum message size for this request is six words.

| Word | Field | Value | Description |
|---|---|---|---|
| 0 | [47:08] | 55 | Send MCS result message. |
|  | [22:23] |  | LSN. |
| 3 | [47:24] |  | Original result type. |
| 6 to end |  |  | Text (if any) of original MCS result message. |

### Explanation

An MCS uses the SEND MCS RESULT MESSAGE DCWRITE function to forward an MCS result message to another MCS. When this DCWRITE request is received, the data comm subsystem removes the header and places the specified result (word 6 through the end of the message), without alteration, in the primary queue of the MCS that currently controls the specified station.

This DCWRITE request has been designed to allow an MCS that has requested full participation for a pseudostation to forward MCS result messages to the MCS that is currently controlling that station. After receiving an intercepted DCWRITE request in its primary queue, the fully participating MCS normally sends a corresponding result message back to the controlling MCS.

This DCWRITE request is valid only on a transferred pseudostation that has a fully participating MCS. If the station is not a pseudostation, or the station has no fully participating MCS, a DCWRITE error is returned. If the MCS that issues this request is not the fully participating MCS, a DCWRITE error is returned.

## SET PSEUDOSTATION ATTRIBUTES (DCWRITE Type = 56)

### Required Parameters/Fields

The minimum message size for this request is eight words, and the maximum size is 52 words if the pseudostation title is updated.

| Word | Field | Value | Description |
|---|---|---|---|
| 0 | [47:08] | 56 | Type. |
| | [39:16] | | Variant field, as follows: |
| | [39:01] | 1 | SCREEN to be updated. |
| | [38:01] | 1 | MYUSE to be updated. |
| | [37:01] | 1 | WIDTH to be updated. |
| | [36:01] | 1 | PAGE to be updated. |
| | [35:01] | 1 | MAXINPUT to be updated. |
| | [34:01] | 1 | MAXOUTPUT to be updated. |
| | [33:01] | 1 | SEQUENCEMODECAPABLE to be updated. |
| | [32:01] | 1 | WRAPAROUNDCAPABLE to be updated. |
| | [31:01] | 1 | Pseudostation title to be updated. |
| | [22:23] | | LSN. |
| 6 | [47:01] | | SCREEN attribute, as follows: |
| | | 0 | Terminal is not a screen device (CRT). |
| | | 1 | Terminal is a screen device (CRT). |
| | [46:02] | | MYUSE attribute, as follows: |
| | | 1 | IN. |
| | | 2 | OUT. |
| | | 3 | I/O. |
| | [44:05] | | Not in use: filler to byte boundary. |
| | [39:16] | | LINEWIDTH attribute: bytes per line. |
| | [23:16] | | PAGESIZE attribute: lines per page. |
| 7 | [47:16] | | MAXINPUT. |
| | [31:16] | | MAXOUTPUT. |
| | [15:01] | | SEQUENCEMODECAPABLE. |
| | [14:01] | | WRAPAROUNDCAPABLE. |
| 8to50 | | | Pseudostation title in display form. |

### Explanation

The SET PSEUDOSTATION ATTRIBUTES DCWRITE function can be used by an MCS to turn on certain attributes of a pseudostation.

This request is designed to be used by the allocating MCS before it transfers control of the pseudostation to another MCS. The new controlling MCS can determine the attributes of the station either by examining the TRANSFER STATION CONTROL (Class = 16) result message it receives or by issuing an INTERROGATE STATION ENVIRONMENT (DCWRITE Type = 4) request. That is, the values turned on by the SET PSEUDOSTATION ATTRIBUTES request are stored by the data comm subsystem and are made available to other MCSs.

This DCWRITE type is valid only for a pseudostation and can be issued only by the MCS that has allocated the station. This DCWRITE type cannot be issued on a station that has a fully participating MCS; that is, the allocating MCS must have exclusive control over the pseudostation. If any of these rules are violated, a DCWRITE error is returned.

### Example

```
ALLOCATE(MSG,8);
MSG[0] := LSN & 56 [47:8] & 1 [38:1];
MSG[6].[46:2] := 3;
RESULT := DCWRITE(MSG);
```

## STATION ASSIGNMENT TO FILE (DCWRITE Type = 64)

### Required Parameters/Fields

A message parameter and the following are required:

| Word | Field | Value | Description |
|---|---|---|---|
| 0 | [47:08] | 64 | Type. |
| | [39:16] | | Variant, as follows: |
| | [35:02] | | Tanking value, as follows: |
| | | 0 | Unspecified. |
| | | 1 | None. |
| | | 2 | Synchronous. |
| | | 3 | Asynchronous. |
| | [33:02] | 0 | MCS allows assignment to file. |
| | | 1 | The station is not available for assignment. |
| | | 2 | The station is in use (the MCS might or might not assign it later). |
| | | 3 | Invalid use of station (for example, open input on an output-only device). |
| | [29:06] | | As detailed in STATION ATTACH (DCWRITE Type = 1) DCWRITE. |
| | [23:24] | | FRSN (as received in the FILE OPEN (Class = 2) message for this station). |
| 2 | [39:16] | | Text size. If not zero, the text in MSG[6] is sent to the station if the assignment is allowed and the file MYUSE is OUT or IO. |
| 6 to end | | | See text for MSG[2].[39:16]. |

### Explanation

When an object job opens a remote file, the operating system procedure DCOPEN is called to create the station list of the remote file. DCOPEN generates a FILE OPEN (Class = 2) message for each station in the file and sends this message to the controlling MCS of the station. The “FILE OPEN (Class = 2)” message is described in the “MCS Result Message Formats” section of this manual. Receipt of the FILE OPEN message is interpreted by the MCS as a request by the DCOPEN procedure for the MCS to grant permission for the remote file to communicate with the station. On receipt of a FILE OPEN message for one of its stations, the controlling MCS replies with a STATION ASSIGNMENT TO FILE DCWRITE function. This DCWRITE type allows the MCS to determine the actual status of the assignment of the station in regard to the file.

With an appropriately assigned value in the variant field (MSG[0].[33:02]), the MCS can do one of the following:

- Allow assignment of the station to the file (MSG[0].[33.02] = 0).
- Unconditionally deny assignment of the station to the file (MSG[0].[33:02] = 1).
- Mark the station as being currently in use, with the possibility of assignment allowed at a later time (MSG[0].[33:02] = 2).
- Notify the operating system that the intended use of the station by the file is invalid (MSG[0].[33:02] = 3).

The system can override a request from the MCS to assign a station to the file if the intended use of the station is invalid. In this case, the DCWRITE function returns the value 101.

The object program in which the file is declared can interrogate the file DISPOSITION attribute to discover whether or not the station has been assigned.

When the MCS sends the STATION ASSIGNMENT TO FILE DCWRITE message with the variant equal to 0 (assignment allowed), the DCWRITE function checks to see if the requested use of the station is compatible with its description in the DATACOMINFO file. If, for example, the DATACOMINFO file defines the station to be an output-only device and the open request is for an input file, the DCWRITE function changes the DISPOSITION attribute of the station to illegal use (6), places an end of file message for the station in the input queue of the file, and returns a BAD DCWRITE RESULT value of 101.

If an object job opens a remote file (input only or I/O), the DCWRITE function verifies that the station whose assignment is requested is not already assigned to another input or I/O file unless the MCS is participating in I/O. An object job is not restricted in the number and types of remote files it can use, but a station can be assigned to only one input or I/O file at any given time without the cooperation of the MCS. After a station has been assigned to an input or I/O file, the DCWRITE function denies the request to assign the station to another one, and gives the same response as described above for illegal use, if the MCS is not participating. If the DCWRITE function does not find that the assignment allowed is an illegal use, the DCWRITE function does the following:

- Updates the DISPOSITION attribute of the station
- Requests the station to be enabled for input (if the station is not already enabled and the MCS is not participating in I/O)
- Raises the file POPULATION attribute count by one

If the MCS postpones assignment by sending the STATION ASSIGNMENT TO FILE DCWRITE message with the variant set to 2 (assignment postponed), the DCWRITE function updates the DISPOSITION attribute of the station.

If the MCS denies assignment (variant value of 1 or 3, which it can also do after allowing or postponing assignment), the DCWRITE function does the following:

- Updates the DISPOSITION attribute of the station
- Decreases the file POPULATION count (if assignment had previously been allowed)
- Places an end of file message in the input queue of the file

Subsequent attempts to communicate with this station cause an end of file action for the file.

The STATION ASSIGNMENT TO FILE DCWRITE function is intercepted when performed on a pseudostation that has a fully participating MCS. The DCWRITE function must be performed by the MCS that currently controls the station. The intercepted messages are placed in the primary queue of the fully participating MCS.

## Output Tanking for Remote Files

The data comm I/O subsystem supports disk tanking for remote output files. This feature is available only for nondirect files and for those stations for which the controlling MCS does not elect to participate in I/O.

The four states for tanking are UNSPECIFIED, NONE, SYNC, and ASYNC.

UNSPECIFIED is the default state. A file with an UNSPECIFIED tanking value is not tanked unless told to do so by an MCS at file assignment time.

NONE causes a file not to be tanked and prevents an MCS from designating tanking at file assignment time.

SYNC causes a file to be tanked. When the file is closed, the task does not resume execution until all tanked output has been completed. The controlling MCS for stations in a file for which the user designates a tanking value of SYNC cannot change this value at file assignment time.

ASYNC also causes a file to be tanked, but, unlike SYNC, the task continues execution after closing the file without waiting for all tanked output to be completed. This task can go to EOT while its output continues to be sent. The controlling MCS for stations in a file for which the user designates a tanking value of ASYNC cannot change this value at file assignment time.

If output is still queued when the file is closed with ASYNC tanking, the controlling MCS for each station in the file receives two file-close messages. The first is identical to the normal file-close message except that MSG[0].[24:01] equals 1. The MCS can then expect to receive a second file-close message indicating that all output to that station from that file has finished. The second file-close message does not contain such information as the INTNAME and TITLE of the remote file, the task name, or the mix number. Such information can be saved, if necessary, from the first file-close message. MSG[0].[25:01] equals 1 if it is the final, skeletal, file-close message.

Tanking can be invoked in three ways:

- By the task attribute TANKING
- By the file attribute TANKING
- By an MCS using a variant of the ASSIGNMENT TO FILE DCWRITE (DCWRITE Type = 64)

The TANKING task attribute is described in the *Task Attributes Programming Reference Manual.* The TANKING file attribute is described in the *I/O Subsystem Programming Reference Manual*. An MCS can also designate tanking at file assignment time. MSG[0].[25:02] of the file-open message contains the user-indicated tanking value (as determined by the TANKING task attribute or the TANKING file attribute). If the MCS does not wish to allow a file to be opened with the tanking value supplied to it, it can deny assignment. If MSG[0].[25:02] is 0 (UNSPECIFIED), the MCS can determine the tanking value for the file.

The format for the STATION ASSIGNMENT TO FILE DCWRITE message is as follows: MSG[0].[33:02] contains the file disposition, and MSG[0].[35:02] contains the tanking value that the MCS wishes for the file. If the value that the MCS designates is not compatible with the value that the user designates or with the value it or another MCS has designated for another station in the file, it produces a DCWRITE error 144, INVALID OUTPUT TANKING SPECIFICATION.

If the user causes a break-on-output action for a remote file that has been tanked, all tanked output from that file to that station is discarded until the break is handled by the user task. If the task has already closed the file, all tanked output to that station is discarded. The output to other stations in the file continues. This condition could result in a great deal of discarded output but does allow unwanted output for a task that can no longer be running to be stopped. Tanked output is not returned to an MCS if it performs a RECALL MESSAGES (DCWRITE Type = 41) DCWRITE function.

Tanked output is separate from output that is directly queued for a station. Queued output for a station can contain output from several files and can contain messages from an MCS. As the amount of queued output from the particular tanked file is completed, more of the tanked output is detanked and queued. Therefore, output from different remote files from the same task can be received by the station in a different order than that in which it was sent. All output from the same file is received in the order it was sent.

If a task is run in a swapspace, it is not swapped out for remote output if its remote files are tanked. If a task is not run in swapspace, it is not suspended remote output if its remote files are tanked. This situation means that some tasks that do a large amount of remote output (or do bursts of remote output) remain on the processor longer and have their performance improved.

## MCS Participation in I/O

The MCS can also designate those options under which the station message traffic is to be handled by turning on the desired option bits in MSG[0].[29:06]. (Refer to the "STATION ATTACH (DCWRITE Type = 1)" DCWRITE function in this section for the option bit interpretations.) If MSG[0].[29:01] = 1, the MCS indicates its desire to participate in the I/O functions for the station and the file. Participation in I/O means that the MCS acts as arbiter for messages to and from the file. Specifically, the MCS receives all input messages directly from the station and decides whether the input is to be acted on by itself or forwarded to the object program file. Any input to the object job file is forwarded through the use of the WRITE TO OBJECT JOB (DCWRITE Type = 65) DCWRITE function. Output to the station from the object job file (writes executed by the object job) is sent directly to the MCS in the current queue of the station in the form of an OBJECT JOB OUTPUT  (Class= 3) message. Output to the station is forwarded with the customary WRITE (DCWRITE Type = 33) DCWRITE function.

The ability of the MCS to interpose between the object job file and the station allows it to perform pre-editing of output records to a station from an object job, message switching functions (such as direct input from a station to any one of several files), or other extended functions are required. The MCS allows the station to be assigned to one or more files at any given time in situations in which the MCS is involved in the I/O functions.

If MSG[0].[29:01] = 0, the MCS relinquishes all responsibility for control of functions related to the handling of I/O. In this situation, all GOOD INPUT RECEIVED (Class = 0) messages from the station are forwarded to the object program file, and all OBJECT JOB OUTPUT (Class = 3) messages from the object program are forwarded directly to the station.

The MCS can allow a station to be assigned to one or more files, but a station can be assigned to only one input file at a time without the participation of the MCS in I/O for the station. Therefore, a station can be opened for output using one file and opened for input using another (for example, in the same program) so that the object job appears to read one file and write to another, although the physical terminal is the same in both cases.

When a station is assigned to an input file and the MCS has elected not to participate in I/O, the DCWRITE function automatically enables the station for input if it is not already enabled. However, if the MCS participates in I/O, it is responsible for enabling or disabling the station.

Whether or not the MCS elects to participate in I/O, it receives all ERROR RESULT (Class = 99) messages, STATION EVENT (Class = 1) messages, and FILE CLOSE (Class = 4) messages of the station.

Also, the MCS could choose to participate in I/O functions for some of the stations in the file and not to participate for other stations, because stations are assigned to files on a station by station basis.

The STATION ASSIGNMENT TO FILE DCWRITE function requires the use of an FRSN rather than the customary LSN. The MCS must retain the FRSNs given to it by the FILE OPEN message and maintain the correspondence between LSN and FRSN. This maintenance of correspondence is particularly significant if the MCS participates in I/O functions where a station is a member of (assigned to) more than one file and the MCS is to perform functions such as message switching. When a station is a member of more than one file at a time, an LSN can have several related FRSNs. If MSG[2].[39:16] is not zero, the MCS allows the file assignment; if the MYUSE of the file is OUT or IO, the DCWRITE function generates an output to the station of the text starting in MSG[6].

## Stations without Line Assignments

Stations declared in the DATACOMINFO file without line assignments can be included as FAMILY members in the FILE section of the DATACOMINFO file. If an object program opens a remote file containing one of these stations, the controlling MCS of the station is restricted in its response to the FILE OPEN message unless it is participating in I/O for the station. If the MCS is not participating in I/O for the station and attempts to allow the STATION ASSIGNMENT TO FILE, the DCWRITE function returns the result value of 88 (the station has no line assignment), and the disposition of the station in the file remains unknown (0). If the program tries to write to the station (or does a broadcast write), the program waits in the write statement until the disposition of the station becomes known (assignment was allowed, denied, or postponed, or an invalid use was attempted). Therefore, the MCS should either deny or postpone the assignment of the station to the file.

If the MCS does participate in I/O for the station without line assignment, it must perform message switching. Because the MCS is participating in I/O, every output from the program to the station without a line assignment is placed in the current queue of the station. The MCS must change the class field of the message from OBJECT JOB OUTPUT (Class = 3) to WRITE (DCWRITE Type = 33) and the LSN/FRSN field to the LSN of the station actually communicating with the program. The MCS must then pass the message on using the DCWRITE function.

In the case of input when the MCS is participating, the MCS must do the following:

- Change GOOD INPUT RECEIVED (Class = 0) messages from the station that is actually communicating with the program to WRITE TO OBJECT JOB (DCWRITE Type = 65) messages
- Change the LSN/FRSN field to the FRSN of the station without line assignment
- Pass the message to the object job by using the function DCWRITE

The MCS must be participating in I/O for both the real and the dummy station to capture and switch input messages.

### Example

```
ALLOCATE(MSG,6);
MSG[0] := 0 & 64 [47:8] & 0 [39:10] & 0 [29:6]
           & FRSN [23:24];
RESULT := DCWRITE(MSG);
```

# WRITE TO OBJECT JOB (DCWRITE Type = 65)

### Required Parameters/Fields

A message parameter and the following are required:

| Word | Field | Value | Description |
|---|---|---|---|
| 0 | [47:08] | 65 | Type. |
| | [39:16] | | Variant, not used. |
| | [23:24] | | FRSN. |
| 2 | [39:16] | | Number of bytes of meaningful text supplied, starting at MSG[6]. |
| 6 to end | | | The number of words of text area, consistent with the number of bytes specified in MSG [2].[39:16]. (For example, n must satisfy n > (MSG[2].[39:16] + 5) DIV 6.) |

### Explanation

The WRITE TO OBJECT JOB DCWRITE function allows an MCS to forward input records to an object job file. If the MCS elects to participate in I/O handling for a station, all inputs from the station are sent directly to the MCS; the MCS is then responsible for transferring the input to the file. By allowing the MCS to participate in I/O, the MCS can perform preediting of input or output data records, message switching functions, and so forth.

The MCS must maintain the correlation between the LSN of a station and any FRSNs so that this DCWRITE type can be invoked with the desired results.

This DCWRITE can also be used after the STATION ASSIGNMENT TO FILE (DCWRITE Type = 64) DCWRITE function, whether or not the MCS has elected to participate in the I/O handling.

### Example

```
ALLOCATE(MSG,16);
MSG[0] := FRSN & 65 [47:8];
MSG[2].[39:16] := 57;
REPLACE POINTER(MSG[6],8) BY
   "THE OBJECT JOB WILL RECEIVE THIS TEXT AS AN INPUT",
   "RECORD.";
RESULT := DCWRITE(MSG);
```

# STATION BREAK (DCWRITE Type = 66)

### Required Parameters/Fields

A message parameter and the following are required:

| Word | Field | Value | Description |
|---|---|---|---|
| 0 | [47:08] | 66 | Type. |
| | [39:16] | | Variant field, not used. |
| | [23:24] | | FRSN. |

### Explanation

The STATION BREAK DCWRITE function allows the MCS to inform an object job that a break-on-output occurred at a station. Any subsequent attempt at output to the station from that file results in break error action.

In addition, if output is tanked for the file and all output-capable stations in the file with line assignments have break conditions, all tanked output for the file is discarded.

### Example

```
ALLOCATE(MSG,6);
MSG[0] := FRSN & 66 [47:8];
RESULT := DCWRITE(MSG);
```

# ADD STATION TO FILE (DCWRITE Type = 67)

### Required Parameters/Fields

A message parameter and the following are required:

| Word | Field | Value | Description |
|---|---|---|---|
| 0 | [47:08] | 67 | Type. |
| | [39:16] | | As detailed in the STATION ASSIGNMENT TO FILE (DCWRITE Type = 64) DCWRITE. |
| | [23:10] | | File number. |
| 1 | [23:24] | | LSN of the station to be added. |
| 2 | [39:16] | | Text size of the output text. |
| 6 to end | | | Output text. |

### Explanation

The ADD STATION TO FILE DCWRITE function allows an MCS to perform the equivalent of the following ALGOL construct:

```
REPLACE FILEID.FAMILY BY * + "STATIONNAME"
```

This DCWRITE type permits an MCS to spontaneously add a new station to an open remote file currently under control of the MCS and to simultaneously perform the necessary assignment of the file as would otherwise be required by the STATION ASSIGNMENT TO FILE (DCWRITE Type = 64) DCWRITE function. Although similar in function to the Type = 64 DCWRITE function, this DCWRITE type cannot be used in response to a FILE OPEN (Class = 2) result message.

The station to be added is designated by an LSN in MSG[1].[23:24] and must be currently attached to the MCS. The file to which the station is to be added is designated in MSG[0].[23:10]. Refer to the description of the “FILE OPEN (Class = 2)” result message in the “MCS Result Message Formats” section for the meaning and origin of the file number.) If the specified station is already a member of the file, error 82 (STATION ALREADY IN FILE) is returned. MSG[0].[33:02] indicates the disposition to be assigned to the station. (Refer to the semantics of the STATION ASSIGNMENT TO FILE (DCWRITE Type = 64) DCWRITE function for the meaning of this field.)

If no errors are detected by the DCWRITE function, the content of MSG[0].[13:14] is the RSN that has been assigned to the new station within the file.

Notification of the action performed by this DCWRITE type is not given to the object program owning the file. In particular, the LASTSTATION file attribute is not altered. The MCS can elect to inform the object program by issuing a WRITE TO OBJECT JOB (DCWRITE Type = 65) DCWRITE function with the RSN value returned. If MSG[2].[39:16] is not zero, the station is successfully added to the file. If the file MYUSE is OUT or IO, this DCWRITE type sends the text starting in MSG[6] to the station.

The ADD STATION TO FILE DCWRITE function is intercepted only if it contains text and is performed on a pseudostation that has a fully participating MCS. The DCWRITE function must be performed by the MCS that currently controls the station. The intercepted messages are placed in the primary queue of the fully participating MCS.

### Example

```
ALLOCATE(MSG,8);
MSG[0] := 0 & 67 [47:8] & FILENR [23:10];
MSG[1] := LSN;
RESULT := DCWRITE(MSG);
CHANGE TERMINAL ATTRIBUTES (DCWRITE Type = 68)
```

### Required Parameters/Fields

The minimum message size for this DCWRITE type is seven words. A message parameter and the following are required:

| Word | Field | Value | Description |
|---|---|---|---|
| 0 | [47:08] | 68 | Type. |
| | [39:16] | | Variant, as follows: |
| | [26:01] | 1 | Screen to be updated. |
| | [25:01] | 1 | Width to be updated. |
| | [24:01] | 1 | Page size to be updated. |
| | [23:24] | | FRSN. |
| 6 | | | As follows: |
| | [24:01] | | New screen value. |
| | [23:12] | | New width value. |
| | [11:12] | | New page size value. |

### Explanation

This FRSN-oriented DCWRITE function is used to change the PAGESIZE, WIDTH, or SCREEN file attributes of a remote file. These three attributes are initially equal to values given from the corresponding station attributes in the DATACOMINFO file. On receipt of a FILE OPEN message, the MCS can choose to override the declared values by using the CHANGE TERMINAL ATTRIBUTES DCWRITE function. These three attributes are accessible to the object program when the remote file is opened. Because the object program and the MCS operate asynchronously, a small period can exist between the time the file is marked open and the time the MCS performs the above DCWRITE function; during this time, the object program can interrogate these attributes and obtain the default values from the DATACOMINFO file. Thus, the MCS should perform this DCWRITE type before it performs the STATION ASSIGNMENT TO FILE (DCWRITE Type = 64) DCWRITE function; the latter DCWRITE type allocates the DISPOSITION attribute, which can be used by the object program to determine the validity of other remote attributes.

The CHANGE TERMINAL ATTRIBUTES DCWRITE function can be used any time the remote file is open. The changes effected by this DCWRITE function are not permanent; the update terminal attribute values are discarded by the system when the file is closed.

The CHANGE TERMINAL ATTRIBUTES DCWRITE function is intercepted when performed on a pseudostation that has a fully participating MCS. The DCWRITE function must be performed by the MCS that currently controls the station. The intercepted messages are placed in the primary queue of the fully participating MCS.

### Example

```
ALLOCATE(MSG,7);
MSG[0] := FRSN & 68 [47:8] & 1 [25:1];
MSG[6] := 0 & 72 [23:12];
RESULT := DCWRITE(MSG);
```

# SUBTRACT STATION FROM FILE (DCWRITE Type = 69)

### Required Parameters/Fields

A message parameter and the following are required:

| Word | Field | Value | Description |
|---|---|---|---|
| 0 | [47:08] | 69 | Type. |
| | [23:10] | | File number. |
| 1 | [23:24] | | LSN of station to be subtracted. |

### Explanation

The SUBTRACT STATION FROM FILE DCWRITE function allows an MCS to perform the equivalent of the following ALGOL construct:

```
REPLACE FILEID.FAMILY BY * - "STATIONNAME.";
```

This DCWRITE type permits an MCS to spontaneously subtract a station from an open remote file currently under the control of the MCS.

# Line-Oriented Requests

The following DCWRITE types are line-oriented requests. The additional requirements that apply to all line-oriented DCWRITE types are as follows:

- The minimum acceptable message size is 8 words.
- MSG[0].[23:24] can contain one of the following:
  - An LSN for any station on the desired line, provided that the requesting MCS controls that station
  - A relative NSP number and line number in the following form and hereafter referred to as DL:

```
MSG[0].[23:01] = 1, not an LSN
MSG[0].[22:07] = Relative NSP number
MSG[0].[15:08] = Line number
```

The MCS must control at least one station on the designated line when using the DL number (except for the INTERROGATE LINE DCWRITE function).

## MAKE LINE READY (DCWRITE Type = 96)

### Required Parameters/Fields

A message parameter and the following are required:

| Word | Field | Value | Description |
|---|---|---|---|
| 0 | [47:08] | 96 | Type. |
| | [39:16] | | Variant, not used. |
| | [23:24] | | LSN or DL number. |

### Explanation

The MAKE LINE READY DCWRITE function allows an MCS to resume execution of the NDLII algorithm that had been suspended because of a MAKE LINE NOT READY DCWRITE function.

A LINE STATUS CHANGE RESULT (Class = 9) message is always generated by the NSP as a part of its response to the MAKE LINE READY DCWRITE function (even if the line is already READY). The Class = 9 message is generated only for the lowest-numbered valid station on the line and is routed to the current queue of that station if the controlling MCS indicates that it is to receive Class = 9 messages. Otherwise, the message is discarded by the operating system.

The MAKE LINE READY DCWRITE function is intercepted when performed on a pseudostation that has a fully participating MCS. The DCWRITE must be performed by the MCS that currently controls the station. The intercepted messages are placed in the primary queue of the fully participating MCS.

### Example

```
ALLOCATE(MSG,8);
MSG[0] := LSN & 96 [47:8];
RESULT := DCWRITE(MSG);
```

# MAKE LINE NOT READY (DCWRITE Type = 97)

### Required Parameters/Fields

A message parameter and the following are required:

| Word | Field | Value | Description |
|---|---|---|---|
| 0 | [47:08] | 97 | Type. |
|  | [39:16] |  | Variant field, not used. |
|  | [23:24] |  | LSN or DL number. |

### Explanation

The MAKE LINE NOT READY DCWRITE function allows an MCS to make a line NOT READY. If the LINE.BUSY variable is TRUE when this request is presented to the NSP, the LINE STATUS CHANGE RESULT (Class = 9) message is not generated until the line has actually become NOT READY.

A LINE STATUS CHANGE RESULT (Class = 9) message is generated by the NSP as a part of its response to the MAKE LINE NOT READY DCWRITE function if, and only if, the line was READY prior to the MAKE LINE NOT READY DCWRITE function. The Class = 9 message is generated only for the lowest-numbered valid station on the line and is routed to the current queue of that station if the controlling MCS indicates that it is to receive Class = 9 messages. Otherwise, the message is discarded by the operating system.

The MAKE LINE NOT READY DCWRITE function is intercepted when performed on a pseudostation that has a fully participating MCS. The DCWRITE function must be performed by the MCS that currently controls the station. The intercepted messages are placed in the primary queue of the fully participating MCS.

### Example

```
ALLOCATE(MSG,8);
MSG[0] := 0 & 97 [47:8] & 1 [23:1] & NSPNR [22:7]
            & LINENR [15:8] & STANR [7:8];
RESULT := DCWRITE(MSG);
```

# DIALOUT (DCWRITE Type = 98)

### Required Parameters/Fields

The minimum size message size for the DIALOUT DCWRITE function is eight words. A message parameter and the following are required:

| Word | Field | Value | Description |
|---|---|---|---|
| 0 | [47:08] | 98 | Type. |
| | [39:16] | | Variant field, not used. |
| | [23:24] | | LSN or DL. |
| 2 | [39:16] | | Number of digits of the telephone number (in bytes). |
| 6 to end | | | The telephone number to be dialed, represented as a dialing sequence, which is a left-justified EBCDIC numeric string. The dialing sequence is a string of operators, each one byte long. The operators and their corresponding EBCDIC codes are shown in Table 5–5. |

Table 5–5 presents the values that are used to control telephone dialing. These values must be used in the telephone number field, which begins in word 6 of the DIALOUT DCWRITE message.

### Table 5–5. Dialing Sequence Operators

| Operator | Code | Operator | Code |
|---|---|---|---|
| Dial 0 | 1111 0000 | EON | 1111 1100 |
| Dial 1 | 1111 0001 | WFSDT | 1111 1101 |
| Dial 2 | 1111 0010 | Delay 1 sec. | 1100 0001 |
| Dial 3 | 1111 0011 | Delay 2 sec. | 1100 0010 |
| Dial 4 | 1111 0100 | Delay 3 sec. | 1100 0011 |
| Dial 5 | 1111 0101 | Delay 4 sec. | 1100 0100 |
| Dial 6 | 1111 0110 | Delay 5 sec. | 1100 0101 |
| Dial 7 | 1111 0111 | Delay 6 sec. | 1100 0110 |
| Dial 8 | 1111 1000 | Delay 7 sec. | 1100 0111 |
| Dial 9 | 1111 1001 | Delay 8 sec. | 1100 1000 |
| Dial * | 1111 1010 | Delay 9 sec. | 1100 1001 |
| Dial # | 1111 1011 | Delay 10 sec. | 1100 1010 |

## Explanation

The DIALOUT DCWRITE function allows a remote station to be called through the telephone switching network. The line involved must have been declared in the DATACOMINFO file to have dialout capability and must be READY, disconnected, and neither ringing nor SWITCHEDBUSY.

The NSP always retries a dialout operation five times. If the dialout operation fails with an ABANDON-CALL-RETRY error, no further retries are attempted.

Telephone numbers of any reasonable length can be used to call an in-house station with an extension number or an outside line with or without an area code. Provided that the number can be reached properly, the DCC releases the dialout logic only after a proper “handshake” between the data sets involved has been made and the system data set is in data mode. Normal operation can then proceed.

Some automatic calling units (ACUs) are equipped with an optional feature that requires the phone number supplied to be terminated by an end of number (EON) character. The DATACOMINFO file designates which ACUs have this requirement by use of the ENDOFNUMBER attribute. The DCALGOL programmer is not required to terminate phone numbers for such ACUs with the EON character. Instead, the DCC supplies this character when it is required.

The Wait For Supplementary Dial Tone (WFSDT) operator causes the ACU to wait until it detects a supplementary dial tone before signaling the NSP to proceed. Not all ACUs support this feature.

The delay operators cause the NSP to wait the specified number of seconds before it continues the dialout operation.

### Example

```
ALLOCATE(MSG,8);
MSG[0] := LSN & 98 [47:8];
MSG[2].[39:16] := 10;
REPLACE POINTER(MSG[6],8) BY "2135556521";
RESULT := DCWRITE(MSG);
```

# DISCONNECT (DCWRITE Type = 99)

### Required Parameters/Fields

The minimum message size required for the DISCONNECT DCWRITE function is eight words. A message parameter and the following are required:

| Word | Field | Value | Description |
|---|---|---|---|
| 0 | [47:08] | 99 | Type. |
| | [39:16] | | Variant, not used. |
| | [23:24] | | LSN or DL number. |

### Explanation

The DISCONNECT DCWRITE function causes the system to “hang up” on a line connected through a telephone switchboard or switching network. The line must have been declared in the DATACOMINFO file with the SWITCHED line attribute equal to TRUE, and the line must be READY, CONNECTED, and not BUSY.

### Example

```
ALLOCATE(MSG,8);
MSG[0] := 0 & 99 [47:8] & 1 [23:1] & NSPNR [22:7]
           & LINENR [15:8] & STANR [7:8];
RESULT := DCWRITE(MSG);
```

# INTERROGATE SWITCHED STATUS (DCWRITE Type = 101)

## Required Parameters/Fields

The minimum message size required for the INTERROGATE SWITCHED STATUS DCWRITE function is eight words. A message parameter and the following are required:

| Word | Field | Value | Description |
|---|---|---|---|
| 0 | [47:08] | 101 | Type. |
| | [39:16] | | Variant, not used. |
| | [23:24] | | LSN or DL. |

## Explanation

The INTERROGATE SWITCHED STATUS DCWRITE function is allowed only for a station on a line declared in the DATACOMINFO file with the SWITCHED line attribute equal to TRUE. In response to the DCWRITE function, the DCC generates a SWITCHED STATUS RESULT (Class = 7) message which, if the DL number is used, it places in the current queue of the lowest-numbered valid station on the line. If LSN is used, it places the message in the current queue of the indicated station.

## Example

```
ALLOCATE(MSG,8);
MSG[0] := LSN & 101 [47:8];
RESULT := DCWRITE(MSG);
```

## SET/RESET AUTOANSWER (DCWRITE Type = 102)

### Required Parameters/Fields

The minimum message size required for the SET/RESET AUTOANSWER DCWRITE function is eight words. A message parameter and the following are required:

| Word | Field | Value | Description |
|---|---|---|---|
| 0 | [47:08] | 102 | Type. |
|  | [39:16] | 0 | AUTOANSWER has the value FALSE. |
|  |  | 1 | AUTOANSWER has the value TRUE. |
|  | [23:24] |  | LSN or DL. |

### Explanation

The SET/RESET AUTOANSWER DCWRITE function is allowed for stations on lines declared in the DATACOMINFO file with the SWITCHED line attribute equal to TRUE, regardless of the status of the line. For all lines so declared that are in a normal disconnected state, the NSP monitors the data set lead ring indicator (RI) for incoming calls. When AUTOANSWER is TRUE and ringing is detected, the NSP answers the phone automatically. If AUTOANSWER is FALSE, the phone is not answered.

The SET AUTOANSWER DCWRITE function causes the NDLII variable LINE.DISCONNECTACTION have the value AUTOANSWER. The RESET AUTOANSWER DCWRITE function causes LINE.DISCONNECTACTION to have the value NONE and the ring indication to be ignored.

### Example

```
ALLOCATE(MSG,8);
MSG[0] := LSN & 102 [47:8] & 1 [39:16];
RESULT := DCWRITE(MSG);
```

# SET/RESET LINE TOGS-TALLYS (DCWRITE Type = 103)

### Required Parameters/Fields

The minimum message for this DCWRITE type is eight words. A message parameter and the following are required:

| Word | Field | Value | Description |
|---|---|---|---|
| 0 | [47:08] | 103 | Type. |
| | [39:16] | | Variant, not used. |
| | [23:01] | 1 | MSG[0].[23:24] contains DL. |
| | [23:24] | | DL or LSN. |
| 6 | [23:01] | 1 | Enable setting of LINE(TOG[1]). |
| | [22:01] | 1 | Enable setting of LINE(TOG[0]). |
| | [21:01] | 1 | Enable setting of LINE(TALLY[1]). |
| | [20:01] | 1 | Enable setting of LINE(TALLY[0]). |
| | [17:01] | | Setting of LINE(TOG[1]). |
| | [16:01] | | Setting of LINE(TOG[0]). |
| | [15:08] | | Setting of LINE(TALLY[1]). |
| | [07:08] | | Setting of LINE(TALLY[0]). |

### Explanation

The SET/RESET LINE TOGS-TALLYS DCWRITE function allows an MCS to dynamically turn on or off any or all line TOGs or line TALLYs for a given line.

The SET/RESET LINE TOGS-TALLYS DCWRITE function is intercepted when performed on a pseudostation that has a fully participating MCS. The DCWRITE function must be performed by the MCS that currently controls the station. The intercepted messages are placed in the primary queue of the fully participating MCS.

### Example

```
ALLOCATE(MSG,8);
MSG[0] := DL & 1 [23:1] & 103 [47:8];
MSG[6] := 0 & 1 [22:1] & 1 [21:1] & 0 [16:1] & 100 [15:8];
RESULT := DCWRITE(MSG);
```

# LINE INTERROGATE (DCWRITE Type = 104)

### Required Parameters/Fields

The minimum message size is 16 words. A message parameter and the following are required:

| Word | Field | Value | Description |
|---|---|---|---|
| 0 | [47:08] | 104 | Type. |
| | [39:16] | | Variant, not used. |
| | [23:24] | | LSN or DLS number. |

### Explanation

The LINE INTERROGATE DCWRITE function requests information about current line status. Refer to the “LINE INTERROGATE RESULT (Class = 24)” message in the “MCS Result Message Formats” section for a description of the result indication of the contents.

An MCS can use this DCWRITE function to determine the status of any line in the data comm network, whether or not it controls any stations on the line.

The LINE INTERROGATE DCWRITE function is intercepted when performed on a pseudostation that has a fully participating MCS. The DCWRITE function must be performed by the MCS that currently controls the station. The intercepted messages are placed in the primary queue of the fully participating MCS.

### Example

```
ALLOCATE(MSG,16);
MSG[0] := LSN & 104 [47:8];
RESULT := DCWRITE(MSG);
```

# FORCE LINE NOT READY (DCWRITE Type = 105)

### Required Parameters/Fields

The minimum message size for this DCWRITE function is eight words. A message parameter and the following are required:

| Word | Field | Value | Description |
|---|---|---|---|
| 0 | [47:08] | 105 | Type. |
| | [23:24] | | LSN or DLS number. |

### Explanation

The FORCE LINE NOT READY DCWRITE function is identical to that of the MAKE LINE NOT READY (DCWRITE Type = 97) DCWRITE function.

### Example

```
ALLOCATE(MSG,8);
MSG[0] := LSN & 105 [47:8];
RESULT := DCWRITE(MSG);
```

# Reconfiguration Request DCWRITE Types

Reconfiguration requests allow the data comm environment to be dynamically modified. Reconfiguration has been implemented in such a fashion as to allow all data comm functions not directly affected by any reconfiguration request to continue in parallel with that request and to allow resumption of activities for LSPs, lines, and stations that were directly affected after completion of the request. Reconfiguration requests cause the NSP and operating system tables to be updated in memory and the same changes to be reflected in the DATACOMINFO file on disk, so that subsequent reinitialization of data comm accurately reflects the new configuration. A request for reconfiguration invokes DCRECON as an independent runner (if it is not already running) and enqueues the request for the reconfiguration routine. Multiple requests for reconfiguration are processed serially, although in parallel with other data comm activity wherever possible. No initialization of NSPs occurs during the processing of a given request; the initialization remains pending until the completion of the request in progress.

All message control systems affected by any reconfiguration request (whether the request for reconfiguration was issued by that MCS or not) are notified of any successfully completed reconfiguration. An affected MCS is determined to be any MCS currently attached to one or more stations whose relative NSP number, line number, or station number was changed as a result of a reconfiguration request. Affected message control systems receive an exact copy of the LSP EXCHANGE RESULT (Class = 8) message, the SWAP LINE RESULT (Class = 10) message, or the MOVE/ADD/SUBTRACT STATION (Class = 11) message sent to the original requesting MCS, as well as one or more DLS UPDATE RESULT (Class = 12) messages pertinent to the stations known to each MCS.

## SWAP LINES (DCWRITE Type = 128)

### Required Parameters/Fields

The minimum message size required for the SWAP LINES DCWRITE function is eight words. A message parameter and the following are required:

| Word | Field | Value | Description |
|---|---|---|---|
| 0 | [47:08] | 128 | Type. |
| | [39:16] | 0 | Lines are left NOT READY after reconfiguration is completed. |
| | | 1 | Lines are left READY after reconfiguration is completed. |
| | [23:24] | | LSN or DL number of the first line. |
| 7 | [23:24] | | DL number of the second line. |

### Explanation

The SWAP LINES DCWRITE function allows an MCS to logically swap two lines and includes any and all stations contained on each of the lines. The requesting MCS receives a SWAP LINES RESULT (Class = 10) message in its primary queue, which signifies completion of processing of the request. If no errors occurred, one or more DLS UPDATE RESULT (Class = 12) messages can have been issued to the primary queue of the MCSs owning stations on the lines.

If the first line has stations on it, the MCS must control at least one of them. If either line is DIALOUT, both must be DIALOUT.

### Example

```
ALLOCATE(MSG,8);
MSG[0] := LSN & 128 [47:8] & 1 [39:16];
MSG[7] := 0 & 1 [23:1] & DL [22:15];
RESULT := DCWRITE(MSG);
```

# EXCHANGE LSPS (DCWRITE Type = 129)

### Required Parameters/Fields

A message parameter and the following are required:

| Word | Field | Value | Description |
|---|---|---|---|
| 0 | [47:08] | 129 | Type. |
| | [39:16] | | Variant field, as follows: |
| | [26:01] | 0 | The target NSP can be uninitialized, but paths to LSPs being exchanged onto that NSP are tested for availability. |
| | | 1 | If the target NSP is uninitialized, paths to LSPs being exchanged onto it are to be tested for availability. The value of this field is ignored if MSG[0].[25:01] = 1. |
| | [25:01] | 0 | The target NSP can be uninitialized. |
| | | 1 | The target NSP is initialized if required. |
| | [24:01] | 0 | Lines on LSPs are left NOT READY after reconfiguration. |
| | | 1 | Lines on LSPs are left READY after reconfiguration. |
| | [22:07] | | Relative NSP number. |
| | [15:16] | | LSP mask. |

### Explanation

The EXCHANGE LSPS DCWRITE function allows an MCS to transfer control of any or all LSPs specified in the LSP alternates declaration to another NSP that has an I/O path to the LSPs being exchanged.

Receipt of an LSP EXCHANGE RESULT (Class = 8) message signifies that processing of the request has completed. The LSP mask in MSG[0].[15:16] is updated in the LSP EXCHANGE RESULT (Class = 8) message to reflect any LSPs that failed to initialize successfully after the transfer to the recipient NSP. One or more DLS UPDATE RESULT (Class = 12) messages could be issued to the primary queue of the MCS as a consequence of this DCWRITE type.

If [0].[25:01] = 0, the LSP exchange occurs regardless of the state of the target NSP.

If [0].[25:01] = 1, the target NSP is initialized if it is not running. If initialization is required and fails because of a hardware condition, error 87 (uninitialized NSP) is returned.

The value in the field [0].[26:01] is tested. If [0].[25:01] = 1, the value in [0].[26:01] is ignored.

If [0].[26:01] = 0 and the target NSP is uninitialized, the availability is tested of a path to each LSP that is to be exchanged onto that NSP.

If [0].[26:01] = 1 and the target NSP is uninitialized, the path tests are overridden, and the subject NSP does not have I/O requests issued to it.

### Example

```
ALLOCATE(MSG,8);
MSG[0] := LSPMASK & 129 [47:8] & 1 [39:16] & NSPNR [22:7];
RESULT := DCWRITE(MSG);
```

## MOVE/ADD/SUBTRACT STATION (DCWRITE Type = 130)

### Required Parameters/Fields

The minimum message size for the MOVE/ADD/SUBTRACT STATION DCWRITE function is eight words. A message parameter and the following are required:

| Word | Field | Value | Description |
|---|---|---|---|
| 0 | [47:08] | 130 | Type. |
| | [39:16] | | Variant field, as follows: |
| | [25:01] | 0 | Station attributes are not updated. |
| | | 1 | Station attributes are updated as specified. |
| | [24:01] | 0 | The station is left NOT READY after reconfiguration is completed. |
| | | 1 | The station is left READY after reconfiguration is completed. |
| | [23:24] | | The current LSN or DLS number. |
| 7 | [23:01] | 0 | The station is to be subtracted. MSG[7].[22:15] is irrelevant. |
| | | 1 | The station is to be added or moved. MSG[7].[22:15] is the receiving DL number. |

## MSG[0].[25:01] = 1

The following optional information is required if MSG[0].[25:01] = 1:

| Word | Field | Description |
|---|---|---|
| 8 | | New line attributes. |
| | [39:08] | NDLIITERMINX is the index within MSG of the name of the terminal with the required logical station attributes. |
| | [31:08] | NDLIILINEINX is the index within MSG of the name of the line with the required physical line attributes. |
| MSG[NDLIITERMINX] | | The name of the terminal with the required logical station attributes for the station that is being moved or added. |
| MSG[NDLIILINEINX] | | The name of the line that has declared physical attributes for the empty line to which the station is being moved or added. |

### Explanation

The MOVE/ADD/SUBTRACT STATION DCWRITE function allows an MCS to do one of the following:

- Move stations from one line to another
- Assign a station to a line where it had no prior line assignment
- Remove a station from a line (thus, leaving it with no line assignment)

If MSG[7].[23:01] is equal to 0, MSG[7].[22:23] does not contain a DL number and a subtraction of the station from a line is indicated, which leaves the station with no line assignment. In the event of such a pure subtraction, if any requests are queued for the station, they are returned in the current message queue of the station as RECALL MESSAGE (Class = 6) messages, as though a RECALL MESSAGE (DCWRITE Type = 41) DCWRITE function had been performed by the requesting MCS. Receipt of a MOVE/ADD/SUBTRACT STATION (Class = 11) message in the primary message queue of the requesting MCS signifies completion of the request. A DLS UPDATE RESULT (Class = 12) message could also be placed in the primary message queue of the MCS as a consequence of the MOVE/ADD/SUBTRACT STATION DCWRITE function.

If MSG[0].[25:01] is equal to 1 and MSG[7].[23:01] is not equal to 0, the line attributes designated in MSG[8] are applied to the new line. In a particular attribute field in MSG[8] is 0, the current line attribute is not modified. Name attributes must be designated in EBCDIC and must terminate with a period. These name attributes must correspond to valid nondefault names in the DATACOMINFO file.

The maximum number of stations per line is 255.

If MSG[0].[25:01] = 1, a line cannot be named (that is, the index NDLIILINEINX must be 0) if the target line has stations on it. A line can be named only if the target line is empty.

## Physical Attributes

The physical attributes of the target line can be mapped from those of the source line from which a station is being moved if the following requirements are met:

- MSG[0].[25:01] = 1.
- There are no stations on the target line.
- The target line is different from the source line.
- No line is named in the request (the index NDLIILINEINX is 0).

The changes made to the physical connection characteristics of the target line must be within the scope of the hardware settings of the line adapter and electrical interface (EI). Thus, a character-oriented line cannot be changed to a bit-oriented line and vice versa, a private line cannot be changed to a switched line and vice versa, and an autodial line cannot be changed to an autoanswer line and vice versa. However, a single line adapter can be used alternately with asynchronous and synchronous modems if they are character-oriented and of similar switching characteristics.

The following is a list of physical attributes copied from the named line (or source line) to the target line:

- CRC polynomial
- CRC polynomial initial setting
- CRC polynomial final setting
- Mode (async, sync, or bit-sync)
- Vertical parity (mode = async or sync)
- Horizontal parity (mode = async or sync)
- BCS type (mode = async or sync)
- Stop bits (mode = async)
- Bit rate (that is, line speed)
- Sync character (mode = sync)
- DLE character (mode = sync)
- Address mode (mode = bit-sync)
- Control mode (mode = bit-sync)
- CLASS number
- Translate table
- Receive delay

- Transmit delay
- Receive timeout
- DPR delay for ACU
- Set digit delay for ACU
- Secondary address (mode = bit-sync)
- Function (mode = bit-sync)
- Disconnect action on loss of DSR
- Receive ready
- End of number for ACU

Whether MSG[0].[25:01] = 1 or 0, the transmit delay associated with the station being added or moved is imposed on the target line if that delay is longer than that already in use on the target line.

## Logical Attributes

If MSG[0].[25:01] = 1, the station is being moved or added, and there is a terminal named in the request (NDLIITERMINX is not zero), then logical characteristics of the station are altered to match those associated with the named terminal.

The following is a list of logical attributes that are copied from the named terminal to the station being moved or added:

- MAXINPUT message size
- MAXOUTPUT message size
- SCREEN
- WRAPAROUND
- LINEWIDTH
- PAGESIZE
- PAGECOUNT

## DCWRITE Errors

The following DCWRITE errors can be reported for the MOVE/ADD/SUBTRACT STATION request when MSG[0].[25:01] = 1:

| Number | Meaning |
|---|---|
| 73 | The message was less than or equal to nine words in length although MSG[0].[25:01] = 1. |
| 79 | The line name or terminal name was submitted in a bad format or without a terminating period. |
| 92 | During the reconfiguration, the station information was not found in the DATACOMINFO file. |
| 98 | There was no room on the target line for the station being moved or added. |
| 107 | The MSG[8].[31:08] (NDLIILINEINX) value or the MSG[8].[39:08] (NDLIITERMINX) value indicated a word outside the request message. |
| 108 | The terminal specified by the terminal name was not found. |
| 110 | The line specified by the line name was not found. |
| 113 | The algorithm identities associated with the station, the named line (if present), the source line (if appropriate), and the target line were not the same. |
| 114 | The switchable attributes (PRIVATE, DIALOUT, and AUTOANSWER) of the target line do not conform to those of the named line (if present), or to those of the source line in the case of a station move with no named line. |
| 125 | The ADD STATION or MOVE STATION request included a named line when the target line was already supporting stations. |
| 126 | The SUBTRACT STATION request included a terminal name and MSG[0].[25:01] = 1. (The change of station logical attributes is appropriate only when not making a null line assignment.) |
| 171 | During the reconfiguration, an I/O error occurred or the link to the DATACOMSUPPORT library failed. |
| 173 | During the move or subtract, the station could not be made NOT READY, and the line/LSP could not be cleared. |
| 174 | During the reconfiguration, the target NSP did not respond correctly while structures were being added to it. |
| 195 | A bit-oriented line cannot be changed into a character-oriented line and vice versa. |

### Examples

```
ALLOCATE(MSG,8);
MSG[0] := LSN & 130 [47:8] & 1 [39:16];
MSG[7] := 0 & 1 [23:1] &89L [22:15];
RESULT := DCWRITE(MSG);

ALLOCATE(MSG,11);
MSG[0] := LSN & 130 [47:8] & 3 [39:16];
MSG[7] := 0 & 1 [23:1] & DL [22:15];
MSG[8] := 0 & 20 [23:8] % NEW ADAPTER TYPE
            & 9 [15:8]; % NEW TERMINAL NAME INDEX
REPLACE POINTER(MSG[9],8) BY "TERMINAL29.";
RESULT := DCWRITE(MSG);
```

# UPDATE LINE ATTRIBUTES (DCWRITE Type = 131)

### Required Parameters/Fields

The minimum message size for the UPDATE LINE ATTRIBUTES DCWRITE function is nine words. A message parameter and the following are required:

| Word | Field | Value | Description |
|---|---|---|---|
| 0 | [47:08] | 131 | Type. |
| 7 | [23:24] | | DL number, as follows: |
| | [23:01] | 1 | |
| | [22:07] | | Relative NSP number. |
| | [15:08] | | Line number. |
| 8 | [31:08] | | NDLIILINEINX is the index within MSG of the name of the line. |

The word MSG[NDLIILINEINX] is the name of the line.

### Explanation

The UPDATE LINE ATTRIBUTES DCWRITE function allows an MCS to modify operations parameters for a particular line. The line update is performed for the specified line, and if any stations exist on the line, the line is left READY.

The only field in MSG[8] that is not ignored is [31:08] (NDLIILINEINX). If this field contains 0, error 107 is returned to indicate that the request was incomplete. The use of the line name referenced by NDLIILINEINX allows the requestor to modify the connection characteristics of a line in a manner consistent with the hardware strappings on the line adapter.

The following is a list of physical attributes copied from the named line to the target line:

- CRC polynomial
- CRC polynomial initial setting
- CRC polynomial final setting
- Mode (async, sync, or bit-sync)
- Vertical parity (mode = async or sync)
- Horizontal parity (mode = async or sync)
- BCS type (mode = async or sync)
- Stop bits (mode = async)
- Bit rate (that is, line speed)
- Sync character (mode = sync)

- DLE character (mode = sync)
- Address mode (mode = bit-sync)
- Control mode (mode = bit-sync)
- CLASS number
- Translate table
- Receive delay
- Transmit delay
- Receive timeout
- DPR delay for ACU
- Set digit delay for ACU
- Secondary address (mode = bit-sync)
- Function (mode = bit-sync)
- Disconnect action on loss of DSR
- Receive ready
- End of number for ACU

## DCWRITE Errors

The following DCWRITE errors can be returned:

| Number | Meaning |
|---|---|
| 73 | The message was less than or equal to nine words in length. |
| 76 | The value of the DLS number indicated an invalid NSP-relative number. |
| 77 | The value of the DLS number indicated an invalid line number. |
| 79 | The line name was submitted in a bad format or without a terminating period. |
| 91 | The target line had stations on it, and none of them were associated with the requestor MCS. None of these stations has to be attached to the requestor MCS: the MCS number associated with a station has to be the same as that of the requestor MCS for at least one of the stations on the target line. |
| 107 | The value in MSG[8].[31:08] (NDLIILINEINX) was 0 or indicated a word outside the message. |
| 110 | The line specified by the line name was not found. |
| 113 | The algorithms associated with the target line and the named line were not the same. |
| 114 | There was a mismatch between switch attributes of the named line and of the target line: both must be either PRIVATE, DIALOUT, or AUTOANSWER. |
| 171 | During the reconfiguration, an I/O error occurred or the link to the DATACOMSUPPORT library failed. |
| 176 | There was an incompatibility found between the named line and the target line other than those reported specifically. |
| 195 | A bit-oriented line cannot be changed into a character-oriented line and vice versa. |

### Example

```
ALLOCATE(MSG,11);
MSG[0] := 0 & 131 [47:8];
MSG[7] := 0 & 1 [23:1] & DL [22:15];
MSG[8] := 0 & 2 [22:7] & 9 [7:8];
REPLACE POINTER(MSG[9],8) BY "NEWMODEM.";
RESULT := DCWRITE(MSG);
```

# Section 6
# MCS Result Message Formats

This section contains the following topics:

- General result message formats
- Specific result message formats
- Error result message format

## General Result Message Format

Result messages generated by the data comm system can be inserted periodically into the primary queue or current queue of an MCS. The class of each message is identified by a unique value in the class field of the message. In most cases, the class indicates the meaning of the result message and the format of the message. In one case (Result Class = 99), the class is supplemented by the result-byte index to indicate the meaning and format of the message.

In many cases, a result message is generated when the operating system or the NSP merely modifies certain fields within the original DCWRITE request message. In other cases, the result message is created spontaneously.

The format of these result messages varies somewhat from class to class, but generally appears as in Table 6–1.

**Table 6–1. General Message Format**

| Word | Field | Description |
|---|---|---|
| [0] | [47:08] | Class field. |
| | [39:16] | Variant field. |
| | [23:24] | LSN-DLS number field. |
| [1] | [47:08] | Result-byte index field. |
| | [39:08] to [32:01] | TOGGLES: [39:01] = TOGGLE [7], [38:01] = TOGGLE [6], and so on until [32:01] = TOGGLE [0]. |
| | [31:08] | Last error flag set in MSG[1].[23:24] (might not always be turned on). |
| | [23:24] | Error flag field. |

**Table 6–1. General Message Format**

| Word | Field | Description |
|---|---|---|
| [2] | [47:08] | Retry count field. |
| | [39:16] | Text size field. |
| | [23:24] | Transmission number field. |
| [3] | [47:24] | Time field. |
| | [23:08] | Tally [0]. |
| | [15:08] | Tally [1]. |
| | [07:08] | Tally [2]. |
| [4] | [47:24] | Message number field. |
| | [23:24] | Original DCWRITE type (the original value in MSG[0].[47:24] when the DCWRITE function was called). |
| [5] | [27:01] | 1 = Sequence number present. |
| | [26:27] | Sequence number. |
| [6] to [N] | | Text |

The meanings of the fields listed in Table 6–1 follow. These meanings apply to the general case. Meanings that differ in particular result messages are presented in the discussions of the individual result messages later in this section.

## Class Field (MSG[0].[47:08])

The class field determines the way in which the remainder of the message is interpreted. For example, if MSG[0].[47:08] = 0, the message contains “good” (error-free) input from the NSP. The values for this field are presented in Table 6–2.

**Table 6–2. Message Classes**

| Class | Meaning |
|---|---|
| 00 | GOOD INPUT |
| 01 | STATION EVENT |
| 02 | FILE OPEN |
| 03 | OBJECT JOB OUTPUT |

Table 6–2. Message Classes

| Class | Meaning |
|---|---|
| 04 | FILE CLOSE |
| 05 | GOOD RESULT |
| 06 | RECALLED MESSAGE |
| 07 | SWITCHED LINE STATUS RESULT |
| 08 | LSP EXCHANGE RESULT |
| 09 | LINE STATUS RESULT |
| 10 | LINE SWAP RESULT |
| 11 | MOVE/ADD/SUBTRACT STATION RESULT |
| 12 | DLS UPDATE RESULT |
| 13 | MESSAGE FROM ANOTHER MCS |
| 14 | NSP TERMINATED |
| 15 | INTERROGATE STATION RESULT |
| 16 | TRANSFER STATION CONTROL RESULT |
| 17 | SEND TO MCS RESULT |
| 18 | SEND TO STATION RESULT |
| 19 | UPDATE LINE ATTRIBUTES |
| 21 | MESSAGE FROM CONTROLLER |
| 24 | LINE INTERROGATE RESULT |
| 25 | OBJECT JOB INPUT REQUEST |
| 29 | INTERCEPTED MESSAGE RESULT |
| 30 | NSPINITIALIZED RESULT |
| 32 | POWER OFF PENDING RESULT |
| 80 | ODT MODE SWITCH NOTICE RESULT |
| 81 | INPUT FROM AN ODT RESULT |
| 99 | ERROR MESSAGE |

## Variant Field (MSG[0].[39:16])

The variant field is used for qualification, variations, or additional information concerning certain message types. Specific values or interpretations of the variant field are presented in this section wherever applicable.

## LSN Field (MSG[0].[23:24])

Most result messages received by an MCS contain an LSN in this field, but some can contain a DLS number or other quantity depending on the type of result message.

## Result-byte Index Field (MSG[1].[47:08])

Most nonerror messages received by an MCS have a result-byte index value equal to 0. A nonzero value usually implies an error detected by the NSP (see interpretation of this field for the Class = 99 error result message). Deviations from this practice are described in this section wherever applicable.

## Toggle Field (MSG[1].[39:08])

An NDLII algorithm can store information in this field by setting the REQUEST.TOGS attribute. The interpretation of these bits by the MCS is determined by programming convention and agreement between the MCS author and the NDLII algorithm author.

## Last Error Flag Set Field (MSG[1].[31:08])

In most cases, the last error flag set field contains the bit number of the bit most recently turned on in the error flag field (MSG[1].[23:24]) as a binary value. That is, if bit 6 was the last bit turned on by the NSP in the error flag field, MSG[1].[31:08] = 6.
MSG[1].[31:08] = 4FF if MSG[1].[23:24] = 0.

## Error Flag Field (MSG[1].[23:24])

The error flag field indicates any errors or situations encountered by the NSP while attempting to honor this particular request (often, but not necessarily, as the result of an MCS-requested DCWRITE). Depending on several factors, bits can be turned on in this field even in messages that are not error messages. For example, an error situation might have occurred, but the NDLII algorithm was able to achieve recovery with subsequent retry measures, or the NDLII algorithm might have decided to perform a SENDHOST input instead of a SENDHOST error on encountering an irrecoverable error condition. (This practice can be useful in debugging NDLII algorithms, but is not recommended for final code.) The interpretation of the error flag field is shown in Table 6–3.

Note that the ERROR RESULT message (class = 99) also has an error result field.

### Table 6–3. Error Flag Values

| Bit | Value | Description |
|---|---|---|
| 0 | 1 | A timeout occurred (NDLII TIMEOUT). |
| 1 | 1 | A stop-bit error occurred (NDLII STOPBIT). |
| 2 | 1 | An LSP-buffer overflow occurred (NDLII OVERRUN). |
| 3 | 1 | A break-on-input occurred (NDLII FRAMEABORT). |
| 4 | 1 | A disconnect occurred (NDLII DISCONNECT). |
| 5 | 1 | A break-on-output occurred (NDLII FRAMEABORT). |
| 6 | 1 | A vertical character parity error occurred (NDLII PARITY). |
| 7 | 1 | A horizontal parity (BCC or CRC) error occurred (NDLII BCSERROR). |
| 8 | 1 | An address-character error occurred (NDLII ADDRESSERROR). |
| 9 | 1 | A transmission-number error occurred (NDLII TRANSERROR). |
| 10 | 1 | A format error on input occurred (NDLII FORMATERROR). |
| 11 | 1 | An output was not acknowledged (NDLII NAKFLAG). |
| 12 | 1 | The first character of the message contains the control character of the station (as defined in NDLII or NDLII bit-variable CONTROLMESSAGE, or set by MCS). |
| 13 | 1 | A WRU (who-are-you) was received (NDLII WRUFLAG). |
| 14 | 1 | A sequence number overflow occurred (NDLII SEQUENCEERROR). |
| 15 | 1 | The message is to be acknowledged (ACK). |
| 16 | 1 | A NAK-ON-SELECT occurred (NDLII NAKONSELECT). |
| 17 | 1 | An END-OF-BUFFER (message overflow) occurred (NDLII ENDOFBUFFER). |
| 18 | 1 | A LOSS-OF-CARRIER occurred (NDLII LOSSOFCARRIER). |
| 20 | 1 | A station or line is NOT READY. |
| 21 | 1 | An idle condition was detected on a bit-oriented line (NDLII IDLEDETECT). |
| 22 | | (Reserved for expansion.) |
| 23 | | (Reserved for expansion.) |

## Retry Count Field (MSG[2].[47:08])

The retry count field contains the number of retries that remain at the time the NSP finished the request. The validity and value of this field rest on the programming conventions employed by the author of the NDLII algorithm. Normally, the NDLII programmer decreases the retry count by 1 for each unsuccessful attempt at an operation in the NDLII algorithm until RETRY equals 0; a SENDHOST error or FINISHREQUEST statement is then performed.

## Text Size Field (MSG[2].[39:16])

The text size field is valid for messages that constitute input (implied or otherwise). This field contains the number of bytes of meaningful text stored as input by the NDLII algorithm (starting in the left character position of MSG[6], 6 characters per word).

## Transmission Number Field (MSG[2].[23:24])

The transmission number field is valid for messages that constitute input (implied or otherwise) from the NSP (a station) where the station in question uses transmission number conventions. The transmission number received with this input is stored as three EBCDIC numeric characters.

## Time Field (MSG[3].[47:24])

The time field, if 0 initially, contains the time (in 60ths of a second) at whichthe NSP returned the result or at which the central system received this message from the NSP. The form of this value is equivalent to that obtained as TIME(1) by the ALGOL TIME function intrinsic. (Refer to “Arithmetic Intrinsic Names” in the *ALGOL Programming Reference Manual, Volume 1: Basic Implementation.*)

## TALLY[0], TALLY[1], and TALLY[2]

These tally fields are MSG[3].[23:08], MSG[3].[15:08], andMSG[3].[07:08], respectively. The interpretation of these fields is analogous to that for the togglefield (MSG[1].[39:08]).

## Message Number Field (MSG[4].[47:24])

The message number field is discussed in “DCWRITE Message Format” in the “DCWRITE Information” section. The values in this field for messages other than those generated by an MCS by use of the DCWRITE function are undefined except for the OBJECT JOB OUTPUT (Result Class = 3) message or the RECALLED MESSAGE (Result Class = 6) message that corresponds to a (Result Class = 3) message; in those cases, the message number field contains the FRSN of the originating file.

## Original DCWRITE Type Field (MSG[4].[23:24])

If this message originally came from an MCS by use of the DCWRITE function (or as a result of object job output, where the MCS elected not to participate in object job I/O), the original DCWRITE type field (MSG[0].[47:08]) and DCWRITE variant field (MSG[0].[39:16]) appear in this field. Input messages obtained because a station was enabled have the ENABLE INPUT type value (35) in [23:08] of this field.

## Sequence Number Present Field (MSG[5].[27:01]

A value of 1 in this field indicates that there is a valid value in the sequence number field.

## Sequence Number Field (MSG[5].[26:27])

The sequence number field is set from the sequence-result field in the result message from the NSP, if the result class is equal to SEQUENCE RESULT.

## Text (beginning at MSG[6])

Text appears in EBCDIC character form, left justified, starting in relative word 6 of the message, six characters per word. For input if translation is necessary, the NSP and LSP perform the translation.

### Text (beginning at MSG[6])

## Specific Result Message Formats

The following describes the ways in which the formats and meanings of the various result message classes differ from the general case. Refer to the beginning of this section for the description of the general case.

The current queue of a station is, by default, the primary queue of the controlling MCS until (or unless) a CHANGE CURRENT QUEUE (Type = 32) DCWRITE is performed that names a queue other than the primary queue of the MCS. Also, in the case in which the current queue of the station and the primary queue of the MCS are not the same, the settings of the option bits MSG[0].[39:16] in the STATIONATTACH (Type = 1), CHANGE CURRENT QUEUE (Type = 32), and STATIONASSIGNMENT TO FILE (Type = 64) DCWRITE types assume an important role in the determination of ultimate queue destinations for the various types of result messages that an MCS can receive.

In the following descriptions of specific result message formats, this important information is not restated except in cases of special interest. As an aid to clarification and understanding of the following material, you should occasionally review the option bits (refer to “STATION ATTACH (DCWRITE Type = 1)” in the “DCWRITE Information” section). It is also important to realize that an MCS might never see result messages of certain classes for a given station because the MCS might have elected to have the messages discarded and not returned.

DCALGOL is a constantly changing language; changes are made primarily through the addition of new functions, new statements, new DCWRITE types, and new result message classes. The new DCWRITE types cannot cause a problem with an old MCS. However, each MCS should be written so that new result message classes do not cause problems. That is, if an MCS encounters a result message in a current queue or its primary queue with a message class that is unrecognized by the MCS, the MCS should discard the result message and continue processing.

# GOOD INPUT RECEIVED (Result Class = 0)

### Message format

| Word | Field | Value | Description |
|---|---|---|---|
| 0 | [47:08] | 0 | Class. |
| | [29:01] | 1 | More blocks to follow this one. |
| | [23:24] | | LSN. |

### Explanation

Receipt of a message of Class = 0 in the current queue of the station is a direct result of an NDLII algorithm performing a SENDHOST input. The input might have been initiated because of a READ-ONCE ONLY (Type = 34) DCWRITE (in which case, MSG[4].[23:08] = 34) or because the station has been enabled (in which case, MSG[4].[23:08] = 35). In the former case, the GOODINPUT RECEIVED message space is the same as presented in the READ-ONCE ONLY DCWRITE. If the station is not enabled for input and no outstanding READ-ONCE ONLY requests exist in the station queue of the station, all further input information received from a station is discarded until the station is enabled or another READ-ONCE ONLY request is issued for the station.

Text, if any, appears in MSG[6] through MSG[N] and is valid information for a total-of-text size field (MSG[2].[39:16]) characters.

# STATION EVENT (Result Class = 1)

## Message format

| Word | Field | Value | Description |
|---|---|---|---|
| 0 | [47:08] | 1 | Class. |
| | [39:16] | 0 | Input event. |
| | | 3 | New station activity. |
| | [23:24] | | LSN. |
| 1 | [31:08] | | The switched status of the line (if this message is a new station activity result). |
| | [23:24] | | The error flag field if this MESSAGE is an input event result, or the DLS number if this message is a new station activity result. |
| 6 to end | | | The input text is contained in these fields if the message is an input event. The station name is contained in these fields in display form (for example, TTY/ONE) if the message is caused by new station activity. |

## Explanation

When an MCS receives a STATION EVENT message from the system, a noteworthy event has occurred on one of the stations that this MCS controls.

An input event (MSG[0].[39:16] = 0) result message signifies to the MCS that at least one of the following four cases must have occurred:

1. A station break on input occurred. Bit 3 is turned on in the error flag field (MSG[1].[23:24]).
2. A station break on output occurred. Bit 5 is turned on in the error flag field (MSG[1].[23:24]).
3. The NSP received the station NDLII-defined (or established by the MCS) control character as the first character of the input text. Bit 12 is turned on in the error flag field (MSG[1].[23:24]).
4. The station was disconnected.

Usually, more than one of these four conditions can be flagged in the error flag field for any given input event message; however, this capability is determined to some extent by the NDLII programmer.

The action that the MCS can take when it receives an input event message depends on individual installation requirements. For example, if the MCS receives an input event message in which MSG[1].[12:01] = 1 (the message contains the station control character as the first character), the MCS might choose to treat the message text in a different manner than other input messages.

If MSG[2].[39:16] does not equal 0, text exists in the message.

A new station activity (MSG[0].[39:16] = 3) message signifies to the MCS that a station it controls has become active and attached where it had previously been neither active nor attached. Receipt of the new station activity message is an announcement of implicit station attachment. (Refer to "STATION ATTACH (DCWRITE Type = 1)" in the "DCWRITE Information" section and "FILE OPEN (Result Class = 2)" in this section.) Therefore, a subsequent STATION ATTACH DCWRITE is unnecessary unless options other than the default options are to be exercised (refer to the variant in "STATION ATTACH (Result Type = 1)" in the "DCWRITE Information" section). The name of the station in display form appears beginning in MSG[6] and is left-justified. The DLS number of the station appears in MSG[1].[23:24], the switched status for the line with which the station is associated appears in MSG[1].[31:08], andthe LSN of the station appears in MSG[0].[23:24]. The MCS then proceeds to deal with the station in the same way that the MCS would deal with a station for which a successful STATION ATTACH DCWRITE had been performed.

The message for new station activity is generated by the operating system when it receives a message for a station that was previously unrecognized. Both messages are then placed in the primary operating system queue. The new station activity message appears first in the queue.

# FILE OPEN (Result Class = 2)

## Message Format

| Word | Field | Value | Description |
|---|---|---|---|
| 0 | [47:08] | 2 | Class. |
| | [26:01] | 1 | File is a direct file. |
| | [25:02] | | File output tanking specification. |
| | | 0 | Unspecified. |
| | | 1 | None. |
| | | 2 | SYNC. |
| | | 3 | ASYNC. |
| | [23:24] | | LSN. |
| 1 | [47:08] | | Switched line status. |
| 2 | [39:16] | | Text size field. |
| 3 | [47:24] | | Time of day field. |
| 6 | [45:14] | | Program mix number of the object job. This field is 0 if the controlling MCS is an expanded capacity MCS. Since this field is obsolete, refer to word 8, field [15:08]. |
| | [25:02] | | File MYUSE attribute value. |
| | | 1 | Input. |
| | | 2 | Output. |
| | | 3 | IO. |
| | [23:24] | | FRSN, broken down as follows: |
| | [23:10] | | File number (created at file open time). |
| | [13:14] | | RSN in the file. |
| 7 | [47:08] | | MCS number of the current controlling MCS. |
| | [39:16] | | Record index into the DATACOMINFO file for the station. |
| | [23:24] | | DLS number. |
| | [23:01] | 0 | The station name has no current line assignment. |
| 8 | [47:08] | | The word index into MSG where the station name (title) is found. |
| | [39:08] | | The word index into MSG where the file INTNAME is found. |
| | [31:08] | | The word index into MSG where the title of the program opening the file is found. |
| | [23:08] | | The word index into MSG where the file title is found. |

| Word | Field | Value | Description |
|---|---|---|---|
| | [15:08] | | The word index into MSG where the job number of the program that opens the remote file is contained. Word index + 1 contains the mix number of the object job. |
| 9 to end | | | The station name, file INTNAME program title, file title, and job number are contained in these words. |

### Explanation

Receipt of a FILE OPEN message by an MCS indicates the opening of an object job file that contains one of the stations controlled by the MCS. The MCS receives a FILE OPEN message for each station contained in the file over which it has control. The MCS then performs a STATIONASSIGNMENT TO FILE (Type = 64) DCWRITE. In some cases such as a broadcast write, the object program can be held up until a STATIONASSIGNMENT TO FILE (Type = 64) DCWRITE has been received for each station in the file.

The value of the MYUSE file attribute informs the MCS that the file is opened either input-only, output-only, or I/O. The job number occupies an entire word in binary format. The station name, the file INTNAME, and the program title begin at word boundaries in MSG and appear in display form (for example, "TTY/ONE.") as a left-justified character string. Thus, a pointer expression that points to the first character of the station name would appear as follows:

```
POINTER(MSG[MSG[8].[47:8]],8)
```

The text size field message is defined to be the number of characters from the beginning of MSG[6] to the end of MSG[N], although this field usually is not a string of legitimate characters. This practice allows a consistent calculation of message size based on the textsize field. The station name (title), the file INTNAME, the program title, and the file title are terminated by a period followed by a fill of hexadecimal zeros, if necessary, up to the next word boundary.

## Station Transfer File Open

A FILE OPEN message is generated and sent to the MCS referenced by PSEUDOMCSNRF when stations are transferred and are performing logical I/O.

### Message Format

| Word | Field | Value | Description |
|---|---|---|---|
| 0 | [47:08] | | FILEOPENMSG. |
| | [26:01] | 1 | Flag for audit and debugging use. |
| | [23:24] | | LSN of transferred station. |
| 2 | [39:16] | 3 | Text size. |
| 6 | [23:24] | | FRSN of the LSN. |

## OBJECT JOB OUTPUT (Result Class = 3)

### Message Format

| Word | Field | Value | Description |
|---|---|---|---|
| 0 | [47:08] | 3 | Class. |
| | [39:16] | | Carriage control bits. |
| | [23:24] | | LSN. |
| 2 | [39:16] | | Number of bytes of text to be sent to the station. |
| 4 | [47:24] | | FRSN. |
| 6 to end | | | Text (if any). |

### Explanation

Receipt of the OBJECT JOB OUTPUT message class by an MCS is a direct result of an object job performing a WRITE statement directed to a station under the control of the MCS. An MCS does not receive messages of this class unless bit 29 in the variant field was allocated by the MCS in the most recent CHANGE CURRENT QUEUE (Type = 32), or a STATIONASSIGNMENT TO FILE (Type = 64) DCWRITE was performed for this station to indicate that the MCS wishes to participate in I/O for the station. The format of this message is identical to the WRITE (Type = 33) DCWRITE message format. Changing the type to WRITE (DCWRITE Type = 33) and passing the message to DCWRITE causes the output to be sent to the station.

## FILE CLOSE (Result Class = 4)

### Message Format

| Word | Field | Value | Description |
|---|---|---|---|
| 0 | [47:08] | 4 | Class. |
|  | [23:24] |  | LSN. |
|  | [25:01] | 1 | The second close result for asynchronous tanking. |
|  | [24:01] | 1 | The task has executed CLOSE but output is tanked; the second CLOSE result is sent when all output has been completed. |
| 6 | [23:24] |  | FRSN. |

The rest of the format is identical to that of FILE OPEN (Result Class = 2) result.

### Explanation

Receipt of the FILE CLOSE message by an MCS constitutes notification of the closing of an object job file that included this station. The MCS receives a FILE CLOSE message for each station in the file. The FRSN for the station in this file is no longer valid; however, the station remains attached to the MCS. The FILE CLOSE message contains all the information about the station that is contained in the FILE OPEN (Result Class = 2) message.

Action by the MCS is a matter of program convention. Unlike the FILE OPEN message, the data comm system software neither expects nor requires any particular DCWRITE response on the part of the MCS. However, if an MCS has postponed opening other files on this station, the MCS can use the close result to signal itself that another file can now be opened.

## Station Transfer FILE CLOSE

A FILE CLOSE message is generated and sent to the MCS referenced by PSEUDOMCSNRF when stations are transferred and performing logical I/O.

### Message Format

| Word | Field | Value | Description |
|---|---|---|---|
| 0 | [47:08] | | FILECLOSEMSG. |
| | [26:01] | 1 | Flag for audit and debugging use. |
| | [23:24] | | LSN of the transferred station. |
| 2 | [39:16] | 3 | Text size. |
| 6 | [23:24] | | FRSN of LSN. |

# GOOD RESULTS (Result Class = 5)

### Message Format

| Word | Field | Value | Description |
|---|---|---|---|
| 0 | [47:08] | 5 | Class. |
| | [23:24] | | LSN. |
| 4 | [23:24] | | The original contents of MSG[0].[47:24] of the DCWRITE request prior to the presentation of the message to the NSP are contained in this word. |

### Explanation

If an MCS chooses to receive all results from the NSP for a station, the results are returned with a Class of 5 in the current queue (refer to the variant field in the "STATION ATTACH (DCWRITE Type = 1)" in the "DCWRITE Information" section). The type and variant fields (MSG[0].[47:24]) of the original DCWRITE message are found in each result message in MSG[4].[23:24]. If the original DCWRITE type (found in MSG[4].[23:08]) is one of the following switched line requests, the message result returned is the SWITCHED STATUS (Type = 101) message:

- DIALOUT (DCWRITE Type = 98)
- DISCONNECT (DCWRITE Type = 99)
- ANSWER THE PHONE (DCWRITE Type = 100)
- INTERROGATE SWITCHED STATUS (DCWRITE Type = 101)

However, if the original DCWRITE type (found in MSG[4].[23:08]) is the particular switched line request SET/RESET AUTOANSWER (DCWRITE Type = 102), the message result returned is the GOOD RESULTS (Result Class = 5) message.

The error flag field can have one or more bits turned on in a result message, but the request itself was honored by the NSP. That is, the NSP recovered successfully from any conditions that it encountered. Results from OBJECT JOB OUTPUT (Result Class = 3) messages are also returned to the MCS, even if the MCS chooses not to participate in object job I/O but chooses to receive all results.

GOOD RESULTS messages are interpreted as the assurance that the operation requested of the NSP was successfully completed. Examples are the MAKE STATION READY (Type = 37) DCWRITE or the ENABLE INPUT (Type = 35) DCWRITE. DCWRITE types that cause the creation of a GOOD RESULTS (Result Class = 5) message are as follows:

- CHANGE CURRENT QUEUE (Type = 32) DCWRITE (if the text is not empty)
- WRITE (Type = 33) DCWRITE
- ENABLE INPUT (Type = 35) DCWRITE
- DISABLE INPUT (Type = 36) DCWRITE

- MAKE STATION READY/NOT READY (Type = 37) DCWRITE
- SET APPLICATION NUMBER (Type = 38) DCWRITE
- SET CHARACTERS (Type = 39) DCWRITE
- SET TRANSMISSION NUMBER (Type = 40) DCWRITE
- RECALL MESSAGE (Type = 41) DCWRITE
- STATION DETACH (Type = 42) DCWRITE
- SET/RESET LOGICALACK (Type = 43) DCWRITE
- ACKNOWLEDGE (Type = 44) DCWRITE or WRITE AND RETURN (Type = 46) DCWRITE
- NULL STATION REQUEST (Type = 48) DCWRITE
- SET/RESET SEQUENCE MODE (Type = 49) DCWRITE
- SET/RESET AUTOANSWER (Type = 102) DCWRITE

## Variant Field in Response to RECALL MESSAGE

If the original DCWRITE was RECALL MESSAGE (Type = 41), the variant field MSG[0].[39:16] indicates whether or not any RECALLED MESSAGE (Result Class =6) messages are recalled, as follows:

| Word | Field | Value | Description |
|---|---|---|---|
| 0 | [39:16] | 0 | No messages exist to be recalled. |
| | | 1 | One or more messages that exist in the station queue have been recalled. These messages are next in the current queue of the station. |

# RECALLED MESSAGE (Result Class = 6)

### Message Format

| Word | Field | Value | Description |
|---|---|---|---|
| 0 | [47:08] | 6 | Class. |
|  | [39:16] | 0 | This message is the last of those recalled by the RECALL MESSAGE (Type = 41) DCWRITE. |
|  |  | 1 | This message is not the last of those recalled by the RECALL MESSAGE (Type = 41) DCWRITE. |
|  | [23:24] |  | LSN. |
| 4 | [23:24] |  | The original contents of MSG[0].[47:24] of the DCWRITE request prior to the presentation of the message to the NSP. If the original content of MSG[0].[47:08] prior to the presentation of the message to the NSP was OBJECT JOB OUTPUT (Result Class = 3), MSG[4].[47:24] contains the FRSN. |

### Explanation

If an MCS decides to discontinue output to a station (for example, because of an excessively large number of errors) but decides to save all outstanding requests queued for the station, the MCS can perform a RECALL MESSAGE (Type = 41) DCWRITE. The queued requests are removed from the station queue on a first-in, first-out basis and are placed in the current message queue of the station with the result type equal to 6, thus preserving the integrity of time-ordered sequencing.

# SWITCHED STATUS RESULT (Result Class = 7)

## Message Format

The format for this message is included in the description under "SWITCHED STATUS FORMAT" in "ERROR RESULT (Result Class = 99)" in this section.

## Explanation

The origin of the SWITCHED STATUS RESULT message one of the following:

- The result of the DIALOUT (Type = 98) DCWRITE, the DISCONNECT (Type = 99) DCWRITE, the ANSWER THE PHONE (Type = 100) DCWRITE, or the INTERROGATESWITCHED STATUS (Type = 101) DCWRITE
- An unrequested change of (switched) status automatically reported by the NSP such as an unexpected disconnect or connection by the autoanswer capability

## LSP EXCHANGE RESULT (Result Class = 8)

### Message Format

| Word | Field | Value | Description |
|---|---|---|---|
| 0 | [47:08] | 8 | Class. |
| | [39:16] | | As originally presented to the EXCHANGE LSPS (Type = 129) DCWRITE function. |
| | [22:07] | | Relative number from which the LSPs were transferred. |
| | [15:16] | | Mask of the LSPs that were transferred and that initialized successfully. |
| 1 | [47:08] | 0 | LSP exchange successfully completed. |
| | | not 0 | Interpreted as for DCWRITE error values. |
| 6 | [31:08] | | NSPLSPINX is the index within MSG of the list of new NSP/LSP hardware unit numbers and DLS numbers. |
| | [23:08] | | The exchanged relative NSP number. |
| | [15:08] | | The number of LSPs exchanged and number of additional words, starting with MSG[7]. |
| | [07:08] | | The MCS number of the requesting MCS. |
| 7 and on | | | Contains the following for each LSP: |
| | [39:08] | | The relative LSP number. |
| | [31:16] | | The mask of prior READY-made status for each line (adapter) on LSP (0 = READY, 1 = NOT READY); bit number = line (adapter) number + 16. |
| | [15:16] | | The mask of valid lines (adapters) for the LSP; bit number = line (adapter) number. |
| NSPLSPINX to end | | | Contains the following for each LSP exchanged: |
| | [47:16] | | The NSP hardware unit number on which the LSP is now located. |
| | [31:16] | | The LSP hardware unit number. |
| | [15:01] | 1 | The NSP is available. |
| | | 0 | The NSP is not initialized. |

| Word | Field | Value | Description |
|---|---|---|---|
|  | [14:07] |  | The relative number of the NSP. |
|  | [07:04] |  | The relative number of the LSP. |
|  | [03:04] | 0 |  |

These words are presented in the same order as those beginning at MSG[7]. The LSP numbers in those words represent the LSP-relative numbers of the LSPs before the exchange.

### Explanation

All MCSs affected as a result of LSP exchanging are sent a Class = 8 result message. If LSP exchanging cannot be accomplished for any reason, only the requesting MCS receives a Class = 8 result message and MSG[1].[47:08] is nonzero. (The interpretation of this error value can be derived from the DCWRITE error values.) If MSG[1].[47:08] is 0, the LSP exchange was successfully accomplished and information concerning the actual LSPs involved in the exchange is found in MSG[6] through MSG[N] (where N = 6 + MSG[6].[15:08]). Line numbers, as indicated by the L in the DL number or the DLS number, are actually 8-bit concatenations of 4-bit adapter numbers (where the relative LSP number field comprises the four high-order bits, and the adapter number comprises the four low-order bits).

The Class = 8 result information (from EXCHANGE LSPS (Type = 129) DCWRITE) can be noted or discarded at the discretion of an MCS.

# LINE STATUS CHANGE RESULT (Result Class = 9)

### Message Format

| Word | Field | Value | Description |
|---|---|---|---|
| 0 | [47:08] | 9 | Class. |
| | [23:24] | | LSN. |
| 1 | [20:01] | 0 | The line is READY. |
| | | 1 | The line is NOT READY. |
| 7 | [23:24] | | DL number. |

### Explanation

A LINE STATUS CHANGE RESULT message is always generated by the NSP as a part of its response to the MAKE LINE READY DCWRITE (even if the line is READY). A LINE STATUS CHANGE RESULT message is generated by the NSP as a part of its response to the MAKE LINE NOT READY DCWRITE if, and only if, the line was READY prior to the MAKE LINE NOT READY DCWRITE.

Only one Class = 9 result message is generated. If the result was generated by a MAKE LINE NOT READY (Type = 96) DCWRITE that specified a DL number rather than an LSN, the result is returned to the MCS that controls the lowest-numbered valid station on the line (not necessarily the MCS that performed the original DCWRITE). The result is routed by the operating system to the current queue of that station if the controlling MCS indicated that it is to receive Class = 9 result messages. Otherwise, the operating system discards the message.

# SWAP LINE RESULT (Result Class = 10)

### Message Format

| Word | Field | Value | Description |
|---|---|---|---|
| 0 | [47:08] | 10 | Class. |
|  | [39:40] |  | As originally presented to SWAP LINE (Type = 128) DCWRITE. |
| 1 | [47:08] | 0 | The line swap successfully completed. |
|  |  | not 0 | Interpreted as for DCWRITE error values. |
| 6 |  |  | The MCS number of the requestor. |
| 7 | [47:24] |  | The DL number of the first or source line. |
|  | [23:24] |  | The DL number of the second or target line as presented in the original request. |
| 8 | [47:16] |  | The NSP hardware unit number of the source line. |
|  | [31:16] |  | The LSP hardware unit number of the source line. |
| 9 | [47:16] |  | The NSP hardware unit number of the target line. |
|  | [31:16] |  | The LSP hardware unit number of the target line. |

### Explanation

All MCSs affected as a result of line swapping receive a SWAP LINE RESULT message. If line swapping cannot be accomplished for any reason, only the requesting MCS receives this message and MSG[1].[47:08] is not zero. If MSG[1].[47:08] = 0, the line swap was successfully accomplished.

In addition, all affected MCSs receive DLS UPDATE (Result Class = 12) result messages for all stations that have had their DLS number changed as a result of the line swap.

# MOVE/ADD/SUBTRACT STATION RESULT (Result Class = 11)

### Message Format

| Word | Field | Value | Description |
|---|---|---|---|
| 0 | [47:08] | 11 | Class. |
|  | [39:40] |  | As originally presented to the MOVE/ADD/SUBTRACT (Type = 130) DCWRITE. |
| 1 | [47:08] | 0 | MOVE/ADD/SUBTRACT successfully completed. |
|  |  | not 0 | Interpreted as for DCWRITE error values. |
| 6 |  |  | The MCS number of the requestor. |
| 7 | [47:24] |  | The DLS number of the station after the MOVE or ADD; 0 for SUBTRACT. |
|  | [23:24] |  | The DLS number of the station before reconfiguration; 0 for an ADD. |
| 8 | [47:16] |  | If a MOVE or an ADD, the NSP hardware unit number of the new location of the station. |
|  | [31:16] |  | If a MOVE or an ADD, the LSP hardware unit number of the new location of the station. |
| 9 | [47:16] |  | If a MOVE or a SUBTRACT, the NSP hardware unit number of the old location of the station. |
|  | [31:16] |  | If a MOVE or a SUBTRACT, the LSP hardware unit number of the old location of the station. |

### Explanation

Unlike the LSP EXCHANGE RESULT (Result Class = 8) messages and the SWAPLINE RESULT (Result Class = 10) messages, only the requesting MCS sees the MOVE/ADD/SUBTRACT STATION RESULT message (since it is the only MCS affected by this type of reconfiguration).

Receipt of this message with MSG[1].[47:08] = 0 signifies that the MOVE/ADD/SUBTRACT request was successfully completed. IfMSG[1].[47:08] is not zero, an error was discovered and the interpretation of this field is the same as that for DCWRITE error values.

# DLS UPDATE RESULT (Result Class = 12)

## Message Format

| Word | Field | Value | Description |
|---|---|---|---|
| 0 | [47:08] | 12 | Class. |
| | [00:01] | 0 | More Class = 12 messages are to follow. |
| | | 1 | The last or only Class = 12 message. |
| 6 | [47:48] | | The number of entries in the following words. |
| 7 to end | | | Station information, as follows: |
| | [37:14] | | LSN. |
| | [23:01] | 0 | The station has no line assignment. |
| | | 1 | The DLS number for a station with a line assignment. |
| | [22:23] | | The new DLS number (if [23:01] = 1). |

## Explanation

All MCSs (including the requesting MCS) affected by any reconfiguration request receive one or more DLS UPDATE RESULT messages that contain information pertinent to stations to which the MCS is currently attached and whose relative NSP number, line number, or station number is altered during processing of a reconfiguration request. The maximum number of entries for any given DLS UPDATE RESULT message is 128. If MSG[0].[00:01] = 1 for a given DLS UPDATE RESULT message, this value signifies that this message is the last (and, perhaps, only) message of its class for the current reconfiguration request. An MCS can note or discard at its discretion the information given in a DLS UPDATE RESULT message. However, if an MCS is DLS-number-oriented (for any of a variety of reasons), this information is of particular interest and should be noted.

# INTER-MCS COMMUNICATE RESULT (Result Class = 13)

### Message Format

| Word | Field | Value | Description |
|---|---|---|---|
| 0 | [47:08] | 13 | Class. |
|  | [23:24] |  | MCS number of the communicating MCS. |

### Explanation

Receipt of this message class in the primary queue of an MCS signifies the attempt of another MCS to communicate with the recipient MCS through an INTER-MCS COMMUNICATE (Type = 3) DCWRITE.

The interpretation of, and action taken in, receipt of this message type depends on mutually established conventions between the communicator and the recipient MCSs.

# STATION DETACHED (Result Class = 14)

## Message Format

| Word | Field | Value | Description |
|---|---|---|---|
| 0 | [47:08] | 14 | Class. |
|  | [39:16] | 0 | The station was detached because the NSP terminated. |
|  |  | 1 | The station was detached because of the action of the fully-participating MCS. |
|  |  | 2 | The station was detached as a result of a modification to the configuration made using the Interactive Datacomm Configurator (IDC). |
|  | [23:24] |  | LSN. |
| 1 | [23:24] |  | DLS number. |

## Explanation

Receipt of a STATION DETACHED result message signifies that the station is detached from the MCS. Further attempts to work with the station result in error values returned from calls on DCWRITE.

If word 0 [39:16] equals 0, the station is detached, because the NSP to which it was connected terminated. In this case, a STATION DETACHED result message is sent for each station on the NSP that is attached to an MCS.

If word 0 [39:16] equals 1, the station is detached by the fully-participating MCS. The fully-participating MCS is the MCS that transferred the station to the current owning MCS. If the former detaches the station or terminates, the current owning MCS receives a Class = 14 result message for the station.

If word 0 [39:16] equals 2, the station is detached because someone used the IDC to modify the running configuration. In this case, it is possible that the station is now under the control of a different MCS. This result is generated if an IDC modification causes the station to no longer have a valid DLS number, or if IDC is used to transfer the station to another MCS. Examples of IDC commands that may have these effects are SUBTRACT STATION, SUBTRACT LINE, MOVE STATION, MOVE LINE, MODIFY STATION, and MODIFY LINE. Refer to the *IDC Operations Guide* for descriptions of these commands.

# INTERROGATE STATION ENVIRONMENT RESULT (Result Class = 15)

### Message Format

| Word | Field | Value | Description |
|---|---|---|---|
| 0 | [47:08] | 15 | Class. |
| 0-5 | | | Interpretations applicable to the general message format. |
| 6 | [23:08] | | The size of this message (in words). |
| | [15:08] | | The number of entries in this message. |
| | [07:08] | | The index to first word of the first entry. |

### Explanation

The INTERROGATE STATION ENVIRONMENT RESULT message is returned in response to an INTERROGATE STATION ENVIRONMENT (Type = 4) DCWRITE request.

An MCS can issue an INTERROGATE STATION ENVIRONMENT request to inquire about the attributes of any station. The result indicates whether or not the station is a pseudostation. The response also indicates whether an MCS has requested full participation for the station.

Blanket interrogations can cause one or more Class = 15 message results to be generated; each message contains one or more entries. An entry can consist of one or more of the following types of information:

- Three or four words of station (logical) information from the operating system tables and eight words of NSP station table information
- Two words of terminal (physical) information
- Two words of line information from the operating system tables, and four optional words of NSP line table information
- One or more words of the station name
- One or more words of the terminal name
- An index word that provides pointers to the locations of each of the present information types and an index to index word for the next entry in the message (if any)

This particular message format was chosen so that when additional, desirable information becomes available, MCSs that wish to use the additional information can do so with minimal effort and MCSs that do not wish to use the new data are not forced into reprogramming.

## Explanation of Expanded Message Format

The expanded message format of the INTERROGATE STATION ENVIRONMENT RESULT message is used because of the variety and quantity of information that can be returned in this message.

This result message is an indexed structure. Figure 6–1 illustrates the path through index words to obtain additional station, line, or terminal information.

MSG[6] always contains (among other items) the index of the index word of the first entry. Each entry unconditionally has an index word that contains indexes for its entry to the requested information contained in the entry. The index word of an entry also contains the index of the word in the entry. In the special case in which the value of the index word (of the next index of the entry) is 0, no further entries beyond the current entry in that message exist.

For the remaining index fields within the index word of an entry that contain 0, the interpretation must be that the information was not originally requested and therefore is not supplied. (Refer to “INTERROGATE STATION ENVIRONMENT (DCWRITE Type = 4)” in the “DCWRITE Information” section to determine the method of requesting the various types of information.)

Word 6 of the INTERROGATE STATION ENVIRONMENT RESULT message contains the index to the first word of the first entry of the message. This first word is referenced as the index word, MSG[INX], in Figure 6–1 and in the message format. MSG[INX] holds the indexes to several first words of terminal, station, or line information. The bit fields of the words referenced by the indexes provide details of status and information requested.

The format of the message is illustrated in Figure 6–1. The format and meaning of each bit field is explained following the figure.

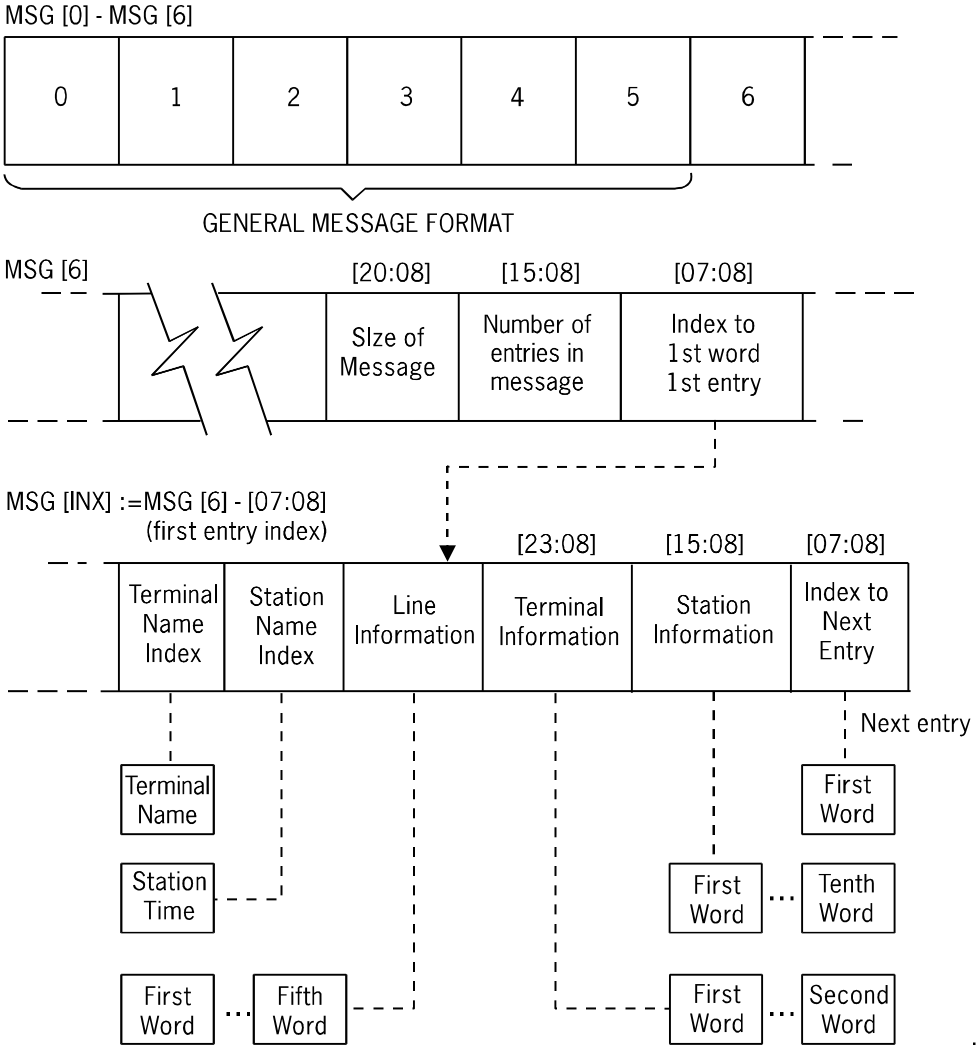


**Figure 6–1. INTERROGATE STATION ENVIRONMENT RESULT Index Diagram**

## INX := MSG[6].[07:08] (First Entry Index)

MSG[6].[07:08] has the following format:

| Word | Field | Description |
|---|---|---|
| MSG[INX] | [47:08] | The index to the terminal name, if not zero. |
| | [39:08] | The index to the station name, if not zero. |
| | [31:08] | The index to the first word of line information (if 0, no entry). |
| | [23:08] | The index to the first word of terminal information (if 0, no entry). |
| | [15:08] | The index to the first word of station information (if 0, no entry). |
| | [07:08] | The index to the first word (index word) of next entry (if 0, this is the last entry of the message). |

The format of this word remains the same for the first word (index word) of each subsequent entry, if any.

## MSG[MSG[INX].[47:08]]

If the terminal name is requested, MSG[MSG[INX].[47:08]] is represented in display form as six or fewer EBCDIC characters per word terminated by a period (for example, “TD830E.”).

## MSG[MSG[INX].[39:08]]

If the station name is requested, MSG[MSG[INX].[39:08]] is represented in display form as six or fewer EBCDIC characters per word terminated by a period (for example, “STATION/ONE.”).

## MSG[MSG[INX].[31:08]]

If the station has line assignment and line information is requested, MSG[MSG[INX].[31:08]] consists of a 2-word or 5-word entry that comprises several bit-fields. The interpretation varies depending on whether NSP line information was requested.

Figure 6–2 illustrates the format of the INTERROGATE STATION ENVIRONMENT RESULT message that is returned when NSP line table information was, and was not, requested.

| |
|---|
| Status Word |
| Word 0 of NSP Line Table |

| |
|---|
| Status Word |
| NSP Line Desriptor |
| Word 0 of NSP Line Table |
| Word 1 of NSP Line Table |
| Hardware Unit Number Word |

**Figure 6–2. Interpretation of MSG[MSG[INX].[31:08]]**

The words contained in MSG[MSG[INX].[31:08]] are explained in the following discussion.

### First Word: MSG[MSG[INX].[31:08]]

This is the status word and has the following format.

| Field | Value | Description |
|---|---|---|
| [46:01] | 1 | The line is NOT READY. |
| [45:01] | 1 | A line change is in progress (reconfiguration in progress). |
| [38:01] | 1 | A switched line error is encountered. |
| [36:01] | 1 | The line status is currently changing. |
| [35:01] | 1 | The line is connected. |
| [34:01] | 1 | Autoanswer is in force. |
| [33:01] | 1 | The line has an associated automatic calling unit (ACU); that is, it can be dialed out. |
| [32:01] | 1 | The line is a switched line. |
| [31:08] | | The current number of stations on this line. |
| [23:08] | | The maximum number of stations allowed for this line. |

### Second Word: MSG[MSG[INX].[31:08]+1] (NSP Line Information Not Requested)

If NSP line information is not requested, this is word 0 of the NSP line table and is the last word of the field. Word 0 of the NSP line table has the following format:

| Field | Description |
|---|---|
| [47:01] | The line is NOT READY. |
| [43:01] | The line is not connected. |
| [41:01] | Line TOG[1]. |
| [40:01] | Line TOG[0]. |
| [31:08] | The maximum number of stations on this line. |
| [23:08] | Line TALLY[1]. |
| [15:08] | Line TALLY[0]. |

### Second Word: MSG[MSG[INX].[31:08]+1] (NSP Line Information Requested)

If NSP line information is requested, this word is the NSP line descriptor and is followed by words 0 and 1 for the NSP line table and a word for the hardware unit number. The NSP line descriptor has the following format:

| Field | Description |
|---|---|
| [31:01] | The line is in a NOT READY PENDING state. |
| [30:01] | The line is in a SWITCHED ERROR state. |
| [28:01] | The line is busy in a switched request. |
| [27:01] | The line is connected. |
| [26:01] | AUTOANSWER is in force. |
| [25:01] | The line has an associated ACU. |
| [24:01] | The line is a switched line. |
| [21:01] | The line is invalid. |
| [20:01] | The line is a synchronous line. |

### Third Word: MSG[MSG[INX].[31:08]+2] (NSP Line Information Requested)

This is word 0 of the NSP line table and has the following format:

| Field | Description |
|---|---|
| [47:01] | The line is NOT READY. |
| [43:01] | The line is not connected. |
| [41:01] | Line TOG[1]. |
| [40:01] | Line TOG[0]. |
| [31:08] | The maximum number of stations on this line. |
| [23:08] | Line TALLY[1]. |
| [15:08] | Line TALLY[0]. |

### Fourth Word: MSG[MSG[INX].[31:08]+3] (NSP Line Information Requested)

This is word 1 of the NSP line table. It is not used.

### Fifth Word: MSG[MSG[INX].[31:08]+4] (NSP Line Information Requested)

This is the hardware unit number word and has the following format:

| Field | Value | Description |
|---|---|---|
| [47:16] | | The NSP hardware unit number if the station has a line assignment; otherwise, 0. |
| [31:16] | | The LSP hardware unit number if the station has a line assignment; otherwise, 0. |
| [15:16] | 0 | |

## MSG[MSG[INX].[23:08]]

If terminal information is requested, MSG[MSG[INX].[23:08]] consists of a 2-word entry comprising several bit-fields that are interpreted as follows:

### First Word: MSG[MSG[INX].[23:08]]

| Field | Value | Description |
|---|---|---|
| [43:01] | 1 | The terminal is a screen device. |
| [42:01] | 0 | This is a regular station or a schedule station. |
|  | 1 | This is a pseudostation. |
| [41:02] |  | MYUSE attribute (1 = IN, 2 = OUT, 3 = IO). |
| [31:16] |  | The terminal line width. |
| [15:16] |  | The terminal buffer size (MAXINPUT value). |

### Second Word: MSG[MSG[INX].[23:08]+1]

| Field | Description |
|---|---|
| [47:16] | Terminal PAGESIZE value. |
| [31:16] | Terminal MAXOUTPUT value. |

## MSG[MSG[INX].[15:08]]

If station information is requested, MSG[MSG[INX].[15:08]] is an entry of at least four words comprising several bit-fields that are interpreted as follows:

### First Word: MSG[MSG[INX].[15:08]]

| Field | Value | Description |
|---|---|---|
| [47:01] | 1 | Station LOGIN=TRUE. |
| [46:01] | 1 | Station WRAPAROUND=TRUE. |
| [45:01] | 1 | Station SPO=TRUE. |
| [44:01] | 1 | Station is ENABLED. |
| [43:01] | 1 | Station is NOT READY. |
| [42:01] | 1 | Station is ATTACHED. |
| [41:01] | 1 | Station change is in progress. |
| [40:01] | 1 | Station has been prepared for detach. |
| [39:01] | 0 | No MCS has requested full participation for this station. |
| | 1 | An MCS has requested full participation for this station. |
| [38:01] | 1 | The station is capable of automatic sequence mode. |
| [37:08] | | The MCS number of the controlling MCS. |
| [29:14] | | The LSN. If the LSN is a pseudostation and its value is greater than 16383, then the value of this field is zero. In this case, the LSN can be found in MSG[MSG[INX] . [15:08] +1] . [[23:24]. |
| [15:16] | | The line width of the station. |

### Second Word: MSG[MSG[INX].[15:08]+1]

| Field | Value | Description |
|---|---|---|
| [47:08] | | The control character of the station (EBCDIC). |
| [39:01] | 1 | The MCS participates in object I/O. |
| [38:01] | 1 | Station event messages are sent to the current queue. |
| [37:01] | 1 | Station NOT READY results are sent to the current queue. |
| [36:01] | 1 | Messages beginning with the control character of the station are sent to the current queue. |
| [35:01] | 1 | Errors are sent to current queue. |
| [34:01] | 1 | The MCS wants all results. |
| [33:01] | 1 | The station is transferred to another MCS. |
| [32:01] | 1 | Schedule station. |
| [31:08] | | The RETRY count as specified in the DATACOMINFO file. |
| [23:24] | | The DLS number of the station. If [23:01] = 0, the station has no current line assignment. If the station is a pseudostation and the LSN is greater than 16393, this field contains the LSN number of that station. |

### Third Word: MSG[MSG[INX].[15:08]+2] (NSP Information Not Requested)

If NSP station information is not requested, the third word contains the receive and transmit address characters. The format of this word is as follows:

| Field | Description |
|---|---|
| [47:24] | Receive address characters. |
| [23:24] | Transmit address characters. |

### Third Word: MSG[MSG[INX].[15:08]+2] (NSP Information Requested)

If NSP station information is requested, the third word contains the NSP station descriptor. The format of this word is as follows:

| Field | Description |
|---|---|
| [47:08] | Station priority. |
| [39:01] | Station acknowledge. |
| [31:08] | Station TALLY. |
| [23:01] | Station is queued. |
| [22:01] | Station is enabled. |
| [21:01] | Station is not ready. |
| [20:01] | Station is invalid. |

### Fourth Word: MSG[MSG[INX].[15:08]+3]

If NSP station information is not requested, the fourth word is reserved.

If NSP station information is requested, the fourth word contains the first word of the NSP station table. The format of this word is as follows:

| Field | Description |
|---|---|
| [47:08] | Station control character. |
| [39:08] | Station end of text character. |
| [31:08] | Station backspace character. |
| [23:08] | Station line delete character. |
| [15:08] | Station WRU (who are you) character. |
| [07:08] | Station application number. |

### Fifth Word: MSG[MSG[INX].[15:08]+4]

This word is reserved.

### Sixth Word: MSG[MSG[INX].[15:08]+5]

This word is reserved.

### Seventh Word: MSG[MSG[INX].[15:08]+6]

| Field | Description |
|---|---|
| [47:48] | Transmit or receive transmission number characters. |

### Eighth Word: MSG[MSG[INX].[15:08]+7]

| Field | Description |
|---|---|
| [47:24] | Receive station address characters. |
| [23:24] | Transmit station address characters. |

### Ninth Word: MSG[MSG[INX].[15:08]+8]

This word is reserved.

### Tenth Word: MSG[MSG[INX].[15:08]+9]

This word is reserved.

### Eleventh Word

This word is reserved.

# TRANSFER STATION CONTROL RESULT (Result Class = 16)

## Message Format

| Word | Field | Value | Description |
|---|---|---|---|
| 0 | [47:08] | 16 | Class. |
| | [26:01] | | BNA transfer bit. |
| | | 0 | The station is transferred to the designated MCS. |
| | | 1 | Word 6.[47:12] supplies the index to the Station Transfer Index Control Word (INX = MSG[6].[47:12]). The host name and MCS name are indexed by MSG[INX].[11:12] and MSG[INX].[23:12] respectively.<br>The station is transferred to the BNA station transfer MCS, SYSTEM/STATION/TRANSFER.<br>The host name must be supplied, and the MCS name is optional. A 0 in the MCS name index field indicates that no MCS name was supplied. |
| | [25:02] | 0 | This is the result of a Transfer Station Control (Type = 45) DCWRITE. |
| | | 1-3 | This is the result of a Station Detach (Type = 42) DCWRITE. |
| | [23:24] | | The LSN of the station that has been transferred. |
| 1 | [23:24] | | The MCS number of the MCS that performed the transfer. |
| 4 | [47:24] | | The ODT unit number, if the variant field of the original DCWRITE type (MSG[4].[23:24]) indicates that a pseudostation that represents one of the system ODT units is being transferred by the COMS/ODT/DRIVER. |
| 6 | [47:12] | | Index to the Station Transfer Index Control Word. |
| | [32:09] | | The index to the information to be passed by the Transfer Station Control (Type = 45) DCWRITE, only if Word[0].[30:01] is set in DCWRITE. |
| | [23:24] | | Indexes to station interrogate information. |

### Explanation

The TRANSFER STATION CONTROL RESULT message is received by an MCS in its primary queue when another MCS uses the TRANSFER STATION CONTROL (Type = 45) DCWRITE request to give it control over a station.

The MCS that receives this message can check it to see if the transferred station is a pseudostation and if the MCS that sent the message has requested full participation for the station. MSG[1].[23:24] contains the MCS number of the MCS that transferred control. Whenever desired, the MCS that receives control of the station can return control back to the MCS that transferred the station.

This message can be used to pass session information from a controlling MCS to another MCS acting as a subordinate application. The subordinate MCS can use the session information passed by the controlling MCS rather than requesting it from the user.

If the station is a pseudostation that has been retracted by means of the Station Detach DCWRITE (that is, the field [25:02] of word 0 has a non-zero value), this message indicates to the receiving MCS that the retraction is complete.

Refer to the *Transaction Server for ClearPath MCP Configuration Guide* for more information.

The following are the contents and formats of the bit fields in the expanded format of this result.

## INX := MSG[6].[47:12] (Index to the Station Transfer Index Control Word)

| Word | Field | Description |
|---|---|---|
| MSG[INX] | [23:12] | Index to MCS name. |
| | | @[23:12].[47:16] - Length of MCS name in bytes. |
| | | @[23.12].[32:08] - First character of MCS name. |
| | [11:12] | Index to host name. |
| | | @[11:12].[47:16] - Length of host name in bytes. |
| | | @[11:12].[32:08] - First character of host name. |

## INX := MSG[6].[32:09] (Header Word of Information Area)

| Word | Field | Description |
|---|---|---|
| MSG[INX] | [15:16] | The number of bytes of MCS information passed from the Transfer Station Control (Type = 45) DCWRITE. |

## INX := MSG[6].[32:09] + 1 (First Word of Information Area)

| Word | Field | Value | Description |
|---|---|---|---|
| MSG[INX] | [47:24] | | Reserved. |
| | [23:24] | | The LSN of the real station that the controlling MCS associates with the pseudostation being transferred. |
| MSG[INX+1] | [47:32] | | To be used for the identification of the controlling MCS. |
| | [15:16] | 1 | The format version number. |
| MSG[INX+2] | [47:37] | | Reserved. |
| | [10:01] | 0 | No authentication information is passed. |
| | | 1 | Authentication information is passed. |
| | [09:01] | 0 | No open text information is passed. |
| | | 1 | The open text information is passed. |
| | [08:01] | 0 | No single window is requested. |
| | | 1 | The single window is requested. |
| | [07:01] | 0 | No window information is passed. |
| | | 1 | The window information is passed. |
| | [06:01] | 0 | No chargecode information is passed. |
| | | 1 | The chargecode information is passed. |
| | [05:01] | 0 | No accesscode information is passed. |
| | | 1 | Accesscode information is passed. |
| | [04:01] | 0 | No system user information is passed. |
| | | 1 | System user information is passed. |
| | [03:01] | 0 | Privileged access is not allowed by the controlling MCS (valid if word [INX+2] field [02:01] equals 1). |
| | | 1 | Privileged access is allowed by the controlling MCS (valid if word [INX+2] field [02:01] equals 1). |
| | [02:01] | 0 | No privileged information is passed. |
| | | 1 | The privileged access capability is passed by the controlling MCS in word [INX+2] field [03:01]. |
| | [01:01] | 0 | No usercode is passed. |
| | [01:01] | 1 | The usercode of the controlling MCS session is passed. |
| | [00:01] | 0 | The controlling MCS session is not control capable. |

| Word | Field | Value | Description |
|---|---|---|---|
| | [00:01] | 1 | The controlling MCS session is control capable. |
| MSG[INX+3] | | | Reserved. |
| MSG[INX+4] | [47:16] | | Reserved. |
| | [31:16] | | The relative index in MSG of the authentication information (valid if word [INX+2] field [10:01] equals 1). |
| | [15:16] | | The relative index in MSG of usercode (valid if word [INX+2] field [01:01] equals 1). |
| MSG[INX+5] | [47:16] | | The relative index in MSG of the accesscode information (valid only if Accesscode Information Passed flag is set). |
| | [31:16] | | The relative index in MSG of the chargecode information (valid only if Chargecode Information Passed flag is set). |
| | [15:16] | | The relative index in MSG of the window information (valid only if Window Information Passed flag is set). |
| MSG[INX+6] | [47:32] | | Reserved. |
| | [15:16] | | The relative index in MSG of the open notification text (valid only if Open Text Passed flag is set). |

## UINX := INX+MSG[INX+4].[31:16] (First Word of Authentication Location)

| Word | Field | Description |
|---|---|---|
| MSG[AUTHINX] | [47:08] | The type of authentication. (A value of 1 indicates GSS Authentication.) |
| | [39:08] | The length of the authentication information (in bytes). |
| | [31:08] | The first byte of the authentication information. |

## UINX := INX+MSG[INX+4].[15:16] (First Word of Usercode Location)

| Word | Field | Description |
|---|---|---|
| MSG[UINX] | [47:08] | The length of the usercode (in bytes). |
| | [39:08] | The first byte of the usercode. |

## AINX := INX+MSG[INX+5].[47:16] (First Word of Accesscode Location)

| Word | Field | Description |
|---|---|---|
| MSG[AINX] | [47:08] | The length of the accesscode (in bytes). |
| | [39:08] | The first byte of the accesscode. |

## CINX := INX+MSG[INX+5].[31:16] (First Word of Chargecode Location)

| Word | Field | Description |
|---|---|---|
| MSG[CINX] | [47:08] | The length of the chargecode (in bytes). |
| | [39:08] | The first byte of the chargecode. |

## WINX := INX+MSG[INX+5].[15:16] (First Word of Window Information Location)

| Word | Field | Description |
|---|---|---|
| MSG[WINX] | [47:08] | The window dialog number. |
| | [39:08] | The first byte of the usercode. |
| | [31:08] | The first byte of the window name. |

## OINX := INX+MSG[INX+6].[15:16] (First Word of Open Text Location)

| Word | Field | Description |
|---|---|---|
| MSG[OINX] | [47:08] | The length of the open text (in bytes). |
| | [39:08] | The first byte of the open text. |

The remainder of the message is formatted as in the INTERROGATE STATION ENVIRONMENT RESULT (Result Class = 15) message for which station, terminal, line, and station name are given.

Additional information about the station that was transferred is included in the same message. The format of the information is identical to that of the INTERROGATE STATION ENVIRONMENT RESULT (Result Class = 15) message. The types of information supplied are station name, station (logical) information, terminal (physical) information, and line information.

The recipient MCS is responsible for all error recovery for the station from the time of the transfer one of the following actions occurs:

- The MCS transfers control of the station to some other MCS.
- A halt/load sequence is initiated.
- A complete shutdown of data comm is performed.

At the time of transfer, all subsequent message traffic associated with the transferred station appears in the primary queue of the recipient MCS. If necessary, the CHANGE CURRENT QUEUE (Type = 32) DCWRITE function can be invoked by the recipient MCS to establish a different current queue or to establish a different set of option bits (for example, MCS wants all results and errors sent to the current queue).

# ODT-TO-MCS RESULT (Result Class = 17)

### Message Format

| Word | Field | Value | Description |
|---|---|---|---|
| 0 | [47:08] | 17 | Class. |
| 2 | [39:16] | | Number of characters of text. |
| 6 to end | | | Text. |

### Explanation

Receipt of an ODT-TO-MCS RESULT message (in the primary queue of the MCS) is the direct result of the input of an SM (Send to MCS) system message from the ODT. The text appears exactly as it was presented following the colon in the SM system message. The interpretation of, and action taken on, this message class by an MCS is solely the responsibility of the receiving MCS.

# ODT-TO-STATION RESULT (Result Class = 18)

## Message Format

| Word | Field | Value | Description |
|---|---|---|---|
| 0 | [47:08] | 18 | Class. |
|  | [23:24] |  | LSN. |
| 2 | [39:16] |  | Number of characters of text. |
| 6 to end |  |  | Text. |

## Explanation

Receipt of an ODT-TO-STATION RESULT message is the direct result of the input of an SS (Send to Station) system message from the ODT. The message text appears exactly as it was presented following the colon in the SS message. Action taken on, and interpretation of, this message class is the responsibility of the receiving MCS. The CONTROLLER inserts this result in the current queue of the station.

# UPDATE LINE ATTRIBUTES RESULT (Result Class = 19)

### Message Format

| Word | Field | Value | Description |
|---|---|---|---|
| 0 | [47:08] | 19 | Class. |
| 1 | [47:08] | | The error value (corresponds to the DCWRITE result). |

### Explanation

The UPDATE LINE ATTRIBUTES RESULT message is the result of the UPDATE LINE ATTRIBUTES (Type = 131) DCWRITE. Except as specified, the format of this message is the same as the message supplied in the UPDATE LINE ATTRIBUTES (Type = 131) DCWRITE that caused this result message.

# MESSAGE FROM CONTROLLER RESULT (Result Class = 21)

## Message Format

| Word | Field | Value | Description |
|---|---|---|---|
| 0 | [47:08] | 21 | Class. |
| | [39:16] | | Variant field, as follows: |
| | [39:08] | | Message types, as listed below: |
| | | 1 | WFL card (outgoing only); refer to the CONTROLCARD function (WFL Card Image). |
| | | 2 | CONTROLLER command (outgoing only); refer to the SETUPINTERCOM function (Operator Request). |
| | | 3 | Reply to CONTROLLER keyin (ODT). Refer to the SETUPINTERCOM function (Operator Request). |
| | | 4 | Next message. Refer to the SETUPINTERCOM function (Operator Request). |
| | | 5 | Translated MESSER messages. Used only when the MCS is a nonexpanded capacity MCS, where variant MSG[0] . [24:01] is not set on Initialize Primary Queue. |
| | | 6 | Backup notice or job queue insertion notice. |
| | | 7 | EOT/EOJ notice. |
| | | 8 | BOT/BOJ notice. Used only when the MCS is a nonexpanded capacity MCS, where variant MSG[0] . [24:01] is not set on Initialize Primary Queue. |
| | | 9 | The job or task is scheduled. Used only when the MCS is a nonexpanded capacity MCS, where variant MSG[0] . [24:01] is not set on Initialize Primary Queue. |
| | | 10 | The job or task is awakened. Used only when the MCS is a non-expanded capacity MCS, where variant MSG[0] . [24:01] was not set on Initialize Primary Queue. |
| | | 11 | The system security options have changed. Word [1] contains the current value of the MCP Security Option Word. |
| | | 25 | Security MESSER messages. |
| | | 250 | For an ODT-simulating MCS: a message from the CONTROLLER to indicate a message that is displayed under the command C. The format of the message depends on the operating system. |

| Word | Field | Value | Description |
|---|---|---|---|
| | | 251 | For an ODT-simulating MCS: a message from the CONTROLLER to indicate a message that is displayed under the command MSG. The format of the message depends on the operating system. |
| | | 252 | For an ODT-simulating MCS: a message from the CONTROLLER to indicate that the CONTROLLER has too many dialogues waiting for an answer. |
| | | 253 | Reply to an operator request from an ODT-simulating MCS. The format of the message is the same as that of Type 3. |
| | | 254 | Next message from an ODT-simulating MCS. The format of the message is the same as that of Type 4. |
| | | 255 | For an ODT-simulating MCS: a message from the CONTROLLER to indicate that all the lines of the request have been sent. |
| | [31:08] | | Information about message types in MSG[0].[39:08]. |
| | | 1 | Type 6 contains the job queue insertion notice. |
| | | 2 | Type 6 contains the job backup notice. |
| | | 0-255 | Type 3 contains the value inserted and defined by MCS. Refer to the SETUPINTERCOM function. |
| | | N | Types 4, 5, 7, 8, 9, and 10 contain the MCS number. |
| | [23:24] | | Types 5, 9, and 10 contain the LSN of remote terminal. |
| | | | Types 250, 251, 252, 253, 254, and 255 contain the dialogue number. |
| | [23:09] | | Type 3 is used by the sending MCS. |
| | | 0 | Types 6, 7, and 8 are not used. |
| | [14:15] | | Types 3, 6, 7, and 8 contain the LSN of the remote terminal. |
| 1 | | | Information about message types in MSG[0].[39:08]: |
| | | | Type 3 contains the text length in characters. |
| | | | Type 4 contains the terminal information word. |
| | | | Type 5 contains the job serial word. |
| | | | Type 6 contains the job number. |
| | | | Type 7. If MSG[1].[47:01] = 1, the usercode follows the message plus 1 word.<br>If MSG[1].[39:01] = 1, the EOT is for a remote task.<br>MSG[1].[11:12] is the text length in characters. |

| Word | Field | Value | Description |
|---|---|---|---|
|  |  |  | Types 8, 9, and 10 contain the job serial word. |
|  |  |  | Type 11 contains the current value of the MCP's Security Option Word. |
|  |  |  | Type 252 : MSG[1].[47:16] contains the index of the first waiting entry. MSG[1].[46:07] contains the number of entries. |
| 2 |  |  | Information about message types in MSG[0].[39:08]: |
|  |  |  | Types 3 and 7 contain text through MSG[N]. |
|  |  |  | Type 4 contains the contents set up by the operating system (through MSG[N]). |
|  |  |  | Type 5: Contains the MCP message number |
|  |  |  | Type 6 contains the queue number (job queue insertion notice. |
|  |  |  | Type 8 contains the priority of the job. |
|  |  |  | Types 9 and 10 contain the usercode (through MSG[4]). If MSG[2].[47:01] = 1, the length of the simple form usercode is in MSG[2].[46:07]. |
|  |  | 1 | Type 6 contains the printer backup (backup notice). |
|  |  | 2 | Type 6 contains the punch backup (backup notice). |
| 3 |  |  | Type 5 contains a message code that indicates the type of MCP message. |
|  | [15:16] |  | SUMLOG minor type, which indicates the type of message, FATAL, RSVP, etc. For details, see the *System Log Programming Reference Manual*. |
|  | [27:12] |  | SUMLOG major type 3, which indicates that this is a display message. |
|  |  |  | Type 6 contains the simple form usercode (through MSG[5]), and MSG[3].[47:08] contains the length of the name. This entry appears only if this result has been returned by Print Job File -- that is, by a WFL job, and word [0].[31:8] is equal to 2. |
| 4 |  |  | Information about the message types in MSG[0].[39:08]: |
|  |  |  | Type 5 contains the text length in characters. |
|  |  |  | Type 8 contains the simple form usercode (through MSG[6]); MSG[4].[47:08] contains the length of the name. |
| 5 |  |  | Information about the message types in MSG[0].[39:08]: |
|  |  |  | Type 5 contains the text (through MSG[N]). |
| 6 |  |  |  |

| Word | Field | Value | Description |
|---|---|---|---|
| 7 | | | Information about the message types in MSG[0].[39:08]:<br>Type 8 contains the simple form compiler name (through MSG[9]); MSG[7].[47:08] contains the length of the name. |
| 10 | | | Information about the message types in MSG[0].[39:08]:<br>Type 8 contains the start of the standard form task name (through MSG[N]). |

Subtypes 14, 17, 18, and 19 are used when the MCS is registered as an expanded capacity MCS, where variant MSG[0].[24:1] is set on Initialize Primary Queue. These subtypes correspond to the following previously used types:

- Subtype 14 = Type 5
- Subtype 17 = Type 8
- Subtype 18 = Type 9
- Subtype 19 = Type 10

## Class 21, Subtype 14

| Word | Field | Value | Description |
|---|---|---|---|
| 0 | [47:08] | 21 | Class |
| | [39:08] | 14 | Subtype |
| | [31:08] | | MCS number |
| | [23:24] | | LSN |
| 1 | [39:40] | | Stack number |
| 2 | [39:40] | | Job number |
| 3 | [39:40] | | Task number |
| 4 | | | <Reserved> |
| 5 | | | MCP message number |
| 6 | | | <Reserved> |
| 7 | | | Message code that indicates the type of MCP message. |
| 8 | [47:01] | | If 1, this indicates that a usercode exists. |
| | [31:16] | | If [47:01] is 1, this contains the offset of the usercode from word 0. |
| | [15:16] | | Message length in characters |
| 9 - n | | | Variable length information, with the message text beginning at word 9 and followed by the usercode, if it exists. |

## Class 21, Subtype 17

| Word | Field | Value | Description |
|---|---|---|---|
| 0 | [47:08] | 21 | Class |
|  | [39:08] | 17 | Subtype |
|  | [31:08] |  | MCS number |
|  | [23:24] |  | LSN |
| 1 |  |  | <Reserved> |
| 2 |  |  | <Reserved> |
| 3 |  |  | <Reserved> |
| 4 | [39:40] |  | Stack number |
| 5 | [39:40] |  | Job number |
| 6 | [39:40] |  | Task number |
| 7 |  |  | <Reserved> |
| 8 |  |  | Contains the priority of the job. |
| 9 |  |  | <Reserved> |
| 10 |  |  | <Reserved> |
| 11 |  |  | <Reserved> |
| 12 |  |  | Contains the word offset from word 0 of the simple form usercode. If the field [47:8] of the word pointed to is nonzero, it contains the length of the usercode. |
| 13 |  |  | Contains the word offset from word 0 of the standard form compiler name. If field [47:8] of the word pointed to is nonzero, it contains the length of the compiler name. |
| 14 |  |  | <Reserved> |
| 15 |  |  | <Reserved> |
| 16 |  |  | <Reserved> |
| 17 |  |  | <Reserved> |
| 18 |  |  | <Reserved> |
| 19 |  |  | <Reserved> |
| 20 |  |  | Contains the word offset from word 0 of the standard form task name. |
| n - end |  |  | Variable length information that the above offsets point to. |

## Class 21, Subtypes 18 and 19

| Word | Field | Value | Description |
|---|---|---|---|
| 0 | [47:08] | 21 | Class |
| | [39:08] | 18 or 19 | Subtypes |
| | [31:08] | | MCS number |
| | [23:24] | | LSN |
| 1 | [39:40] | | Stack number |
| 2 | [39:40] | | Job number |
| 3 | [39:40] | | Task number |
| 4 | | | <Reserved> |
| 5 | [15:16] | | If a usercode exists, contains the offset from word 0 of the usercode. |
| n | | | Simple form usercode |
| | [47:01] | | If 1, length of usercode is in field [46:07] |
| | [46:07] | | Length of simple form usercode. |

### Explanation

This message is received by or sent to the queue designated using the SETUPINTERCOM function. The types of messages received are determined by the functions performed by the MCS, such as initiating tasks that cause task activity.

# LINE INTERROGATE RESULT (Result Class = 24)

### Message Format

| Word | Field | Value | Description |
|---|---|---|---|
| 0 | [47:08] | 24 | Class. |
| | [23:24] | | LSN. |
| 7 | | | DLS number. |
| 8 | | | Line descriptor and LSP information (primary line). |
| | [30:07] | | Switched status. |
| | [23:01] | 1 | LOSSOFCARRIER = DISCONNECT option. |
| | [21:01] | 1 | Line is invalid. |
| | [20:01] | 0 | Asynchronous line. |
| | | 1 | Synchronous line. |
| 9 | | | First control word for line. |
| 10 | | | First line table word. |
| | [47:06] | | Line status. This field is the same as MSG[1].[39:06] in an error result message. |
| | [40:01] | | Value of LINE(TOG[0]). |
| | [31:08] | | Value of MAXSTATIONS. |
| | [23:08] | | Value of LINE(TALLY[1]). |
| | [15:08] | | Value of LINE(TALLY[0]). |
| 11 | | | Station descriptor for the station. |
| | [47:08] | | Value of STATION(FREQUENCY). |
| | [22:01] | | Value of STATION(ENABLED). |
| | [21:01] | 0 | Station is READY. |
| | [20:01] | 0 | Station is valid. |

### Explanation

The LINE INTERROGATE RESULT returns information about the line and the station specified in the original LINE INTERROGATE (Type = 104) DCWRITE.

# OBJECT JOB INPUT REQUEST RESULT (Result Class = 25)

### Message Format

| Word | Field | Value | Description |
|---|---|---|---|
| 0 | [47:08] | 25 | Class. |
| | [23:24] | | LSN of the schedule station. |
| 4 | [47:24] | | FRSN. |

### Explanation

When an object job requests input from a schedule station (by a READ) and no input is currently queued for the station, an OBJECT JOB INPUT result message is constructed and sent to the current queue of the controlling MCS.

# INTERCEPTED MESSAGE RESULT (Result Class = 29)

### Message Format

The minimum length of this result is six words.

| Word | Field | Value | Description |
|---|---|---|---|
| 0 | [47:08] | 29 | Class. |
| | | | The following description of the variant field (the next 16 bits) applies only if the originating DCWRITE is a WRITE (Type = 33); otherwise, the header is identical to the original DCWRITE request message. |
| | [39:16] | | Carriage control fields. |
| | [39:08] | | The channel number to skip to or the number of lines to skip (NDLII SKIPCOUNT). |
| | [31:01] | 1 | The tabulation to be done (NDLII TAB). |
| | [30:01] | 1 | Carriage control should be done before text is transmitted (NDLII MOTIONBEFORE). |
| | [29:01] | 1 | More blocks to follow this one (NDLII BLOCKED). |
| | [28:01] | 1 | The value stored in MSG[0].[39:08] is the number of vertical lines that should be skipped (NDLII SPACE). |
| | [27:01] | 1 | The value stored in MSG[0].[39:08] is the channel number to skip to (NDLII SKIPLINE). |
| | [26:01] | 1 | A new page is required for the output device (NDLII NEWPAGE). |
| | [25:01] | 1 | Carriage return is suppressed (NDLII NOCARRIAGERETURN). |
| | [24:01] | 1 | Line feed is suppressed (NDLII NOLINEFEED). |
| | [23:24] | | LSN or DLS number. |
| 1 | [47:08] | | Priority of output. |
| 2 | [39:16] | | Text size field. |
| 4 | [47:24] | | The message number field or the FRSN. |
| | [23:24] | | Original DCWRITE Type Field: contains MSG[0].[47:24] from the original message. |
| 6 to end | | | Text (if any) from the original DCWRITE request. If text is to be transmitted to the station, a byte count must be given in the text size field MSG[2].[39:16]. |

### Explanation

The INTERCEPTED MESSAGE RESULT is received by an MCS, in its primary queue, when it has requested full participation for a station that it transferred to another MCS.

The fully participating MCS receives an INTERCEPTED MESSAGE RESULT in its primary queue every time the MCS that controls the station performs certain DCWRITE requests on that station. The data comm subsystem intercepts each request, converts it to the INTERCEPTED MESSAGE RESULT format, and places the result message in the primary queue of the fully participating MCS. The fully participating MCS can reconstruct the original DCWRITE request from the information in the result.

An INTERCEPTED MESSAGE RESULT message informs an MCS that the operating system has intercepted an attempt by another MCS or program to send output to a station it controls as a pseudo-MCS. Refer to the description of station control transfer under “TRANSFER STATION CONTROL (DCWRITE Type = 45)” in the “DCWRITE Information” section.

# NSPINITIALIZED RESULT (Result Class = 30)

### Message Format

| Word | Field | Value | Description |
|---|---|---|---|
| 0 | [47:08] | 30 | Class. |
| | [23:01] | 1 | DLSNOTLSN. |
| | [22:07] | | NSP number. |

### Explanation

The NSPINITIALIZED RESULT indicates when an NSP has completed initialization and is ready for use. One copy of this message is issued to each MCS that has initialized its primary queue at the time the initialization takes place.

## STATION REINITIALIZED (Result Class = 31)

### Message Format

| Word | Field | Value | Description |
|---|---|---|---|
| 0 | [47:08] | 31 | Class. |
| | [24:01] | 1 | The line was also reinitialized. |
| | | 0 | Only the station was reinitialized. |
| | [23:24] | | LSN. |
| 1 | [23:24] | | DLS number. |

### Explanation

Receipt of a STATION REINITIALIZED result indicates that the station is deleted from the NSP and then added again. This process causes all NSP station variables to be reinitialized. In addition, the values of any of the station attributes or station terminal attributes can be modified. If bit 24 of word 1 is turned on, the NSP line variables have also been reinitialized for the line to which the station is attached. In addition, any of the line attributes can be altered.

This result is issued when a line or station in the configuration is modified with the IDC. For information about IDC, refer to the *IDC Operations Guide*.

# POWER OFF PENDING RESULT (Result Class = 32)

### Message Format

| Word | Field | Value | Description |
|---|---|---|---|
| 0 | [47:08] | 32 | Class. |
| 6 | [47:24] | | Pointer to text. |
| | [23:24] | | Number of characters of text. |
| 7 | | | Number of minutes until power off. |
| 8 | | | Type of message, as follows: |
| | | 1 | Scheduled power off. |
| | | 2 | Unscheduled due to thermal overload. |
| | | 3 | Thermal overload warning. |
| | | 4 | Unscheduled power off (request for immediate power off). |
| | | 5 | Canceled power off. |

Word MSG[MSG[6].[47:24]] to end is text.

### Explanation

A POWER OFF PENDING RESULT (Result Class = 32) message (in the primary message queue of the MCS) is a notification of a pending power-off operation, cancellation of a pending power off, or warning of a possible power-off operation. The text contains the information regarding the type of power-off message, such as the time left in minutes or the type of warning. The handling of this message class by an MCS is solely the responsibility of the receiving MCS.

# ODT MODE SWITCH NOTICE RESULT (Result Class = 80)

### Message Format

| Word | Field | Value | Description |
|---|---|---|---|
| 0 | [47:08] | 80 | Class. |
|  | [39:16] |  | Variant field, as follows: |
|  | [24:01] | 0 | The ODT has been switched to communicate with the CONTROLLER. |
|  |  | 1 | The ODT has been switched to communicate through the COMS/ODT/DRIVER. |
|  | [23:24] |  | The ODT unit number. |
| 3 | [47:24] |  | The timestamp. |
| 6 |  | 2 | The ODT is a TD804. |
|  |  | 6 | The ODT is a TD831. |
| 7 |  |  | MCS number to which the transfer is desired. |

### Explanation

An ODT MODE SWITCH NOTICE RESULT message is placed in the INTERCOMQUEUE of the COMS/ODT/DRIVER when someone enters either *??MARC* or *??ODT* at an ODT. This message is formatted by the operating system. MSG[6] and MSG[7] are valid only if MSG[0].[24:01] = 1.

# INPUT FROM AN ODT RESULT (Result Class = 81)

### Message Format

| Word | Field | Value | Description |
|---|---|---|---|
| 0 | [47:08] | 81 | Class. |
|  | [23:24] |  | ODT unit number. |
| 2 | [39:16] |  | Number of characters of text. |
| 3 | [47:24] |  | Timestamp. |
| 6 to end |  |  | Text (if MSG[2].[39:16] > 0). |

### Explanation

An INPUT FROM AN ODT RESULT message is placed in the INTERCOMQUEUE of the COMS/ODT/DRIVER when input is entered at an ODT that has been placed under the jurisdiction of the COMS/ODT/DRIVER by an ODT switch primitive.

# ERROR RESULT (Result Class = 99)

The ERROR RESULT message is used by the data comm subsystem to report errors on lines and stations over which the MCS has control. The ERROR RESULT message is recognized by the value 99 in the class field.

The class message has two formats that are described in the information that follows.

- The line/station format
- The switched status format

## Line/Station Format of ERROR RESULT Message

### Message Format

| Word | Field | Value | Description |
|---|---|---|---|
| 0 | [47:08] | 99 | Class. |
| | [23:24] | | LSN. |
| 1 | [47:08] | | Result byte index. |
| | [39:06] | | Line status prior to abort. |
| | [33:01] | | NDLII LINE (TOG [1]). |
| | [32:01] | | NDLII LINE (TOG [0]). |
| | [31:08] | | Last flag set in MSG[1].[23:24]. Refer to the "General Result Message Format" earlier in this section. |
| | [23:24] | | Error flag field. Refer to the "General Result Message Format" earlier in this section. |
| 4 | [23:24] | | Original DCWRITE type (the original contents of MSG[0].[47:24] prior to presentation of the message to the NSP). |

### Explanation

This format is used to return error results that occur during data transmission. The RESULT BYTE INDEX (MSG[1].[47:08]) can have the value 1, 2, 3, 6, or 7. This format can be returned on an error on input from a terminal.

### Error Results in Line/Station Format

Table 6–4 lists the error results returned in the line/station format of the ERROR RESULT message.

Note that the general result message format also has an error flag field.

### Table 6–4. Error Result, Line/Station Format

| ResultField | Value |
|---|---|
| MSG[0].[47:08] | 99 |
| MSG[0].[39:16] | 0 |
| MSG[0].[39:08] | 0 |
| MSG[0].[31:08] | 0 |
| MSG[0].[23:24] | LSN |
| MSG[1].[47:08] | Result byte index |
| MSG[1].[39:08] | 0 |
| MSG[1].[39:06] | 0 |
| MSG[1].[33:01] | 0 |
| MSG[1].[32:01] | 0 |
| MSG[1].[31:08] | First flag set (from FIRSTERROR) |
| MSG[1].[23:24] | Error flag field (from NSP errors) |
| MSG[4].[23:24] | 35 (ENABLE INPUT) |

## Result Byte Index

Table 6–5 lists the values and definitions of the result byte index field MSG[1].[47:08] in the line/station format of the ERROR RESULT message.

### Table 6–5. Result Byte Index Values

| Value | Description |
|---|---|
| 1 | An error was detected; the NDLII algorithm performed a SENDHOST ERROR statement. |
| 2 | A no-label error occurred; an error condition for which no provision was made was encountered while executing NDLII-written code. |
| 3 | This request was not applicable to the station; the NSP was requested to perform an operation on a station or line that makes no sense for the station or line (for example, a disconnect request for a direct-connect line). |
| 6 | An adapter fault caused an abort; an adapter fault was encountered by an NSP. |
| 9 | The station does not support this option. This result is applicable to the SET TRANSMISSION NUMBER (Type = 40) and SET/RESET SEQUENCE MODE (Type = 49) DCWRITE types. |
| 10 | The line structure was not present, which causes the request to be rejected. |
| 11 | The station structure was not present, which causes the request to be rejected. |

Table 6–5. Result Byte Index Values

| Value | Description |
|---|---|
| 13 | The line or station was not in the proper state for the requested action, which causes the request to be rejected. |
| 14 | A switched line error occurred. |

If the result byte index value is 1, 2, or 3, the station is placed in a NOT READY state; the MCS must perform a MAKE STATION READY/NOT READY (Type = 37) DCWRITE to ready the station if the NSP is to resume work for that station.

### Line Status Prior to Abort (MSG[1].[39:06])

Table 6–6 lists the values of the Line Status Prior To Abort field of the ERROR RESULT message. This field comes from the line table word of the NSP for the line in question and describes the NSP line status prior to the abort condition.

Table 6–6. Line Status Prior to Abort Values

| Field | Value | Description |
|---|---|---|
| [39:01] | 1 | The line is NOT READY. |
| [35:01] | 1 | The line is not connected. |

### NDLII LINE.TOG_1 and NDLII LINE.TOG_0

The NDLII variables LINE.TOG_1 and LINE.TOG_0 that are controlled by the line control procedure for the line are stored in this field. The meanings of these two bits are determined through convention between the NDLII programmer and the DCALGOL programmer.

## Switched Status Format of ERROR RESULT Message

### Message Format

| Word | Field | Value | Description |
|---|---|---|---|
| 0 | [47:08] | 5 | Class (for GOOD RESULT Class = 5). |
| | | 7 | Class (for SWITCHED STATUS RESULT Class = 7). |
| | | 99 | Class (for ERROR RESULT Class = 99). |
| | [23:24] | | LSN. |
| 1 | [47:08] | | Result byte index. |
| | | 00 | For SWITCHED STATUS RESULT Class = 7. |
| | | 13 | The connection cannot be completed because of the current status for ERROR RESULT Class = 99. |
| | | 4 | The message cannot be completed for ERROR RESULT (Result Class = 99). |
| | [39:08] | | The termination reason for the ERROR RESULT (Result Class = 99) of a DIALOUT (Type = 98) DCWRITE. |
| | | | TOGGLES; otherwise, one of the following: |
| | [30:07] | | Switched status byte. |
| | [30:01] | | SWITCHEDERROR. |
| | [28:01] | | SWITCHEDBUSY. |
| | [27:01] | | CONNECTED. |
| | [26:01] | | AUTOANSWER. |
| | [25:01] | | DIALOUT. |
| | [24:01] | | DIALIN (= 1). |
| 4 | [23:08] | | Original DCWRITE type. |
| | | 98 | For DIALOUT. |
| | | 99 | For DISCONNECT. |
| | | 101 | For INTERROGATE SWITCHED STATUS. |
| | | 102 | For SET/RESET AUTOANSWER. This field contains the value 101 if no original DCWRITE causes the result, that is, a spontaneous result. |

### Explanation

This format is used to return error results in response to the following:

- The DIALOUT (Type = 98) DCWRITE
- The DISCONNECT (Type = 99) DCWRITE
- The ANSWER THE PHONE (Type = 100) DCWRITE

The result byte index (MSG[1].[47:08]) has the value 13 or 14.

## Error Results in Switched Status Format

Table 6–7 lists the error results returned in switched status format.

Note that the general result message format also contains an error flag field.

**Table 6–7. Error Result, Switched Status Format**

| ResultField | Value |
|---|---|
| MSG[0].[47:08] | 99 |
| MSG[0].[39:16] | 0 |
| MSG[0].[39:08] | 0 |
| MSG[0].[31:08] | 0 |
| MSG[0],[23:24] | LSN |
| MSG[1].[47:08] | Result byte index |
| MSG[1].[39:08] | Termination reason or 0 |
| MSG[1].[30:07] | Switched status |
| MSG[4].[23:08] | Original DCWRITE type |

### Using Switched Status Format

Messages that have the switched status format occur in the four cases that follow:

- If any of the following DCWRITE types are initiated and succeed without failure, the system generates a SWITCHED STATUS RESULT (Result Class = 7) message with the switched status format:
  - DIALOUT (Type = 98) DCWRITE
  - DISCONNECT (Type = 99) DCWRITE
  - INTERROGATE SWITCHED STATUS (Type = 101) DCWRITE
- A SWITCHED STATUS RESULT (Result Class = 7) message with the switched status format is generated by the NSP whenever an unrequested change of (switched) status occurs. For example, the NSP automatically reports all unexpected disconnects and all connections made by the autoanswer capability.
- An ERROR RESULT (Result Class = 99) message with the switched status format is generated by the NSP whenever an error condition arises while the NSP is attempting to satisfy any of the following DCWRITE types:
  - DIALOUT (Type = 98) DCWRITE
  - DISCONNECT (Type = 99) DCWRITE
- The SET/RESET AUTOANSWER (DCWRITE Type = 102) DCWRITE generates a GOOD RESULT (Result Class = 5) message of the switched status format. The MCS can elect to receive the message or to have the system discard it.

### Switched Status Byte Values

The switched status byte contains five switched status flags as listed in Table 6–8.

**Table 6–8. Switched Status Byte Values**

| Field | Description |
|---|---|
| [28:01] | SWITCHEDBUSY: The bit is turned on whenever the NSP is in the process of changing its connect state (disconnecting if connected, answering if not connected, or dialing if not connected). |
| [27:01] | CONNECTED: The bit is turned on whenever the NSP is connected to a station on a dial network. |
| [26:01] | AUTOANSWER flag: This flag is an option by which the NSP can decide whether or not to answer incoming calls. |
| [25:01] | DIALOUT: A line is dialout (1) if its data set is connected to the telephone switching network (and has a telephone number that can be called by, and connected to, any of many remote stations); and (2) if its data set is associated with an auxiliary data set (an automatic calling unit) by which the system can dial up other remote stations. |

### Table 6–8. Switched Status Byte Values

| Field | Description |
|---|---|
| [24:01] | DIALIN: DIALIN = 1 for all lines for which the MCS might receive a message with the switched status format. A line is DIALIN if its data set is connected to the telephone switching network and has a telephone number that can be called by, and connected to, one of many remote stations. |

### Switched Status Format Flags after DIALOUT

In the case in which the original DCWRITE is DIALOUT (Type = 98), the switched status flags are as follows:

- If the message Class = 07 (SWITCHED STATUS RESULT), the result byte index = 00 (GOOD RESULT), SWITCHEDBUSY = 0, CONNECTED = 1, and DIALOUT = 1. The newly connected state, as this combination reports, also involves an automatic initiation of READY stations on the line.
- If the message class = 99 (ERROR RESULT) and if the result byte index = 13 (unable to initiate), the reason is that SWITCHEDBUSY = 1, or CONNECTED = 1. Because the NSP and an MCS are asynchronous from one another, a DCWRITE function that is known by the MCS to be consistent with the line status can be found not so by the NSP. For example, an MCS can present the DIALOUT (TYPE = 98) DCWRITE before having a chance to remove a SWITCHED STATUS RESULT showing connected, especially if the NSP was constructing it at the same time.
- If the message Class = 99 and if the result byte index = 14 (unable to complete), the reason is that the station called was busy, failed to answer in a reasonable amount of time, and so forth. Therefore, the status flags are as for an original DISCONNECT (Type = 99) DCWRITE with the result byte index = 00 or 14. (Refer to “Switched Status Format Flags after DISCONNECT” in this section.)

### Switched Status Format Flags after DISCONNECT

In the case in which the original DCWRITE is DISCONNECT (Type = 99), the switched status flags are as follows:

- If the message class = 07 (SWITCHED STATUS RESULT), the result byte index = 00 (GOOD RESULT), SWITCHEDBUSY = 0, and CONNECTED = 0.
- If the message class = 99 (ERROR RESULT) and if the result byte index = 13 (unable to initiate), the reason is that CONNECTED = 0 (already disconnected), or CONNECTED = 1 and SWITCHEDBUSY = 1 (already busy disconnecting).

### Switched Status Format Flags after INTERROGATE SWITCHED STATUS

If the original DCWRITE is INTERROGATE SWITCHED STATUS (Type = 101), the message class of the result message is always 07 for switched status result (no ERROR RESULT is possible), the result byte index = 00 (GOOD RESULT), and any of the possible combinations of flags described in the previous or subsequent cases of this section are possible.

### Switched Status Format Flags SET/RESET AUTOANSWER

If the original DCWRITE is SET/RESET AUTOANSWER (Type = 102), the message class is always 05 for GOOD RESULTS (no error result is possible), the result byte index = 00 (GOOD RESULT), the autoanswer flag is turned on or off appropriately, and all other switched status flags are returned as for an INTERROGATE SWITCHED STATUS (Type = 101) DCWRITE.

### Switched Status Format Flags after Automatic Switched Status

In the case in which no original DCWRITE exists and an automatic switched status result is generated by the NSP because of a change of state, the switched status flags have the following values:

- The message class = 07 (SWITCHED STATUS result)
- The result byte index = 00 (GOOD RESULT)
- The original DCWRITE type field equals 101 (for INTERROGATE SWITCHED STATUS) although no original DCWRITE existed.

Thus, messages that arise from the automatic message generation of the NSP because of a change of the state are indistinguishable from message results arising from an INTERROGATE SWITCHED STATUS (Type = 101) DCWRITE.

# Appendix A
# Sample MCS

The following is a sample of a simple MCS. The MCS can be transferred to, or a station can be assigned in the DATACOMINFO file.

```
BEGIN
FILE LINE(KIND=DISK,MAXRECSIZE=15,BLOCKSIZE=450,NEWFILE,
    PROTECTION=SAVE);
MESSAGE MSG,ERRMSG;
QUEUE PRIMQ,CURRQ;
QUEUE ARRAY REFERENCE INTERCOMQUEUES[0];
REAL MYNUM,RSLT,LSNR,FRSN,MSGSIZE,COUNT;
ARRAY DATA,ZIPARRAY[0:30];
POINTER PTEMP;
LABEL ABORT,ENDCURRQ;
TASK ARRAY TSK[0:6];
PROCEDURE PROX(A);
    ARRAY A[*]; EXTERNAL;
DEFINE
    TYPEF   = [47:8] #,
    VARF    = [39:8] #,
    VARIANTF= [39:16] #,
    LFSNF   = [22:23] #,
    LENGTHF = [23:24] #;
ALLOCATE(MSG,8);
MSG[0]:=0;
IF (RSLT:=DCWRITE(MSG,PRIMQ)) > 63 THEN
    GO TO ABORT;
MYNUM:=MSG[1];
SETUPINTERCOM(INTERCOMQUEUES,PRIMQ);

ON ANYFAULT,
   GO TO ABORT;
WHILE TRUE DO
    CASE WAIT(PRIMQ.QINSERTEVENT,CURRQ.QINSERTEVENT)-1 OF
        BEGIN
    (0):
            BEGIN                 % INPUT IN PRIMARY QUEUE
            MSGSIZE:=REMOVE(MSG,PRIMQ);
            LSNR:=MSG[0].LFSNF;
            IF (MSG[0].TYPEF = 1 AND     % STATION EVENT
                MSG[0].VARIANTF = 3)     % NEW STATION ACTIVITY
                OR MSG[0].TYPEF = 16     % STATION TRANSFER
            THEN
```

```
     BEGIN
    ALLOCATE(MSG,18);
     MSG[0]:=LSNR & 32 TYPEF;    % CHANGE CURRENT QUEUE AND
     MSG[2].VARIANTF:=26;        % GIVE GREETING
     REPLACE POINTER(MSG[6],8)
         BY "WELCOME TO B6000",
            "DATA COMM";
     IF (RSLT:=DCWRITE(MSG,CURRQ)) > 63 THEN
         GO TO ABORT;
     END
 ELSE
 IF MSG[0].TYPEF = 1
    AND MSG[0].VARIANTF = 0 AND MSG[1].[12:1] = 1
 THEN               % STATION EVENT AND ? COMMAND
     BEGIN
     ALLOCATE(ERRMSG,6+((MSG[2].VARIANTF+5) DIV 6));
     % GIVE TO CONTROLLER
     ERRMSG[0]:=0 & 21 TYPEF & 2 VARF
                  & 1 [46:1] & MYNUM [31:8] & LSNR LENGTHF;
     ERRMSG[5]:=MSG[2].VARIANTF-1;
     REPLACE POINTER(ERRMSG[6]) BY POINTER(MSG[6])+1
         FOR MSG[2].VARIANTF-1;
     INSERT(ERRMSG,INTERCOMQUEUES[0]);
     MSG[0]:=MSG[0].LFSNF & 33 TYPEF;
     % REFLECT THE COMMAND
     IF (RSLT:=DCWRITE(MSG)) > 63 THEN
         GO TO ABORT;
     END

ELSE
 IF MSG[0].TYPEF = 21 AND MSG[0].VARF = 3 THEN
     BEGIN                % CONTROLLER RESPONSE
     ALLOCATE(ERRMSG,6+((MSG[1]+5) DIV 6));
     ERRMSG[0]:=0 & 33 TYPEF
                  & REAL(MSG[0].[14:15]) LENGTHF;
     ERRMSG[2].VARIANTF:=MSG[1];
     REPLACE POINTER(ERRMSG[6])
         BY POINTER(MSG[2]) FOR MSG[1];
     IF (RSLT:=DCWRITE(ERRMSG,CURRQ)) > 63 THEN
         GO TO ABORT;
     END
 ELSE
 IF MSG[0].TYPEF = 99 THEN
     BEGIN                 % ERROR MESSAGE
     ALLOCATE(ERRMSG,6);
     ERRMSG[0]:=LSNR & 37 TYPEF & 1 VARIANTF;
     IF (RSLT:=DCWRITE(ERRMSG)) > 0 THEN
         GO TO ABORT;
     IF MSG[1].TYPEF GEQ 4 AND MSG[1].TYPEF LEQ 6 THEN
         BEGIN
         ALLOCATE(ERRMSG,8);
         ERRMSG[0]:=LSNR & 96 TYPEF;
```

```
                IF (RSLT:=DCWRITE(ERRMSG)) > 63 THEN
                   GO TO ABORT;
                END;
            END
        ELSE
        IF MSG[0].TYPEF = 2  THEN
            BEGIN                    % FILE OPEN
            LSNR:=MSG[0].LFSNF;
            FRSN:=MSG[6].LFSNF;
            ALLOCATE(MSG,6);
            MSG[0]:=FRSN & 64 TYPEF;
            IF (RSLT:=DCWRITE(MSG)) > 63 THEN
                GO TO ABORT;
            ALLOCATE(MSG,12);
            MSG[0]:=LSNR & 33 TYPEF;
            REPLACE POINTER(MSG[6],8) BY
                "STATION ATTACHED TO FILE" FOR 24;
            MSG[2].VARIANTF:=24;
            IF (RSLT:=DCWRITE(MSG)) > 63 THEN
                GO TO ABORT;
            END

       ELSE
        IF MSG[0].TYPEF = 4  THEN
            BEGIN                     % FILE  CLOSE
            ALLOCATE(MSG,12);
            MSG[0]:=LSNR & 33 TYPEF;
            REPLACE POINTER(MSG[6],8)
                BY "FILE  CLOSED"  FOR 11;
            MSG[2].VARIANTF:=11;
            IF (RSLT:=DCWRITE(MSG)) > 63 THEN
                GO TO ABORT;
            END
        ELSE
            BEGIN
            REPLACE POINTER(DATA) BY POINTER(MSG) FOR 48;
            WRITE(LINE,<6(H12,X1)>,DATA[*]);
            END;
        END OF PRIMQ CASE;
(1):
        BEGIN       % INPUT INTO CURRENT QUEUE
        MSGSIZE:=REMOVE(MSG,CURRQ);
        REPLACE PTEMP:POINTER(DATA,8) BY POINTER(MSG[6],8)
            FOR (COUNT:=MSG[2].VARIANTF);
        SCAN PTEMP:PTEMP FOR COUNT:COUNT UNTIL NEQ " ";
        IF PTEMP = "QUIT" THEN
            GO TO ABORT;
        IF PTEMP = "ZIP" FOR 3 THEN
            BEGIN
            PTEMP:=PTEMP+3;
            REPLACE POINTER(ZIPARRAY) BY 4"6F" FOR 1,
                   PTEMP FOR COUNT-3,"; END." FOR 6;
                ZIP WITH ZIPARRAY[*];
               END
            ELSE
            IF PTEMP = "ACCOUNT" THEN
                BEGIN
                REPLACE TSK[0].NAME
                    BY "OBJECT/ACCOUNT/STATUS/REPORT ON LEDGER.";
                REPLACE TSK[0].FILECARDS
                    BY "FILE LINE(KIND=REMOTE);";
                TSK[0].SOURCESTATION:=LSNR;
                TSK[0].STATION:=LSNR;
                REPLACE POINTER(ZIPARRAY) BY PTEMP+4 FOR COUNT-4;
                PROCESS PROX(ZIPARRAY) [TSK[0]];
                END;
            MSG[0]:=MSG[0].LFSNF & 33 TYPEF;
            IF (RSLT:=DCWRITE(MSG)) > 63 THEN
                GO TO ABORT;

ENDCURRQ:
            END OF CURRENT QUEUE CASE;
    END OF CASE STMT;
ABORT:
WRITE(LINE,<"MCS ABORTED ON DCWRITE ERROR NO." ,R10.0>,RSLT);
ALLOCATE(MSG,7);
MSG[0]:=LSNR & 45 TYPEF & 1 [25:1];
RSLT:=DCWRITE(MSG);
END.
```

The following is another sample of a simple MCS.

```
 BEGIN
  MESSAGE MSG;
  QUEUE PRIMARYQ;
  INTEGER RESULT;
  REAL LOC;
  LABEL ABORT;
%
% The following code segment initializes the primary queue.
%
  ALLOCATE(MSG,6);
  MSG[0].[47:8] := 0;
  IF (RESULT := DCWRITE(MSG,PRIMARYQ)) > 0 THEN
    GO TO ABORT;
%
% The following code segment extracts the COMS mix number using its
% MCS number, 1 in this case (MCS numbers are assigned in the order
% they are declared in the DATACOMINFO file).
%
  ALLOCATE(MSG,6);
  MSG[0] := 1 & 2[47:08];
  IF (RESULT := DCWRITE(MSG)) > 0 THEN
    GO TO ABORT;
  LOC := MSG[MSG[MSG[6].[07:08]].[15:08]].[15:16];
  DISPLAY("COMS MIX NUMBER IS: " CAT STRING(LOC,*));
%
%  The following code segment extracts the COMS mix number using its
%  MCS name as declared in the DATACOMINFO file.
%
  ALLOCATE(MSG,9);
  MSG[0] := 0 & 2[47:08];
  REPLACE POINTER(MSG[6],8) BY "SYSTEM/COMS.";
  IF (RESULT := DCWRITE(MSG)) > 0 THEN
    GO TO ABORT;
  LOC := MSG[MSG[MSG[6].[07:08]].[15:08]].[15:16];
  DISPLAY("COMS MIX NUMBER IS: " CAT STRING(LOC,*));
%
%  The following code segment extracts the CANDE mix number using
%  its MCS number, 2 in this case, and then using its name as declared
%  in the DATACOMINFO file.
%
  ALLOCATE(MSG,6);
  MSG[0] := 2 & 2[47:08];
  IF (RESULT := DCWRITE(MSG)) > 0 THEN
    GO TO ABORT;
  LOC := MSG[MSG[MSG[6].[07:08]].[15:08]].[15:16];
  DISPLAY("CANDE MIX NUMBER IS: " CAT STRING(LOC,*));
%
 ALLOCATE(MSG,9);
 MSG[0] := 0 & 2[47:08];
 REPLACE POINTER(MSG[6],8) BY "SYSTEM/CANDE.";
 IF (RESULT := DCWRITE(MSG)) > 0 THEN
   GO TO ABORT;
 LOC := MSG[MSG[MSG[6].[07:08]].[15:08]].[15:16];
 DISPLAY("CANDE MIX NUMBER IS: " CAT STRING(LOC,*));
%
ABORT:
  DISPLAY ("DCWRITE ERROR " CAT STRING(RESULT,*));
END.
```

# Appendix B
# Reserved Words

Three types of reserved words are defined:

**Type** **Description**

Type 1 Reserved words are words that cannot be used as identifiers anywhere in the source program.

Type 2 Reserved words are words that can be declared to be identifiers (overriding their reserved meaning) but, if undeclared, have a well-defined meaning.

Type 3 Reserved words are words that can be declared to be identifiers but, where used in the language as specified by the syntax, have the reserved meaning.

## Type 1

DCALGOL Type 1 reserved words consist of the ALGOL Type 1 reserved words plus the following:

- EPILOG
- EXCEPTION
- MESSAGE
- QUEUE

## Type 2

DCALGOL Type 2 reserved words consist of the ALGOL Type 2 reserved words plus the following:

- ALLOCATE
- ATTACHSPOQ
- CHECKGUARDFILE
- COMBINE
- CONTROLCARD
- DCERRANALYSIS

- DCERRORLOGGER
- DCKEYIN
- DCSYSTEMTABLES
- DCWRITE
- FLUSH
- GETSTATUS
- INSERT
- MAKEUSER
- MCSLOGGER
- MLSCAPABLE
- NULL
- QUEUEINFO
- REMOVE
- RESIDENT
- SETSTATUS
- SETUPINTERCOM
- SNR
- SYSTEMSTATUS
- USERDATA
- USERDATAFREEZER
- USERDATALOCATOR
- USERDATAREBUILD
- WRITESPO

# Type 3

DCALGOL Type 3 reserved words consist of the ALGOL Type 3 reserved words plus the following:

- QACTIVE
- QBLOCKSIZE
- QDISKERROR
- QHEADSIZE
- QINSERTEVENT
- QMEMORYLIMIT
- QMEMORYSIZE
- QMESSAGECOUNT

- QREMOVEWAIT
- QROWSIZE
- QSIZE
- QTANK
- QUSERCOUNT

# Appendix C
# 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:

```
── REMOVE ─┬──────────┬─────────────────────────────────────────┤
           ├─ SOURCE ─┤
           └─ OBJECT ─┘
```

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.

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:

| 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.

```
── BEGIN ──<statement list>── END ─────────────────────────────────┤
```

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 (>)

- 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.

```
──┬─────┬─<arithmetic expression>──────────────────────────────┤
  ├─ + ─┤
  └─ – ─┘
```

## 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.

```
  ┌←────── ; ─────┐
──┴─<field value>─┴──────────────────────────────────────────────┤
```

## 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.

```
  ┌←─────── , ──────┐
──┴─/2\─┬─ LINKAGE ─┬┴──────────────────────────────────────────┤
        └─ RUNTIME ─┘

  ┌←─/2\────────┐
──┴┬─ LINKAGE ─┬┴───────────────────────────────────────────────┤
   └─ RUNTIME ─┘
```

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.

```
  ┌←─────── , ───────┐
──┴┬─/2*\─ LINKAGE ─┬┴──────────────────────────────────────────┤
   └─ RUNTIME ──────┘
```

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>.

```
         ┌←──────── , ─────┐
── & ──┬─┴─┬─ TYPE ──────┬─┴──┬──────────────────────────────────────┤
       │   ├─ ASCII ─────┤    │
       │   ├─ BCL ───────┤    │
       │   ├─ DECIMAL ───┤    │
       │   ├─ EBCDIC ────┤    │
       │   ├─ HEX ───────┤    │
       │   └─ OCTAL ─────┘    │
       ├─ ADDRESS ────────────┤
       └─ ALTER ──<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>**

```
── LOCK ── ( ── <file identifier> ── ) ──────────────────────────┤
```

| 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.<br>The parentheses are required punctuation, and FILE4 is a sample file identifier. |

### Example 2

**<open statement>**

```
── OPEN ─┬───────────┬─<database name>──────────────────────────┤
         ├─ INQUIRY ─┤
         └─ UPDATE ──┘
```

| Sample Input | Explanation |
|---|---|
| OPEN DATABASE1 | The constant OPEN is followed by the variable DATABASE1, which is a database name.<br>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>

```
── GENERATE ──<subset>── = ─┬─ NULL ────────────────────────┬──────────────┤
                            └<subset>─┬─────────────────────┤
                                      ├─ AND ─┬─<subset>────┘
                                      ├─ OR ──┤
                                      ├─ + ───┤
                                      └─ – ───┘
```

| 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>

```
                      ┌←─────────────────── , ───────────────────┐
── ENTITY REFERENCE ──┴─<entity ref ID>── ( ──<class ID>── ) ────┴──────────┤
```

| 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

```
── PS ── MODIFY ─────────────────────────────────────────────────────→

    ┌←──────────────────── , ─────────────────┐
→─┬─┴─┬─<request number>────────────────────┬─┴─┬─────────────────────→
  │   └─<request number>── ─ ──<request number>─┘   │
  └─ ALL ─┬─────────────────────────────────────────┤
          └─ EXCEPTIONS ────────────────────────────┘

→─┬────────────────────────────────────────┬───────────────────────────┤
  │ ┌←──────────────── , ─────────────┐    │
  └─┴─┬──────┬─<file attribute phrase>─┬─┴────┘
      │└ ─ ─ ┘                          │
      └──────┬─<print modifier phrase>─┘
       └ ─ ─ ┘
```

| Sample Input | Explanation |
|---|---|
| PS MODIFY 11159 | The constants PS and MODIFY are followed by the variable 11159, which is a request number. |
| PS MODIFY<br>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<br>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 D
# Related Product Information

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

***MCP/AS Data Communications Protocols Installation and Implementation Guide* (8600 0486)**

This guide describes the purpose of protocols and procedures for installing protocols. It also provides reference material useful in interpreting dumps associated with data communications data link processors (DCDLPs), enhanced data communications data link processors (EDCDLPs), and data communications adapters (DCAs).

***MCP/AS Interactive Datacomm Configurator (IDC) Operations Guide* (8600 1880)**

This guide explains how to use IDC, a menu-driven utility used to define and modify data communications networks. It provides information on configuring a data communications network using the IDC menu system and basic constructs, and provides reference information about the commands and attributes. This guide is written for individuals who have a basic knowledge of data communications concepts, but who might not know the physical characteristics of hardware devices in the network.

***MCP/AS Network Definition Language II (NDLII) Programming Reference Manual* (8807 7011)**

This manual documents the high-level programming and definition language used to describe a data communications network. It gives a brief overview and functional description of the data comm system and provides a complete description of the syntax and semantics of all language components and compiler options of NDLII.

***Unisys e-@ction Application Development Solutions ALGOL Programming Reference Manual, Volume 1: Basic Implementation* (8600 0098)**

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

***Unisys e-@ction ClearPath Enterprise Servers File Attributes Programming Reference Manual* (8600 0064)**

This manual contains information about each file attribute and each direct I/O buffer attribute. The manual is written for programmers and operations personnel who need to understand the functionality of a given attribute. The *I/O Subsystem Programming Guide* is a companion manual.

***Unisys e-@ction ClearPath Enterprise Servers GETSTATUS/SETSTATUS Programming Reference Manual* (8600 0346)**

This manual explains how to use the various GETSTATUS and SETSTATUS calls used in the DCALGOL programming language. The manual is written for experienced ALGOL programmers who are involved with data communications.

***Unisys e-@ction ClearPath Enterprise Servers MCP System Interfaces Programming Reference Manual* (8600 2029)**

This manual describes selected library objects exported from the MCPSUPPORT library, and describes the ARCHIVESUPPORT, BILLINGSUPPORT, and TAPEMANAGER libraries. This manual is written for system programmers who want to write programs that interface with the system software.

***Unisys e-@ction ClearPath Enterprise Servers MultiLingual System Administration, Operations, and Programming Guide* (8600 0288)**

This guide describes how to use the MLS environment, which encompasses many products. The MLS environment includes a collection of operating system features, productivity tools, utilities, and compiler extensions. The guide explains how these products are used to create application systems tailored to meet the needs of users in a multilingual or multicultural business environment. It explains, for example, the procedures for translating system and application output messages, help text, and user interface screens from one natural language to one or more other languages; for instance, from English to French and Spanish. This guide is written for international vendors, branch systems personnel, system managers, programmers, and customers who wish to create customized application systems.

***Unisys e-@ction ClearPath Enterprise Servers Security Administration Guide* (8600 0973)**

This guide describes systems-level security features and suggests how to use them. It provides administrators with the information necessary to set and implement effective security policy. This guide is written for system administrators, security administrators, and those responsible for establishing and implementing security policy.

***Unisys e-@ction ClearPath Enterprise Servers System Commands Operations Reference Manual* (8600 0395)**

This manual gives a complete description of the system commands used to control system resources and work flow. This manual is written for systems operators and administrators.

### *Unisys e-@ction ClearPath Enterprise Servers System Log Programming Reference Manual* (8600 1807)

This manual describes the format and contents of all the Major Type and Minor Type entries of the system log. It also contains information about controlling the log contents and about writing log analysis programs.

### *Unisys e-@ction ClearPath Enterprise Servers SYSTEMSTATUS Programming Reference Manual* (8600 0452)

This manual documents the SYSTEMSTATUS intrinsic of the Master Control Program (MCP). The SYSTEMSTATUS intrinsic provides information that can be used to efficiently monitor the performance of a running system. This manual is written for systems programmers.

### *Unisys e-@ction ClearPath Enterprise Servers Task Attributes Programming Reference Manual* (8600 0502)

This manual describes all of the task attributes that are available. It also give examples of statements for reading and assigning task attributes in various programming languages. The *Task Management Programming Guide* is a companion manual.

### *Unisys e-@ction Transaction Server for ClearPath MCP Configuration Guide* (8600 0312)

This guide provides an overview of the basic concepts and functions of COMS. It includes instructions for creating a working COMS configuration and information on how to monitor and fine-tune COMS system performance. This guide is written for installation analysts, systems analysts, programmers, administrators, and performance analysts.

# Index

86000841-202