UNISYS

A Series

Task Attributes

**Programming
Reference Manual**

UNISYS

Product Information Announcement

○ New Release   ○ Revision   ● Update   ○ New Mail Code

Title

A Series Task Attributes Programming Reference Manual

This Product Information Announcement announces Update 1 to the September 1991 publication of the *A Series Task Attributes Programming Reference Manual.* The update is relative to the Mark 4.0.0 System Software Release, dated July 1992.

Various technical changes have been made to improve the quality and usability of the document.

Remove	Insert
iii through iv	iii through iv
ix through xvi	ix through xvi
1–9 through 1–10	1–9 through 1–10
	1–10A through 1–10B
1–25 through 1–28	1–25 through 1–28
2–5 through 2–6	2–5 through 2–6
2–27 through 2–28	2–27 through 2–28
2–77 through 2–102	2–77 through 2–102
	2–102A through 2–102B
2–107 through 2–108	2–107 through 2–108
	2–108A through 2–108B
2–113 through 2–118	2–113 through 2–118
	2–118A through 2–118B
2–133 through 2–136	2–133 through 2–136
2–149 through 2–152	2–149 through 2–152
2–157 through 2–160	2–157 through 2–160
2–169 through 2–174	2–169 through 2–174
2–177 through 2–178	2–177 through 2–178
2–191 through 2–196	2–191 through 2–196
	2–196A through 2–196B
2–211 through 2–212	2–211 through 2–212
Glossary–17 through 18	Glossary–17 through 18
Bibliography–1 through 2	Bibliography–1 through 2
Index–1 through 10	Index–1 through 10

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

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

To order additional copies of this document

- United States customers call Unisys Direct at 1-800-448-1424

- All other customers contact your Unisys Subsidiary Librarian

- Unisys personnel use the Electronic Literature Ordering (ELO) system

UNISYS

A Series

Task Attributes

Programming Reference Manual

Page Status

Page	Issue
iii through iv	–010
v through vii	–000
viii	Blank
ix through xiii	–010
xiv	Blank
xv	–010
xvi	Blank
1–1 through 1–8	–000
1–9 through 1–10B	–010
1–11 through 1–24	–000
1–25 through 1–27	–010
1–28	Blank
2–1 through 2–4	–000
2–5 through 2–6	–010
2–7 through 2–26	–000
2–27 through 2–28	–010
2–29 through 2–76	–000
2–77 through 2–102B	–010
2–103 through 2–106	–000
2–107 through 2–108A	–010
2–108B	Blank
2–109 through 2–112	–000
2–113 through 2–118B	–010
2–119 through 2–132	–000
2–133 through 2–136	–010
2–137 through 2–148	–000
2–149 through 2–152	–010
2–153 through 2–156	–000
2–157 through 2–160	–010
2–161 through 2–168	–000
2–169 through 2–174	–010
2–175 through 2–176	–000
2–177 through 2–178	–010
2–179 through 2–190	–000
2–191 through 2–196A	–010
2–196B	Blank
2–197 through 2–210	–000
2–211 through 2–212	–010
2–213 through 2–223	–000
2–224	Blank
A–1 through A–8	–000

continued

continued

Page	Issue
Glossary–1 through 16	–000
Glossary–17 through 18	–010
Bibliography–1 through 2	–010
Index–1 through 9	–010
Index–10	Blank

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

About This Manual

Purpose

This manual describes the A Series task attributes, which are used to record or control various aspects of process behavior. All processes on an A Series system possess all the task attributes described in these pages, though the values of the individual attributes vary from one process to another. The operating system makes use of these attributes in executing a process. Some programming languages also allow you to write applications that query or modify task attributes.

Scope

This manual serves as a complete reference to the uses of task attributes on A Series systems. This manual also provides some examples of statements for accessing task attributes from various sources. However, the formal syntax of these statements is documented in other manuals. For example, the *A Series Work Flow Language (WFL) Programming Reference Manual* provides syntax diagrams for the WFL statements that query or modify task attributes. See "Related Product Information" later in this section for the titles of these companion manuals.

Audience

The audience for this manual consists of programmers who write tasking applications and operators who use task equations.

Prerequisites

Before reading this manual, you should have a basic familiarity with A Series concepts; if not, refer to the *A Series Systems Functional Overview*. A more detailed introduction to tasking and the use of task attributes is provided in the *A Series Task Management Programming Guide*.

How to Use This Manual

This manual is intended primarily as a reference source for information about particular task attributes. For ease of access, these attributes are all presented in Section 2 in alphabetical order. To find a particular task attribute most quickly, open the manual at the "Task Attribute Descriptions" tab and riffle through the pages, noting the task attribute headings on the upper outside corners of the pages.

In this manual, the ANSI-74 version of COBOL is referred to as COBOL74, and the ANSI-68 version is referred to as COBOL(68). Most references to COBOL in this manual mention only COBOL74. However, many of the COBOL74 features mentioned are also available in COBOL(68). For details, refer to the *A Series COBOL ANSI-68 Programming Reference Manual*.

This manual uses railroad diagrams to illustrate the possible values of some task attributes. If you are not familiar with this type of syntax notation, you should read Appendix A of this manual, "Understanding Railroad Diagrams."

Organization

The individual sections and the appendix are described in the following outline of the manual.

Section 1. Accessing Task Attributes

This section explains how system administrators, operators, programmers, and end users can access task attribute values. Examples are given of statements that access various types of task attributes. This section also explains how the system resolves conflicting values and how task attribute errors are handled.

Section 2. Task Attribute Descriptions

This section provides a detailed description of all task attributes. The following qualities of each task attribute are described:

- Name
- Type
- Units
- Range
- Default
- Read time
- Write time
- Inheritance
- Overwrite rules
- Host Services support

- Attribute number
- Synonyms
- Language restrictions
- Explanation of uses
- Run-time errors

Some task attribute descriptions also include examples.

Appendix A. Understanding Railroad Diagrams

This appendix explains how to interpret the syntax diagrams used in this manual.

In addition, this manual includes a glossary, a bibliography, and an extensive index. The index includes page references for all error messages discussed in this manual.

Related Product Information

The following manuals provide useful background information about tasking and about the syntax for task attribute access in various programming languages.

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

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

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

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

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

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

A Series Work Flow Language (WFL) Programming Reference Manual (form 8600 1047)

This manual presents the complete syntax and semantics of WFL. WFL is used to construct jobs that compile or run programs written in other languages and that perform library maintenance such as copying files. This manual is written for individuals who have some experience with programming in a block-structured language such as ALGOL and who know how to create and edit files using CANDE or the Editor.

Contents

Contents

Contents

Appendix A. Understanding Railroad Diagrams

Figures

Tables

Section 1
Accessing Task Attributes

The types of task attribute access that are possible are reading a task attribute
and assigning a value to a task attribute. The end user, programmer, operator, and
system administrator all have the ability to access the task attributes of a process in
various ways. The system software also plays an important role in providing default
and inherited values, resolving conflicting assignments, and issuing errors for invalid
attempts to access task attributes.

Operator and End-User Access to Task Attributes

The operator or end user can affect the task attributes of a process with commands
entered in Command and Edit (CANDE) or Menu-Assisted Resource Control (MARC)
sessions or at the operator display terminal (ODT).

Using CANDE and MARC Task Equations

You can make task attribute assignments in CANDE or MARC by using *task equations*.
Task equations are task attribute assignments that you can append to a process
initiation statement. The system applies these assignments before initiating the process.

In CANDE, you can include task equations after most process initiation statements,
including RUN and UTILITY. In MARC, you can include task equations after the RUN
command. Also, if you initiate a process from the RUN screen, you can enter task
equations on the TASKATTR screen and the FILEEQUATE screen.

The following is a CANDE example:

```
RUN ALGOL/TASK;SW1=TRUE;MAXPROCTIME=20;
              FILE IN=DAILY/DATA;FILE OUT(KIND=DISK,TITLE=OUTPUT);
```

The preceding example shows assignments to several types of task attributes. SW1 is
a Boolean attribute, and MAXPROCTIME is a real attribute. The FILE IN and FILE
OUT assignments are examples of the syntax for assigning the FILECARDS task
attribute.

You can also include task equations after a CANDE *COMPILE* command. Such task
equations can make assignments to the compilation or the resulting object code file. For
details, refer to "Assigning Task Attributes to an Object Code File" in this section.

Note that a process can change the values of many of its own task attributes while it
is running. Thus, a programmer can design a process to override the effects of task
equations submitted by operators.

Assigning Task Attributes to a Session

When you initiate a process from a CANDE or MARC session, the process inherits a
number of task attributes from the session. You can make assignments to some of
the task attributes of the session by using special CANDE and MARC commands such
as FAMILY, LANGUAGE, and so on. Thereafter, all the processes you initiate from
the session inherit these values, unless you override them with task equations. For
details, refer to the discussion of tasking from interactive sources in the *A Series Task
Management Programming Guide*.

Using Operator Commands

You can use any of several system commands to make assignments to the task attributes
of a running process. These system commands, or close equivalents to them, can be
entered at an ODT or in a MARC or CANDE session. These include communication
commands, which affect such task attributes as EXCEPTIONEVENT, ACCEPTEVENT,
and TASKVALUE. You can use other commands to change the PRIORITY value or to
change the STATUS value of the process. For details, refer to the discussion of tasking
from interactive sources in the *A Series Task Management Programming Guide*.

Programmer Access to Task Attributes

You can access task attributes in either of two ways:

- Through language constructs in Work Flow Language (WFL), ALGOL, COBOL(68),
  COBOL74, PL/I, and APLB

- Through calls on the WFLSUPPORT library

The following subsections discuss the WFL, ALGOL, and COBOL74 language constructs
for reading and assigning task attributes, as well as the WFLSUPPORT interface.

For information about task attribute access in APLB, refer to the *A Series APLB
Programming Reference Manual*. For information about task attribute access in
COBOL(68), refer to the *A Series COBOL ANSI-68 Programming Reference Manual*.
For details about task attribute access in PL/I, refer to the *A Series PL/I Reference
Manual*.

Standard Methods for Accessing Task Attributes

You can access task attributes from programs by any of several means, including task
variables, task equations, the WFL job attribute list, and object code file assignments.

Using Task Variables

Task variables are the main means of accessing task attributes from programs. A task
variable is an object that is declared in a program and that accesses the task attributes of
a particular process. The task variable becomes associated with a particular process by

being specified in the statement that initiates that process. For example, the following COBOL74 statement initiates a process and associates the task variable TASK-VAR-1 with that process:

```
PROCESS TASK-VAR-1 WITH PROC-EXTERNAL.
```

Certain predeclared task variables are available that are automatically associated with a particular process. The MYSELF task variable allows a process to access its own task attributes. The MYJOB task variable accesses the task attributes of the job of the process. The task attribute PARTNER accesses the task attributes of the partner process, and the task attribute EXCEPTIONTASK accesses the task attributes of the exception task.

Additionally, a process can access any task variable that is within the extended addressing environment of the outer block of the process. For example, if the process is an internal task, it can access task variables declared globally in its parent. The process can access any task variables declared in its own code. The process can also access any task variables that are passed as parameters.

Task attributes can be assigned to a task variable before the task variable is used in a process initiation statement. These task attributes are assigned to the new process when it is initiated. If the same task attribute is assigned more than once, the most recent value assigned is used when the process is initiated. If the task attributes of the task variable are read before initiation, they return their default values or the values ·they were previously assigned.

If a task variable is associated with a dependent process in the initiation statement, then the task variable remains associated with the process after initiation. The task variable can be used to access the task attributes of the running process. Assignments to the task variable can change the behavior of the process. Interrogations of the task variable can be used to monitor the status of the process.

If a task variable is associated with an independent process in the initiation statement, then any task attributes that were previously assigned to the task variable are applied to the independent process. However, once initiation completes, the task variable ceases to be associated with the independent process. The task attributes of the task variable can be read or written to; however, these operations do not access the task attributes of the independent process.

Once the process has terminated, the task variable can be used to examine the final values of the task attributes of the process. For example, the history-related attributes of the task variable can be examined for information about how the process terminated.

Reusing Task Variables

The same task variable can be specified in more than one task initiation statement in a program. However, the same task variable cannot be associated with two processes at the same time. For example, the following pair of ALGOL statements causes an error:

```
PROCESS PROG1 [T];
PROCESS PROG2 [T];
```

Because the first statement initiates an asynchronous process, task variable T is still in use when the second statement is executed. An ALGOL process that executes the statements in the previous example is discontinued with the run-time error "INITIATE ACTIVE TASK".

More obscure problems can arise from task attributes being carried over from one use of the task variable to another. Consider the following ALGOL statements:

```
CALL PROG1 [T];
CALL PROG2 [T];
```

No error results from these statements, because PROG1 is initiated as a synchronous process. The statement that initiates PROG2 is not executed until PROG1 terminates. However, PROG1 might have used the MYSELF task variable to make an assignment to its FAMILY task attribute. This new FAMILY value is passed on to PROG2, simply because it uses the task variable that was previously associated with PROG1. Other task attribute values can also be passed on in this way.

This problem can be prevented by declaring a different task variable for each process that is to be initiated. The task variable can also be made safe for reuse by reinitializing it. A task variable can be reinitialized by setting the STATUS task attribute to NEVERUSED. This assignment causes all task attributes to be returned to their default values. The following ALGOL statement reinitializes a task variable:

```
TVAR.STATUS := VALUE(NEVERUSED);
```

WFL also provides the INITIALIZE statement for reinitializing task variables. The following is an example of this statement:

```
INITIALIZE (TVAR);
```

These statements reinitialize the task variable only if it is not currently in use. That is, the current value of the STATUS task attribute must be TERMINATED, BADINITIATE, or NEVERUSED. Otherwise, the assignment has no effect.

Using WFL Task Equations

You can use task equations in WFL jobs that are similar to the task equations allowed in CANDE or MARC sessions. You can include task equations after a process initiation statement, such as RUN or PROCESS. Where task equations conflict with previous assignments to the task variable, the task equations take precedence. The following is an example of a WFL job that uses task equations:

```
100 ?BEGIN JOB WFL/TEST;
200   TASK T (TASKVALUE = 3);
300   RUN OBJECT/ALGOL/TASK [T];
400     TASKVALUE = 1;
500 ?END JOB
```

In this example, OBJECT/ALGOL/TASK runs with a TASKVALUE of 1 because the task equation overrides the previous assignment to the task variable.

You can also use task equations with the COMPILE statement to make assignments to the compilation or the resulting object code file. For details, refer to "Assigning Task Attributes to an Object Code File" in this section.

Note that a process can change the values of many of its own task attributes while it is running. Thus, a programmer can design a process to override the effects of task equations submitted through WFL.

Using the WFL Job Attribute List

A WFL job attribute list consists of task attribute assignments in the WFL source program, immediately following the job header. The system applies the assignments in the job attribute list before initiating the job. This feature can be useful because some task attributes can be assigned to a process only before initiation (an example is the CLASS task attribute).

The following is an example of a WFL job with a job attribute list that assigns the CLASS, CHARGE, and JOBSUMMARY task attributes:

```
?BEGIN JOB RUNNER;
   CLASS = 2;
   CHARGE = ORDERS;
   JOBSUMMARY = SUPPRESSED;
 RUN OBJECT/TAU ON PACK;
?END JOB
```

Assigning Task Attributes to an Object Code File

In some cases, you might want certain task attributes to be assigned the same values each time a program is run. For many task attributes, you can achieve this effect by including statements in the source program that assign task attributes to the MYSELF task variable. However, some task attributes can only be assigned before process initiation. For a WFL job, you can assign such task attributes in the job attribute list. For programs written in other languages, you can assign such task attributes to the object code file. The task attributes stored in the object code file are used whenever the object code file is initiated, unless they are overridden by later task attribute assignments.

You can assign task attributes to the object code file at compile time through the use of compiler task equations, which can be included in the WFL or CANDE *COMPILE* statements. You must be careful to distinguish between task equations that affect the compilation itself and task equations that affect the resulting object code file. The following WFL example uses compiler task equations:

```
500 COMPILE OBJECT/X WITH ALGOL LIBRARY;
600    COMPILER FILE CARD (TITLE = X, KIND = DISK);
700    ALGOL PRIORITY = 50;
800    TASKVALUE = 3;
```

In both WFL and CANDE, task equations are applied to the compilation if they are preceded by the word *COMPILER* or the name of the compiler. Otherwise, the task

equations are applied to the object code file. In the preceding example, the task equations at lines 600 and 700 are applied to the compilation. The task equation at line 800 is applied to the object code file.

WFL includes a statement that can be used to make task attribute assignments to an existing object code file. This is the MODIFY statement. Task attributes that are stored by a MODIFY statement have the same effect as task attributes assigned at compile time: they serve as default values for every execution of that object code file. They also override any conflicting task attribute assignments that were made at compile time. The following is an example of a MODIFY statement:

```
MODIFY OBJECT/X;
    CHARGE = ADMIN;
    FILE INPUT = (JAS)DOC/1Ø3 ON DOCPK;
```

Task Attribute Syntax Examples

Different task attributes store different types of values. Most task attributes store values that are of type Boolean, event, integer, mnemonic, real, string, or task. The following pages give examples of how these various types of task attributes can be read or assigned in WFL, ALGOL, and COBOL74. For information about how to access task attributes that are of irregular types, refer to the task attribute descriptions in Section 2, "Task Attribute Descriptions."

Accessing Boolean Task Attributes

Boolean task attributes have a value of TRUE or FALSE. In WFL, these values can be read or assigned directly, or the task attribute can be used in other Boolean expressions. WFL also allows the use of a null assignment, which assigns a value of TRUE. Thus, the following two statements are equivalent. (In these statements, T is a task variable.)

```
T (DISPLAYONLYTOMCS = TRUE);
T (DISPLAYONLYTOMCS);  % Null assignment; assigns a value of TRUE
```

The following WFL examples show the use of Boolean task attributes as expressions. BOOL is a Boolean variable and T is a task variable.

```
BOOL := T(LOCKED);
IF T(SW1) THEN DISPLAY "NO ERRORS FOUND";
```

The ALGOL syntax is similar, except that task attributes are preceded by periods instead of enclosed in parentheses. In the following examples, BOOL is a Boolean variable and T is a task variable:

```
T.DISPLAYONLYTOMCS := TRUE;
BOOL := T.LOCKED;
IF T.SW1 THEN BOOL := TRUE;
```

In COBOL74, Boolean task attributes return a value of 0 if FALSE or 1 if TRUE. Boolean attributes must be moved into a numeric receiving field. However, the VALUE

function can be used when assigning or reading Boolean values. In the following examples, BOOLVAL was declared as *77 BOOLVAL BINARY PIC 9(11)*.

```
MOVE ATTRIBUTE LOCKED OF MYSELF TO BOOLVAL.
CHANGE ATTRIBUTE LOCKED OF MYSELF TO VALUE FALSE.
IF ATTRIBUTE SW1 OF MYSELF = VALUE FALSE
   DISPLAY "SWITCH ONE IS OFF."
```

Accessing Event Task Attributes

The event task attributes are accessed by the same types of statements that access event variables. For a discussion of statements related to events, refer to the *A Series Task Management Programming Guide*.

The following are ALGOL examples:

```
CAUSE (MYSELF.EXCEPTIONTASK.EXCEPTIONEVENT);
WAITANDRESET (MYSELF.EXCEPTIONEVENT);
```

The following are COBOL74 examples:

```
CAUSE ATTRIBUTE EXCEPTIONEVENT OF MYSELF.
WAIT AND RESET UNTIL ATTRIBUTE EXCEPTIONEVENT OF MYSELF.
```

WFL jobs cannot reference event task attributes directly. However, the following statements cause the job to implicitly wait on the exception event and the accept event:

```
WAIT; % Causes the job to wait on its own exception event.
STR := ACCEPT("ENTER A COMMAND"); % Waits on its own accept event &
                                  % stores operator AX command
                                  % input in string variable STR.
```

Accessing Integer and Real Task Attributes

In general, integer and real task attributes accept or return a numeric identifier, literal, or arithmetic expression. The system allows you to mix integer and real types: thus, you can assign a real value to an integer task attribute or read a real task attribute value into an integer variable. The system rounds off real numbers to change them into integers where necessary.

In the following WFL example, INT is an integer variable and T is a task variable:

```
CLASS = 2;
INT := T(TASKVALUE);
```

In the following ALGOL example, INT is an integer variable and T is a task variable:

```
T.TASKVALUE := 3;
INT := T.CORE;
```

In the following COBOL74 examples, INTVAL was declared as *77 INTVAL BINARY PIC 9(11)*.

```
CHANGE ATTRIBUTE TASKVALUE OF MYSELF TO 16.
MOVE ATTRIBUTE TASKVALUE OF MYSELF TO INTVAL.
```

Accessing Mnemonic Task Attributes

In WFL, mnemonic task attributes can be read into string values or compared with string values. Mnemonics can be assigned as keywords, without quotes around them. In the following examples, STR is a string variable and T is a task variable:

```
MYSELF(JOBSUMMARY = SUPPRESSED);
STR := T(HISTORYTYPE);
IF T(HISTORYTYPE) = "NORMALEOTV" THEN DISPLAY "RAN SUCCESSFULLY";
```

In ALGOL, mnemonic task attributes accept or return a numeric value. The VALUE function can be used to translate a mnemonic into a numeric value for assignment to, or comparison with, a mnemonic task attribute. In the following examples, INTVAL is an integer variable and T is a task variable:

```
MYJOB.JOBSUMMARY := VALUE(SUPPRESSED);
INTVAL := T.HISTORYTYPE;
IF T.HISTORYTYPE = VALUE(SUPPRESSED) THEN ...
```

In COBOL74, mnemonic task attributes also accept or return a numeric value and the VALUE function is also available. In the following examples, MNEMVAL was declared as *77 MNEMVAL BINARY PIC 9(11)*:

```
MOVE ATTRIBUTE JOBSUMMARY OF MYSELF TO MNEMVAL. % Returns a number
CHANGE ATTRIBUTE JOBSUMMARY OF MYSELF TO VALUE UNCONDITIONAL.
IF ATTRIBUTE JOBSUMMARY OF MYSELF = VALUE UNCONDITIONAL
    DISPLAY "JOBSUMMARY IS UNCONDITIONAL".
```

Accessing String Task Attributes

In WFL, string task attributes can be read into string variables and assigned string literals, variables, or expressions. WFL also allows some string task attributes to be assigned a nonquoted value. If a string task attribute is assigned a nonquoted value, then the nonquoted value is checked for correct syntax at compile time. If the same task attribute is assigned a string value, the contents of the string are not checked for syntax until run time.

In the following WFL examples, STR is a string variable and T is a task variable:

```
T(FAMILY DISK = DPMAST OTHERWISE DISK); % Nonquoted assignment
T(FAMILY = "DISK = DPMAST OTHERWISE DISK"); % String assignment
T(FAMILY = "GIBBERISH"); % Receives run-time error
STR := T(FAMILY); % Reading the value into a string variable
```

In ALGOL, string task attributes are treated as one-dimensional EBCDIC arrays. You can use REPLACE statements to assign values or to read string task attribute values into EBCDIC arrays. You must terminate values assigned to string task attributes with a period (.). Values returned by string task attributes are also terminated with a period. In the following examples, T is a task variable and ARR is an EBCDIC array that was declared as *EBCDIC ARRAY ARR[0:79]*:

```
REPLACE T.NAME BY "(JASMITH)OBJECT/THETA ON PACK.";
REPLACE ARR BY T.NAME;
```

Note: Note that ALGOL syntax does not *allow you to use string task attributes in the same way as string variables. For example, if STR is a string variable, the statement* STR := T.NAME *results in a syntax error.*

In COBOL74, string task attributes accept or return an alphanumeric item. The value is terminated with a period, as in ALGOL. In the following example, TASK-VAR-1 is a task variable and STRINGVAL was declared as *77 STRINGVAL PIC X(80)*:

```
CHANGE ATTRIBUTE NAME OF TASK-VAR-1 TO "OBJECT/ALGOL/TASK.".
MOVE ATTRIBUTE FILECARDS OF TASK-VAR-1 TO STRINGVAL.
```

Accessing Task-Valued Task Attributes

Task-valued task attributes can be assigned a task variable or can be used as task variables to access the task attributes of a particular process. In the following ALGOL examples, TVAR is a task variable that was previously declared:

```
MYSELF.PARTNER := TVAR;
MYSELF.EXCEPTIONTASK.TASKVALUE := 33;
```

In the following COBOL74 examples, TVAR-1 and TVAR-2 were previously declared as 77-level items with a USAGE of TASK:

```
CHANGE ATTRIBUTE EXCEPTIONTASK OF TVAR-1 TO TVAR-2.
CHANGE ATTRIBUTE PRIORITY OF ATTRIBUTE PARTNER OF MYSELF TO 65.
```

Task-valued task attributes cannot be accessed from WFL.

Accessing Task Attributes at the Bit Level

Some Boolean, integer, and real task attributes return values that are divided into bit fields with distinct meanings. Examples are the ERROR, HISTORY, LIBRARYSTATE, ORGUNIT, SOURCESTATION, and STOPPOINT task attributes.

The following are ALGOL statements that extract the values from various fields of the ERROR task attribute. In these statements, R is a real variable, ERR is a Boolean variable, and ERRNUM and UCERRNUM are real variables:

```
R := TVAR.ERROR;              % Put ERROR value in real variable
ERR := BOOLEAN(R.[46:1]);     % Translate a bit value into a Boolean
ERRNUM := R.[7:8];            % Record the task attribute number
UCERRNUM := R.[27:20];        % Record the USERDATA error number
```

The following are COBOL74 statements that extract the values from the various fields of the ERROR task attribute. The variables INTVAL, ERR, ERRNUM, and UCERRNUM were all declared as 77-level variables of type *BINARY PIC 9(11)*.

```
MOVE ATTRIBUTE ERROR OF TASK-VAR-1 TO INTVAL.
MOVE INTVAL TO ERR [ 46:00:01 ].
MOVE INTVAL TO ERRNUM [ 07:07:08 ].
MOVE INTVAL TO UCERRNUM [ 27:19:20 ].
```

In WFL, there is no direct way to access task attributes at the bit level. However, the ERROR task attribute can be accessed by mnemonic values in WFL. Further, a WFL job can extract values from selected fields of any real or integer value by calling the following ALGOL program:

```
PROCEDURE WORDANALYZER(FULLWORD, STARTPOINT, LENGTH, FIELDVAL);
   VALUE FULLWORD, STARTPOINT, LENGTH;
   REAL FULLWORD, FIELDVAL;
   INTEGER LENGTH, STARTPOINT;
BEGIN
   FIELDVAL := FULLWORD.[STARTPOINT:LENGTH];
END.
```

In the WORDANALYZER program, the FULLWORD parameter receives the real or integer value to be analyzed. The STARTPOINT parameter receives the left-most bit position of the field being evaluated. The LENGTH parameter receives the length of the field being evaluated. The FIELDVAL parameter returns the value of the specified field. Note that the calling WFL job should pass the FIELDVAL parameter by reference.

WFL does not provide access to the HISTORYREASON task attribute. The following WFL job determines the HISTORYREASON value indirectly by calling the WORDANALYZER program. WORDANALYZER extracts field [23:08] from the HISTORY task attribute value.

```
BEGIN JOB TEST/WFL;
REAL HREASON;
TASK T;
RUN OBJECT/DELTA [T];
RUN OBJECT/WORDANALYZER(T(HISTORY),23,8,HREASON REFERENCE);
IF T(HISTORYTYPE) = "DSEDV" AND T(HISTORYCAUSE) = "OPERATORCAUSEV"
   AND HREASON = 2  % Equivalent to HISTORYREASON mnemonic JUSTDSEDV
THEN ABORT "OBJECT/DELTA WAS DSED BY OPERATOR";
END JOB
```

Note: *Some programmers have attempted to use WFL expressions involving DIV and MOD operators to extract the values of fields in words. This method is not recommended, because the DIV and MOD operators interpret a number of the bits in field [47:09] as sign or exponent values. The value of these high-order bits can therefore affect the results of DIV and MOD operations.*

ALGOL, COBOL74, and WFL all provide bit-level access to the OPTION task attribute by way of special mnemonics that specify the bit position. For examples, refer to the discussion of OPTION in Section 2, "Task Attribute Descriptions."

Using WFLSUPPORT to Access Task Attributes

The WFLSUPPORT system library exports two library procedures that assist in assigning attributes to a task variable: the HANDLEATTRIBUTES procedure and the ATTRIBUTEMESSAGE procedure. The HANDLEATTRIBUTES procedure accepts a string of text containing task attribute assignments, and makes the requested assignments to a task variable. The ATTRIBUTEMESSAGE procedure accepts an encoded task attribute assignment error as input and returns a textual error message.

Assigning Task Attributes through HANDLEATTRIBUTES

The HANDLEATTRIBUTES procedure has the following primary uses, which are both illustrated by the examples later in this section.

- It can be used to pass task attributes to a compiler for insertion into an object code file. The use of HANDLEATTRIBUTES replaces the old mechanism of attaching attributes to the compiler SHEET array. This old mechanism is to be deimplemented on a future release.

- It can be used by an interactive program to allow the user to enter task attribute assignments at run time. Because HANDLEATTRIBUTES includes all the logic for checking the task attribute syntax, the interactive program need not be changed as new task attributes are implemented in the future.

The HANDLEATTRIBUTES procedure assumes that the task attribute assignments follow the syntax of a task equation list in WFL, except that local data specifications cannot be included. For the syntax of the WFL task equation list, refer to the *A Series Work Flow Language (WFL) Programming Reference Manual*.

HANDLEATTRIBUTES can handle assignments to all the task attributes that can be specified in WFL. These include assignments to the DATABASE, FILECARDS, and LIBRARY task attributes, which are known in WFL as database equations, file equations, and library equations, respectively. Like WFL, HANDLEATTRIBUTES does not handle assignments to task attributes of type event or task.

If any of the task attribute assignments contains an error, HANDLEATTRIBUTES returns without making the requested assignments. You can specify options to tell HANDLEATTRIBUTES whether to accept assignments that generate warnings.

You can use the AICOMPILEF field of the HOW1 parameter to specify whether HANDLEATTRIBUTES is to accept both compiler task equations and noncompiler task equations. Additionally, you can use the DISPOSITION parameter to specify whether the equations are to be assigned to the target task variable, assigned to the MYPPB task attribute for later use, or simply checked for syntactical correctness.

The following is an ALGOL example of the way the WFLSUPPORT library declaration and the HANDLEATTRIBUTES procedure declaration look in a calling program:

```
LIBRARY WFLSUPPORT (LIBACCESS=BYFUNCTION);

REAL PROCEDURE HANDLEATTRIBUTES
          (TEXT,TEXTOFFSET,TEXTLENGTH,HOW1,DISPOSITION,TARGET,ERRORLOC);
   REAL          TEXTOFFSET,TEXTLENGTH,HOW1,DISPOSITION,        ERRORLOC;
   EBCDIC ARRAY TEXT[*];
   TASK                                              TARGET;
   LIBRARY WFLSUPPORT;
```

Alternatively, you can use the $INCLUDE compiler option in your program to automatically insert these declarations from the file *SYMBOL/ATTRIBUTE/ INTERPRETER/INTERFACE. It is a good idea to do so because this file also contains many defines that can be used with the HANDLEATTRIBUTES procedure.

Table 1-1 explains all the HANDLEATTRIBUTES parameters. In this table, individual fields are labeled with the name of the corresponding define in *SYMBOL/ATTRIBUTE/INTERPRETER/INTERFACE.

Table 1-1. HANDLEATTRIBUTES Parameters

Parameter	Explanation
TEXT	The calling program must place the text of the task attribute assignments in this parameter. The assignments must follow the syntax of a task equation list in WFL.
TEXTOFFSET	The calling program can use this parameter to specify the offset within the TEXT buffer at which the attribute assignments begin. The offset is zero-relative and expressed in units of bytes.
TEXTLENGTH	The calling program can use this parameter to specify the number of bytes to be parsed starting at the location specified by the TEXTOFFSET parameter. If TEXTLENGTH is 0, then TEXT is scanned until a null character is encountered.
HOW1	The calling program can use this parameter to specify parsing control options. The fields of this parameter have the following meanings:
	[47:23] Reserved. The value of this field must be 0.
	[24:01] AIWARNINGSFATALF.
	If 1, and a warning or error is detected, HANDLEATTRIBUTES returns without making the requested assignments. The procedure result and the ERRLOC parameter store information about the error or warning.
	If 0, and a warning is detected, HANDLEATTRIBUTES displays a warning message and then continues normally. If an error is detected, HANDLEATTRIBUTES behaves as it would if the value of this field were 1.
	[23:23] Reserved. The value of this field must be 0.
	[00:01] AICOMPILEF.
	If AICOMPILEF is 1, compiler mode is enabled. This mode makes it possible to assign task attributes to a compilation or to the resulting object code file. Compiler task equations are those that are prefixed by the word COMPILER or the name of a compiler, such as ALGOL, PASCAL, and so on.
	If AICOMPILEF is 0, then any task attribute assignments preceded by a compiler prefix result in a syntax error.
	For further information, refer to the following discussion of the DISPOSITION parameter.
DISPOSITION	The calling program can use this parameter to specify whether the task attribute assignments are to be applied. The effect of the DISPOSITION parameter varies, depending on whether compiler mode is specified by the AICOMPILEF field of the HOW1 parameter.

Table 1–1. HANDLEATTRIBUTES Parameters (cont.)

Parameter	Explanation
	The values of this parameter have the following meanings:

<table>
<tr><td>0</td><td>AIATTACHV.</td></tr>
</table>

If HANDLEATTRIBUTES is invoked in compiler mode, then

- Compiler task equations are assigned to TARGET.MYPPB. The compiler task equations in TARGET.MYPPB are applied to TARGET when TARGET.APPLYLIST is set to TRUE, or when TARGET is used to initiate a process (whether the process is a compiler or not).

- Noncompiler task equations are assigned as a nested MYPPB value within TARGET.MYPPB. If TARGET is later used to initiate a compiler process, the compiler process reads the nested task equations from TARGET.MYPPB and assigns them to the resulting object code file.

If HANDLEATTRIBUTES is invoked in noncompiler mode, then

- Any compiler task equations receive an error.

- Noncompiler task equations are assigned to TARGET.MYPPB. The equations in TARGET.MYPPB are applied when TARGET.APPLYLIST is set to TRUE or when TARGET is used in a process initiation statement.

<table>
<tr><td>1</td><td>AIAPPLYV.</td></tr>
</table>

If HANDLEATTRIBUTES is invoked in compiler mode, then

- Compiler task equations are applied directly to TARGET.

- Noncompiler task equations are assigned to TARGET.MYPPB. If TARGET is later used to initiate a compiler, the compiler applies the equations in TARGET.MYPPB to the resulting object code file. Note that the system never applies the TARGET.MYPPB equations to TARGET, not even if TARGET.APPLYLIST is set to TRUE or TARGET is used in a process initiation statement.

If HANDLEATTRIBUTES is invoked in noncompiler mode, then

- Any compiler task equations receive an error.

- Noncompiler task equations are applied directly to TARGET. Nothing is written to TARGET.MYPPB.

Table 1–1. **HANDLEATTRIBUTES Parameters** (cont.)

Parameter	Explanation
	2 AISYNTAXONLYV.
	If HANDLEATTRIBUTES is invoked in compiler mode, then both compiler and noncompiler task equations are checked for syntax. None of the equations are applied and nothing is written to TARGET.MYPPB.
	If HANDLEATTRIBUTES is invoked in noncompiler mode, then any compiler task equations receive an error. Noncompiler task equations are checked for syntax. None of the equations are applied and nothing is written to TARGET.MYPPB.
TARGET	The calling program can use this parameter to provide the task variable to which the task attribute assignments are applied. This parameter is ignored if the value of the DISPOSITION parameter is AISYNTAXONLYV.
ERRORLOC	HANDLEATTRIBUTES uses this parameter to return error information to the calling program. If field AIERRORF of the procedure result is 0, meaning that no error occurred, then this parameter stores a 0. If field AIERRORF has a value of 1, then ERRORLOC returns a value divided into the following fields:
	[47:20] This field is always 0.
	[27:01] AIERROFFSETVALIDF.
	If 1, then the error is associated with a particular offset in the input TEXT array. This offset is given in field AIERROFFSETF of the ERRORLOC parameter.
	If 0, then the error is not associated with a specific offset. In this case the AIERROFFSETF field does not store any offset.
	[26:01] This field is always 0.
	[25:18] AIERROFFSETF. This field stores the zero-relative offset of the task attribute error in the TEXT parameter. This field is meaningful only if the AIERROFFSETVALIDF field of the ERRORLOC parameter stores a 1.
	[07:08] This field is always 0.

Table 1–1. HANDLEATTRIBUTES Parameters (cont.)

Parameter	Explanation
Procedure Result	The HANDLEATTRIBUTES result contains general information about task attribute errors. The value is divided into the following fields:
	[47:08] AITYPEF. The type of attribute for which the error was detected. The possible values are as follows:
	1 = FILECARDS task attribute
	2 = Miscellaneous task attribute
	4 = PRINTDEFAULTS task attribute
	6 = LIBRARY task attribute
	7 = DATABASE task attribute
	[39:16] AIATTNUMF. The number of the attribute for which the error was detected. If field [47:08] of the procedure result indicates that FILECARDS had an error, then AIATTNUMF stores the number of the file attribute that caused the error. Otherwise, AIATTNUMF stores the number of the task attribute that caused the error. For file attribute numbers, refer to the *A Series File Attributes Programming Reference Manual*. For task attribute numbers, refer to the discussion of the ERROR task attribute in Section 2, "Task Attribute Descriptions."
	[23:16] AIERRNUMF. The error or exception number. Refer to Table 1–2, "HANDLEATTRIBUTES Error Numbers," for a list of these numbers and their meanings. This information also appears in the file *SYMBOL/ATTRIBUTE/INTERPRETER/INTERFACE.
	[07:05] This field is always 0.
	[02:01] AIATTNUMVALIDF. If 1, then the error is associated with a particular attribute in the input TEXT. In this case, the AIATTNUMF field and the AITYPEF field are valid. If 0, the error is not associated with a particular attribute and these fields are not valid.
	[01:01] AIWARNINGF. If 0, the error is fatal. If 1, it is a warning.
	[00:01] AIERRORF. If 1, an error occurred. If 0, no error occurred and none of the other fields in this result are valid.

Table 1–2 explains the values of the numbers returned in field [23:16] of the HANDLEATTRIBUTES procedure result.

Table 1–2. HANDLEATTRIBUTES Error Numbers

Error Number	Meaning
0	No error or warning occurred.
1-1000	If the error number is in this range, the error number is equal to the HISTORYREASON task attribute value. For information about the value, refer to the description of HISTORYREASON in Section 2, "Task Attribute Descriptions."
1015	A syntax error was detected.
1017	An attribute mnemonic was expected.
1018	A numeric value was expected.
1019	An end-of-text marker was encountered.
1021	The same attribute has been assigned two values. This is a warning in most cases, but it is an error for the PRINTDEFAULTS attribute. If the warning is ignored, the more recent value overwrites the previous value.
1023	A right parenthesis was expected.
1027	A semicolon was expected (:).
1030	A string over 256 characters long was specified.
1031	An ending quotation mark (") is missing.
1032	The maximum number was exceeded.
1033	An illegal character was used.
1034	An illegal file name was used.
1035	An OPTION task attribute mnemonic was expected.
1036	An attribute mnemonic was expected.
1037	An illegal attribute mnemonic was used.
1038	A left parenthesis was expected.
1039	A real constant was expected.
1040	The user part of a file title must be 12 names or less.
1041	The end of the statement was expected.
1042	A task attribute was expected.
1043	A compiler name was expected.
1044	An equal sign (=) was expected.
1045	A simple volume name was expected.

Table 1–2. HANDLEATTRIBUTES Error Numbers (cont.)

Error Number	Meaning
1046	A keyword was not recognized.
1047	An attempt was made to assign a value to a read-only attribute.
1048	Too many serial numbers were specified.
1049	The serial number was too long.
1050	A serial number was expected.
1051	The serial number contained an illegal character.
1052	This construct can be used only in a job heading.
1053	An illegal resource value was specified.
1054	A number from 0 to 255 was expected.
1055	This attribute is not valid in this context.
1056	A comma (,) was expected.
1057	The word OTHERWISE or ONLY was expected.
1058	A WFLSUPPORT fault occurred.
1059	String constants are not allowed here.
1060	This construct is not implemented.
1061	There was an error in numeric constant evaluation.
1062	The DATABASE attribute was expected.
1063	An illegal name was specified.
1064	A hyphen (-) or underscore (_) cannot be the first character of an unquoted name.
1065	The family specification was invalid.
1066	A file attribute was expected.
1067	A print attribute or print modifier was expected.
1068	A file equation for this file was previously specified; the previous equation is ignored.
1072	An invalid type was specified.
1073	An invalid INTNAME file attribute value was specified.
1074	The word UP was expected.
1075	The word FILE was expected.

Decoding Error Values with ATTRIBUTEMESSAGE

The ATTRIBUTEMESSAGE procedure translates the HANDLEATTRIBUTES procedure result into a textual error message, suitable for display to a user.

ATTRIBUTEMESSAGE also allows you to specify the language in which the error message should be displayed, an array to hold the error message, and the offset in the array where the error message should start. ATTRIBUTEMESSAGE places the error message at the requested location in the array, and updates the offset parameter to point to the end of the error message.

The following is an ALGOL example of the way the WFLSUPPORT library declaration and the ATTRIBUTEMESSAGE procedure declaration look in a calling program:

```
LIBRARY WFLSUPPORT (LIBACCESS=BYFUNCTION);

REAL PROCEDURE ATTRIBUTEMESSAGE
                    (ERRINFO,HOW4,LANGUAGE,LANGLENGTH,MSG,MSGOFFSET);
    REAL                ERRINFO,HOW4,         LANGLENGTH,     MSGOFFSET;
    EBCDIC ARRAY                  LANGUAGE[*],          MSG[*];
    LIBRARY WFLSUPPORT;
```

Alternatively, you can use the $INCLUDE compiler option in your program to automatically insert these declarations from the file *SYMBOL/ATTRIBUTE/ INTERPRETER/INTERFACE. It is a good idea to do so because this file also contains many defines that can be used with the ATTRIBUTEMESSAGE procedure.

Table 1-3 explains all the ATTRIBUTEMESSAGE parameters. In this table, individual fields are labeled with the name of the corresponding define in *SYMBOL/ATTRIBUTE/INTERPRETER/INTERFACE.

Table 1-3. ATTRIBUTEMESSAGE Parameters

Parameter	Explanation	
ERRINFO	The calling program must store the encoded error description in this parameter. The format of this word must be the same as the procedure result returned by HANDLEATTRIBUTES.	
HOW4	The calling program can use this parameter to specify some aspects of the ATTRIBUTEMESSAGE interface. This parameter is divided into the following values:	
	[47:04]	AIMESSAGEVERSIONF. This field stores the version number of the ATTRIBUTEMESSAGE interface. For the version supplied with this release, the value should be 0.
	[43:43]	Reserved. The value of this field must be 0.

Table 1–3. ATTRIBUTEMESSAGE Parameters (cont.)

Parameter	Explanation
	[00:01] AIDISPLAYMESSAGEF.
	If 1, ATTRIBUTEMESSAGE issues a DISPLAY statement that causes the resulting error message to appear in the MSG (Display Messages) system command display. (The DISPLAYONLYTOMCS task attribute can limit the display of the message.)
	If 0, the error message does not appear in the MSG display.
LANGUAGE	The calling program can use this parameter to specify the language in which the error message is to be reported. Parsing of the language starts at element 0 of the LANGUAGE value, although leading blanks are ignored. Parsing ceases when a null character is encountered or when the number of characters specified by the LANGLENGTH parameter have been parsed.
	If the requested language is not supported on the system, a warning of AILANGNOTAVAILABLEV is reported and the system default language is used.
LANGLENGTH	The calling program can use this parameter to specify the maximum number of characters in the LANGUAGE parameter to be parsed, starting at element 0 of the LANGUAGE value. If LANGLENGTH is 0, the LANGUAGE parameter is ignored and the LANGUAGE task attribute of the calling process is used instead.
MSG	ATTRIBUTEMESSAGE returns the decoded error message in this parameter. You should take care that the array passed to this parameter is at least as long as the sum of the initial MSGOFFSET value and the value of the AIMSGLENGTHV define in *SYMBOL/ATTRIBUTE/INTERPRETER/INTERFACE. (The AIMSGLENGTHV define specifies the maximum length message that can be returned by the current version of ATTRIBUTEMESSAGE.)
MSGOFFSET	The calling program can use this parameter to specify the offset within the MSG array at which the decoded message should begin. ATTRIBUTEMESSAGE updates this parameter to return the offset of the null character that terminates the decoded message.
Procedure Result	ATTRIBUTEMESSAGE uses this parameter to report errors. The procedure result has the same format as the HANDLEATTRIBUTES procedure result, as previously described in this section.

The format of the message returned in MSG, the message parameter, is as follows when the error pertains to a specific attribute (that is, when field AIATTNUMVALIDF of the ERRINFO parameter equals 1):

```
<attribute type> Attribute "<attribute name>": <error description>
```

For example:

```
Task Attribute "DECLAREDPRIORITY": Cannot recognize keyword
```

If the error does not pertain to a specific attribute (that is, the AIATTNUMVALIDF field of the ERRINFO parameter equals 0), the message has the following format:

```
Attribute Error: <error description>
```

Examples

The following are examples of ALGOL programs that use the HANDLEATTRIBUTES and ATTRIBUTEMESSAGE procedures.

Example 1: Setting Multiple Attributes.

The following interactive program asks a user to supply task attribute assignments. The program then calls HANDLEATTRIBUTES to check the assignments for correctness and apply them to a task variable. If there are no errors, the program uses the task variable to initiate a task. If there are errors, the program uses ATTRIBUTEMESSAGE to display an error message.

```
100 BEGIN
110 $INCLUDE ATTINT = "*SYMBOL/ATTRIBUTE/INTERPRETER/INTERFACE"
120 TASK T;
.130 FILE TERM(KIND = REMOTE,FILEUSE=IO);
140 EBCDIC ARRAY TEXT[Ø:419], LANG[Ø:119], MYPPBVAL[Ø:599];
150 REAL ERRORLOC, ATTERR, MSGERR, ERROFFSET, HOW1;
160 PROCEDURE UTILRUN;
170    EXTERNAL;
180
190 WRITE(TERM,//,"PLEASE ENTER YOUR TASK EQUATIONS");
200 REPLACE TEXT BY 48"ØØ" FOR 420;
210 READ(TERM,420,TEXT);
220 ATTERR := HANDLEATTRIBUTES(TEXT,Ø,Ø,HOW1,AIAPPLYV,T,ERRORLOC);
230 IF ATTERR = Ø THEN
240    CALL UTILRUN [T]
250 ELSE
260    BEGIN
270    DISPLAY(TEXT);
280    IF ERRORLOC.AIERROFFSETVALIDF = 1 THEN
290       BEGIN
300       REPLACE TEXT BY " " FOR ERRORLOC.AIERROFFSETF,
310                      "^", 48"ØØ";
320       DISPLAY(TEXT);
330       END;
340    REPLACE TEXT BY "*" FOR 3;
350    ERROFFSET := 3;
360    REPLACE LANG BY T.LANGUAGE;
370    MSGERR := ATTRIBUTEMESSAGE(ATTERR,1,LANG,Ø,TEXT,ERROFFSET);
380    END;
390 END.
```

The following is an example of the interaction between a user and this program. The user runs the program from a CANDE session. Because the user misspells the

TASKVALUE task attribute, the program returns an error message and does not initiate the requested task.

```
User:     RUN ATTINT/TEST
Response: #RUNNING 9807
Response: #?
Response: PLEASE ENTER YOUR TASK EQUATIONS
User:     NAME=OBJECT/ALGOL/TASK;TASKVALUW=R;PRIORITY=60;
Response: #9807 DISPLAY:NAME=OBJECT/ALGOL/TASK;TASKVALUW=R;PRIORITY=60;
Response:              .
Response: #9807 DISPLAY:                         ^.
Response: #9807 DISPLAY:***Attribute error: Task attribute expected.
Response: #ET=27.6 PT=0.1 IO=0.1
```

Example 2: Inserting Attributes into an Object Code File

Programs that initiate a compiler can cause attributes to be inserted into the resulting object code file. These attributes are applied at task initiation time whenever the object code file is executed. The following example shows how this is done using the HANDLEATTRIBUTES and ATTRIBUTEMESSAGE procedures.

```
100 BEGIN
110 $INCLUDE ATTINT="*SYMBOL/ATTRIBUTE/INTERPRETER/INTERFACE."
120 TASK CTASK;
130 ARRAY SHEET[0:32];
140 EBCDIC ARRAY TEXT[0:299];
150 REAL ERRLOC, ATTERR, MSGERR, MSGOFFSET;
160
170 PROCEDURE ALGOLCOMPILER(SHEET);
180    ARRAY SHEET[*];
190    EXTERNAL;
200
210 REPLACE TEXT BY
220    "ALGOL NAME=*SYSTEM/ALGOL ON DISK;"
230    "ALGOL FILE CARD (KIND=DISK, TITLE=ALGOL/TASK);"
240    "ALGOL FILE CODE (KIND=DISK, TITLE=OBJECT/ALGOL/TASK);"
250    "MAXPROCTIME=20;TASKVALUE=3;"
260    "FILE IN=DAILY/DATA; FILE OUT(KIND=DISK,TITLE=OUTPUT)" 48"00";
270 ATTERR:=HANDLEATTRIBUTES(TEXT,0,0,1,AIAPPLYV,CTASK,ERRLOC);
280
290 REPLACE SHEET BY 0 FOR 33 WORDS;
300 SHEET[8] := VALUE(LIBRARY); % This statement specifies the
310                             % object code file disposition.
320 SHEET[0] := 0 & 1[47:1];
330
340 CALL ALGOLCOMPILER(SHEET) [CTASK];
350
360 END.
```

In this example, the task assignments at lines 220 through 240 are applied to the compilation, because they are preceded by the keyword ALGOL. The task assignments

at lines 250 through 260 are assigned to the resulting object code file, because they have no compiler name prefixing them.

System Administrator Access to Task Attributes

The system administrator can establish defaults and limits on the use of various task attributes by various users. These defaults and limits aid in preserving system security and managing workload.

Assigning Task Attributes to Usercodes

The system administrator can create usercode definitions in the USERDATAFILE by running either MAKEUSER or a DCALGOL program that calls the USERDATA function. By creating a usercode definition, the system administrator makes that usercode a legal value for the USERCODE task attribute. By suspending or removing the usercode definition, the system administrator can prevent processes from being initiated with that USERCODE task attribute value.

The usercode definition can include one or more usercode attributes. Several of these usercode attributes provide values that can be inherited by task attributes of processes that run with that usercode. The following task attributes can be affected by the values of usercode attributes: ACCESSCODE, CHARGE, CLASS, CONVENTION, DEPTASKACCOUNTING, DESTNAME, FAMILY, FILEACCOUNTING, LANGUAGE, PRINTDEFAULTS, PRIORITY, SAVEMEMORYLIMIT, and TEMPFILELIMIT. These can be referred to as the usercode-related task attributes of a process.

These task attributes are not always affected by their corresponding usercode attributes. The system administrator might not have included all the possible usercode attributes in the usercode definition. Furthermore, the usercode attributes are inherited only in the following circumstances:

- Usercode attributes can be inherited by a WFL job that includes a USERCODE assignment in the job attribute list. Any usercode-related task attributes that are not assigned values in the job attribute list receive their values from the usercode attributes.

- Usercode attribute values are inherited by CANDE or MARC session attributes at log-on time. These session attributes are inherited by any processes initiated from the session, unless the user takes actions to change the session attributes or uses task equations to assign different task attributes to a process. (Refer to the discussion of tasking from interactive sources in the *A Series Task Management Programming Guide*.)

The usercode-related task attributes are also inherited from a parent by its offspring, unless specifically overridden. In this way, a usercode attribute can be propagated through an entire process family.

For further details about the inheritance rules for usercode-related task attributes, refer to the descriptions of each of these task attributes in Section 2, "Task Attribute Descriptions."

Assigning Job Queue Attributes

The system administrator can use job queue definitions to affect the task attributes of WFL jobs and their descendants. The job queue definitions are created by way of the MQ (Make or Modify Queue) system command. Each job queue definition can include job queue attributes that specify default or limiting values for task attributes of jobs run from that queue. The job queue attributes correspond mostly to task attributes that impose limits on resource usage, such as MAXPROCTIME and MAXIOTIME. For a summary of the effects of job queue attributes on task attributes, refer to the *A Series Task Management Programming Guide*.

System Access to Task Attributes

The system software plays several roles in the assignment of task attribute values. The system provides values for task attributes in some cases, resolves conflicting assignments from various sources, and issues errors when an attempt is made to access an attribute incorrectly.

Making Automatic Assignments

The system software provides values for task attributes that have not been specifically assigned values by any of the other methods discussed in this section. The following subsections discuss the types of assignments that the system software makes.

Providing Default Values

The default value for a task attribute is the value it assumes if no other factors influence the task attribute value. For Boolean task attributes, the default is typically FALSE; for integer or real task attributes, zero; for string task attributes, a null string.

The default values for all the task attributes are documented in Section 2, "Task Attribute Descriptions."

Providing Inherited Values

Inheritance is the transfer of a task attribute value from a process to one of its descendants. Different inheritance rules are applied to different task attributes; some can inherit values, but others cannot. The inheritance rules for each task attribute are included in the task attribute descriptions in Section 2, "Task Attribute Descriptions."

Some of the basic task attributes that can be inherited are USERCODE, ACCESSCODE, CHARGE, and FAMILY. The inheritance properties save the programmer the trouble of having to assign these task attributes for each member of the process family. A single assignment to the job can be propagated to all its descendants.

The term *inheritance* is also loosely applied to the transfer of values from job queue attributes, session attributes, or usercode attributes to a process. These types of

inheritance are discussed under "Assigning Job Queue Attributes," "Assigning Task Attributes to a Session," and "Assigning Task Attributes to Usercodes" in this section.

Updating Task Attribute Values

During process execution, the system automatically updates the values of certain task attributes. These task attributes return information about dynamic aspects of process status and history. One example is the STATUS task attribute, whose value is updated when the process becomes scheduled, suspended, resumed, or terminated. Other examples are the task attributes that record resource usage, including ACCUMPROCTIME and ACCUMIOTIME. These automatic updates make it possible to use these task attributes to monitor the current state of a process as it executes.

Resolving Conflicting Values

When a process is initiated, the system software evaluates the task attribute values submitted from the various sources discussed in this section. Where different sources have assigned conflicting values to the same task attribute, the system chooses the value submitted from the most dominant source.

The rules used to determine which assignment is most dominant are called *overwrite rules*. The system applies different overwrite rules to different task attributes. However, most task attributes follow either *standard* overwrite rules or *object code file dominant* overwrite rules. The following subsections describe standard and object code file dominant overwrite rules for various types of processes.

Section 2, "Task Attribute Descriptions," includes information about the overwrite rules for each task attribute. That section states whether each task attribute follows standard or object code file dominant overwrite rules. For task attributes that follow irregular rules, the exact behavior of the task attributes is explained.

Overwrite Rules for WFL Jobs

The following are the various sources that can contribute to the initial task attribute values of a WFL job. The sources are listed in order from most dominant to least dominant, according to standard overwrite rules:

1. Assignments in the job attribute list of the WFL job.

2. Usercode attributes, if a USERCODE assignment is included in the job attribute list of the WFL job.

3. Attributes of the CANDE or MARC session, if the WFL job was initiated from a session.

4. Job queue defaults. (By contrast, job queue limits do not affect the initial task attribute values of a WFL job. They simply affect the selection of a queue for the job.)

5. The task attribute default.

Task attributes cannot be assigned to the object code file of a WFL job because a WFL job has no object code file. Object code file dominant task attributes, when applied to a WFL job, follow the standard overwrite rules listed previously.

The following is one illustration of the overwrite rules for WFL jobs. Suppose the job attribute list of a certain WFL job includes a PRINTDEFAULTS assignment, followed by a USERCODE assignment. Further, suppose that the usercode definition in the USERDATAFILE has a PRINTDEFAULTS value associated with it. In this case, only the PRINTDEFAULTS value specified in the job attribute list is used, even though the USERCODE assignment statement occurred last.

Overwrite Rules for Session Tasks

The following are the various sources that can contribute to the initial task attribute values of a task initiated from a CANDE or MARC session. The sources are listed in order from most dominant to least dominant, according to standard overwrite rules:

1. Task equations appended to the initiation statement
2. Inheritance from the attributes of CANDE or MARC sessions
3. Assignments to the object code file
4. The task attribute default

For an object code file dominant task attribute, the order of dominance is the same, except that item 3, assignments to the object code file, is moved to the head of the list.

Overwrite Rules for Other Processes

The following are the various sources that can contribute to the initial task attribute values of a process initiated from a WFL, ALGOL, or COBOL74 process. The sources are listed in order from the most dominant to the least dominant, according to standard overwrite rules:

1. Task equations appended to the initiation statement
2. Task attribute assignments to the task variable outside the task variable declaration
3. Task attribute assignments in the task variable declaration
4. Assignments to the object code file
5. Inheritance from the parent
6. The task attribute default

For an object code file dominant task attribute, the order of dominance is the same, except that item 4, assignments to the object code file, is moved to the head of the list.

Task Attribute Errors

Task attribute errors result from an attempt to access a task attribute in an improper manner. The most basic errors are caught at compile time. These include type

mismatches that occur, for example, from assigning a string to an integer-valued task attribute.

Other task attribute errors are caught only at run time. For example, a run-time error can result from assigning a task attribute a value that is

- Outside the allowed range. For example, if a particular attribute has a range of 1 to 9999, then an assignment of 10500 might cause an error.

- Assigned at the wrong time. Some attributes can be assigned only before initiation; after initiation, assignment causes a run-time error.

- Referring to a nonexistent entity. For example, an error results from assigning a DESTSTATION value that does not correspond to a valid Logical Station Number (LSN).

- Inconsistent with a related attribute. For example, the USERCODE and CHARGE task attributes must be compatible.

An attempt to read a task attribute can also result in an error in some cases. For example, if the private process bit of the OPTION task attribute is set, then other processes are prevented from reading (or assigning) the task attributes of this process.

Some task attributes can cause a delayed error if assigned an invalid value. For example, the STATION task attribute can be assigned a value that refers to a nonexistent station. No error occurs until the process attempts to open a remote file.

The process that attempted to access the task attribute can be referred to as the *accessing process*. The process whose task attribute was accessed can be referred to as the *receiving process*. The accessing process and the receiving process can be the same, for example, if the MYSELF task variable is used.

If the attempted access is illegal, it is the accessing process that incurs the error. If the accessing process is nonprivileged, almost all task attribute errors are fatal. If the accessing process is privileged or a message control system (MCS), then errors in accessing event-valued or file-valued task attributes are generally fatal, but most other task attribute errors are not fatal.

The ERROR task attribute of the receiving process stores the attribute number of the task attribute that was being accessed when the error occurred. The accessing process can read the ERROR task attribute of the receiving process to determine whether the last task attribute access was successful. The system erases the ERROR value each time it is read. TASKERROR is another task attribute that provides error information. Unlike ERROR, the TASKERROR value is not erased when it is read.

For further details about these task attributes, refer to "ERROR" and "TASKERROR" in Section 2, "Task Attribute Descriptions."

The operator or the user is informed of task attribute errors by the display of error messages for the process. Many task attribute error messages are documented in this manual in the "Run-Time Errors" part of many task attribute descriptions. All the errors documented in this manual are also included in the index for easy reference.

The error messages that are displayed for a process are somewhat more informative if the object code file of the process was compiled with the LINEINFO compiler option set. This option causes the sequence number of each record in the source program to be stored in the object code file. When an error occurs, the sequence number of the statement that incurred the error is included at the end of the error message.

If LINEINFO was not set, then error messages display the code address instead of the sequence number of the statement that incurred the error. You can interpret the code address by referring to the printout produced by the compiler if the LIST compiler option was set. For an example of this printout, refer to the discussion of process history in the *A Series Task Management Programming Guide*.

Section 2
Task Attribute Descriptions

The A Series task attributes provide a wide variety of options for process monitoring and control. Using task attributes, you can control various aspects of file usage, memory usage, resource usage, and communication with other processes or with operators. You can also use task attributes to determine the current status of a process or discover how it terminated.

This section includes complete descriptions of all the task attributes that are supported for customer use as of the current Mark release. Note that the file SYMBOL/ATTABLEGEN, which lists all the task attributes, includes several that are not documented in this section. These undocumented task attributes are intended only for internal use. Attempts by customers to use these task attributes result in compile-time errors, run-time errors, or other undefined results.

Choosing the Right Task Attribute

At this time, about a hundred task attributes have been implemented on A Series systems. Each task attribute is designed to assume reasonable default or inherited values. Therefore, it is not necessary for you to learn the functions of all the task attributes. However, by studying the task attributes related to a particular area of process control, you can learn how to take advantage of the abilities the system provides in that area.

Table 2–1, "Task Attribute Functional Groupings," can help you find the task attributes that are relevant to each aspect of process control. For details about any of these task attributes, refer to the individual descriptions in this section.

Table 2–1. Task Attribute Functional Groupings

Category	Attribute
Billing	CHARGE
	USERCODE
Databases	DATABASE
	MAXWAIT

Table 2-1. Task Attribute Functional Groupings (cont.)

Category	Attribute
Data comm	AUTOSWITCHTOMARC
	DESTNAME
	DESTSTATION
	DISPLAYONLYTOMCS
	INHERITMCSSTATUS
	LANGUAGE
	MCSNAME
	ORGUNIT
	SOURCEKIND
	SOURCENAME
	SOURCESTATION
	STATION
	SUPPRESSWARNING
	TANKING
	TASKWARNINGS
Debugging	OPTION
	TADS
	TASKFILE
Files	AUTORESTORE
	FAMILY
	FILEACCESSRULE
	FILECARDS
	OPTION (the AUTORM option)

Table 2–1. Task Attribute Functional Groupings (cont.)

Category	Attribute
History	DEPTASKACCOUNTING
	ERROR
	FILEACCOUNTING
	HISTORY
	HISTORYCAUSE
	HISTORYREASON
	HISTORYTYPE
	HSPARAMSIZE
	OPTION
	STACKHISTORY
	STATUS
	STOPPOINT
	SUPPRESSWARNING
	TASKERROR
	TASKWARNINGS
Identification	JOBNUMBER
	MIXNUMBER
	NAME

Table 2-1. **Task Attribute Functional Groupings** (cont.)

Category	Attribute
Interprocess Communication	ACCEPTEVENT
	EXCEPTIONEVENT
	EXCEPTIONTASK
	LOCKED
	OPTION (the "private process" option)
	PARTNER
	PARTNEREXISTS
	STATUS
	SW1 through SW8
	TARGET
	TASKLIMIT
	TASKSTRING
	TASKVALUE
	TYPE
Job Summaries	JOBSUMMARY
	JOBSUMMARYTITLE
	NOJOBSUMMARYIO
	OPTION (the NOSUMMARY option)
Libraries	LIBRARY
	LIBRARYSTATE
	LIBRARYUSERS
	STATUS
Localization	CONVENTION
	LANGUAGE
Logging	DEPTASKACCOUNTING
	FILEACCOUNTING

Table 2–1. Task Attribute Functional Groupings (cont.)

Category	Attribute
Messages	DISPLAYONLYTOMCS
	LANGUAGE
	SUPPRESSWARNING
	TASKWARNINGS
Memory Management	CORE
	SAVEMEMORYLIMIT
	STACKLIMIT
	STACKSIZE
Printer Output	BACKUPFAMILY
	BDNAME
	DESTNAME
	DESTSTATION
	OPTION (the BACKUP, BDBASE, and NOSUMMARY options)
	PRINTDEFAULTS
	TASKFILE
Remote Tasking	HOSTNAME
	ITINERARY
Resource Usage Data	ACCUMIOTIME
	ACCUMPROCTIME
	ELAPSEDTIME
	INITPBITCOUNT
	INITPBITTIME
	OTHERPBITCOUNT
	OTHERPBITTIME
	TEMPFILEMBYTES

Table 2–1. Task Attribute Functional Groupings (cont.)

Category	Attribute
Resource Usage Limits	DISKLIMIT
	ELAPSEDLIMIT
	MAXCARDS
	MAXIOTIME
	MAXLINES
	MAXPROCTIME
	MAXWAIT
	PRIORITY
	RESOURCE
	SAVEMEMORYLIMIT
	STACKLIMIT
	TASKLIMIT
	TEMPFILELIMIT
	WAITLIMIT
Restarting Processes	BRCLASS
	CHECKPOINTABLE
	RESTART
	RESTARTED
Security	ACCESSCODE
	FILEACCESSRULE
	INHERITMCSSTATUS
	USERCODE
Task Attribute Usage	APPLYLIST
	MYPPB
	ERROR
	TASKERROR

Table 2–1. Task Attribute Functional Groupings (cont.)

Category	Attribute
WFL Jobs	CLASS
	DECKGROUPNO
	FETCH
	STARTTIME

Format of the Descriptions

Each task attribute description in this section includes information about certain characteristics of task attributes. The following subsections explain how these characteristics are presented in the task attribute descriptions.

Name

Each task attribute description begins with a heading that gives the name of the task attribute. A task attribute is generally referred to by the same name from all the sources that can access that attribute. The only exception to this rule is that several task attributes have synonyms and some sources recognize only the synonym. Refer to the "Synonym" discussion in this section.

Type

The "Type" part of the description indicates the type of data that is stored in the task attribute. Almost all task attributes fall into one of the following types: Boolean, event, file, integer, mnemonic, real, string, or task. A few other task attributes, such as OPTION and RESOURCE, are of irregular types. For details about how these types are accessed from the various languages, refer to "Programmer Access to Task Attributes" in Section 1, "Accessing Task Attributes."

Units

The "Units" part of the description specifies, for either a real or integer task attribute, the units that are measured by the task attribute value: seconds, microseconds, words, and so on.

Range

The "Range" part of the description defines the legal values for the attribute. For example, for a mnemonic attribute, the range consists of all the valid mnemonic values

for the attribute. For an integer attribute, the range defines the upper and lower limits of the integers allowed (for example, 1 through 256).

In some cases, the range is defined by a metatoken. A metatoken is a word or phrase that is enclosed within angle brackets, as in the following example: < simple name > . For such items, a railroad diagram is used to describe the exact range of the task attribute values. If a metatoken is referred to by only one task attribute description, then the railroad diagram for that metatoken is given in the description of that task attribute. Metatokens that are referred to by more than one task attribute description are defined in the following syntax discussion. For information about how to read these diagrams, refer to Appendix A, "Understanding Railroad Diagrams."

Syntax

<digit>

Any one of the 10 Arabic numerals 0 through 9.

<family name>

```
—<nonquote identifier>————————————————————————————|
```

<hyphen>

The hyphen character (-).

<identifier>

```
——┬—<nonquote identifier>—————————————————————┬——————————|
   └— " ─┬─/17\-<nonquote EBCDIC character>—┬— " ─┘
         └←─────────────────────────────────┘
```

<name>

```
   ┌←─────── / ───────┐
—┬—/12\-<simple name>─┴——————————————————————————|
```

<nonquote EBCDIC character>

Any uppercase or lowercase letter, number, or special character except a quotation mark (").

<nonquote identifier>

```
   ┌←──────────────────────────┐
—┴—/17\—┬—<uppercase letter>—┬—┴——————————————————|
        └—<digit>————————————┘
```

<password>

```
—<nonquote identifier>————————————————————————————|
```

<simple name>

```
──┬─┬─<uppercase letter>─┬──┬──────────────────────────────────────────┬───┤
    └─<digit>────────────┘  │ ┌←────────────────────────────────┐       │
                            └─/16\─┬─<uppercase letter>─┬────────┘
                                   ├─<digit>────────────┤
                                   ├─<hyphen>───────────┤
                                   └─<underscore>───────┘
```

<title>

```
                            ┌←────────── / ────────┐
──┬──────────────────────┬──/12\─<simple name>─────┴──────────────────────→
  ├─ ( ─<usercode>─ ) ─┤
  └─ * ────────────────┘
```

```
→─┬──────────────────────────────────────────────────────────────────────┤
  └─ ON ─<family name>─┘
```

<underscore>
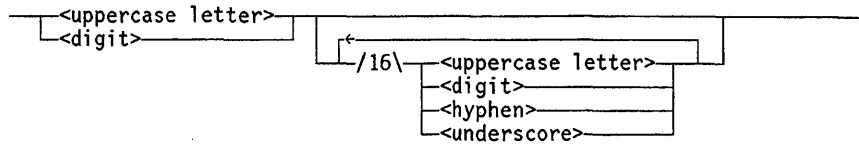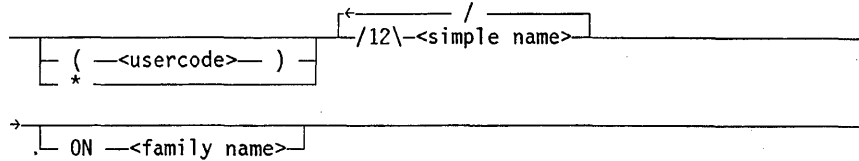
The underscore character (_).

<uppercase letter>

Any one of the 26 uppercase characters A through Z.

<usercode>

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

Default

The "Default" part of the description lists the value that the task attribute assumes if the attribute does not inherit its value and is not assigned a value. For a read-only task attribute, this is the value the task attribute returns if interrogated before initiation.

For many string task attributes, the default listed is *null string*. If this default value is read from Work Flow Language (WFL), a string of zero length ("") is returned. However, if this default value is read from ALGOL or COBOL74, a string that contains a single period (".") is returned.

Read Time

The "Read Time" part of the description defines whether and when the task attribute value can be interrogated by a program. The following are the possible read time values:

- Anytime. The task attribute of the task variable can be read before the process is initiated, while it is running, or after termination.

- Anytime; accurate after initiation. The task attribute can be read at any time, but does not receive its actual value until the process is initiated.

- Anytime; accurate while in use. The task attribute can be read at any time, but is reset to its default when the process terminates.

- Never. The task attribute cannot be read. Such an attribute is called write-only.

- Only while in use. The task attribute can be read only for an in-use process. That is, the task attribute cannot be read before the process is initiated or after it is terminated.

Note that inheritance, object code file assignments, and run-time assignments can cause the values of many attributes to change at initiation time. Therefore, any value that is read before initiation might not reflect the value that the process actually receives.

Write Time

The "Write Time" part of the description defines whether and when the task attribute can be assigned a value by a program. The following are the possible write time values:

- Anytime. The task attribute of the task variable can be assigned before the process is initiated, while it is running, or after termination.

- Anytime; effective before initiation. The task attribute value can be assigned at any time without incurring an error. However, assignments made after initiation are ignored.

- Before initiation. The task attribute must be assigned in one of the following ways:

  - An assignment to the task variable before the process is initiated.

  - A task equation appended to the statement that initiates the process.

  - An assignment in the job attribute list of a WFL job. The job attribute list immediately follows the job heading at the start of the job. These assignments are applied before the job begins execution.

  - An assignment to the object code file of the process. Such assignments can be appended to the WFL or CANDE *COMPILE* statements or can be made to an existing object code file by way of the WFL *MODIFY* statement.

- Never. The task attribute cannot be assigned. Such an attribute is called read-only.

Inheritance

The "Inheritance" part of the description explains whether the task attribute can inherit its value from the equivalent task attribute of an ancestor process or from a job queue attribute, session attribute, or usercode attribute.

If a task attribute is described as being inherited from the parent, then that attribute is inherited both by dependent processes and independent processes, unless otherwise stated.

Although inheritance rules are described in a definite manner, bear in mind that inherited values can be overridden by several other types of explicit and implicit assignments. Refer to "Resolving Conflicting Values" in Section 1, "Accessing Task Attributes," for more information.

Overwrite Rules

The "Overwrite Rules" part of the description specifies which of the possible sources for task attribute values takes precedence at initiation time if there is a conflict. For each task attribute, the overwrite rules are listed as *standard* or as *object code file dominant* or else described in detail. The standard overwrite rules and object code file dominant overwrite rules are discussed under "Resolving Conflicting Values" in Section 1, "Accessing Task Attributes."

Host Services

The "Host Services" part of the description states whether the task attribute is supported by Host Services. If the task attribute is supported, then it is possible for a process running on one host system to access this attribute of a process running on another host system. If the task attribute is not supported, then it is not possible to use the task attribute across hosts.

For a centralized list of the task attributes supported by Host Services, refer to the discussion of tasking across multihost networks in the *A Series Task Management Programming Guide*.

Attribute Number

The "Attribute Number" part of the description specifies the number used to identify the task attribute if an error occurs when a process accesses that task attribute. If such an error occurs, the ERROR task attribute stores the attribute number of the task attribute that was being accessed when the error occurred. A list of task attributes, in numeric order, is given in the discussion of the ERROR task attribute in this section.

Synonym

The "Synonym" part of the description lists an alternate name for the task attribute, if there is one. For the most part, synonyms were implemented because a more concise or more descriptive name was invented after the task attribute was originally implemented. The "Name" part of the task attribute description gives the preferred name for the task attribute. The "Synonym" part lists the nonpreferred name. Most languages allow you to use either name for the task attribute.

Some customers might be familiar with task attributes under the nonpreferred names. Table 2–2 summarizes the preferred and nonpreferred names for the benefit of these readers.

Table 2–2. Task Attribute Synonyms

Nonpreferred Name	Preferred Name
BACKUPDESTINATION	DESTNAME
BACKUPPREFIX	BDNAME
CHARGECODE	CHARGE
COREESTIMATE	CORE
DECLAREDPRIORITY	PRIORITY
FILE .	FILECARDS
INITIATOR	STATION
IOTIME	MAXIOTIME
OPTIONS	OPTION
ORGHOSTNAME	Deimplemented; use the leftmost part of the ITINERARY attribute value instead.
PRINTLIMIT	MAXLINES
PROCESSIOTIME	ACCUMIOTIME
PROCESSTIME	ACCUMPROCTIME
PUNCHLIMIT	MAXCARDS
QUEUE	CLASS
STACK	STACKSIZE
STACKNO	MIXNUMBER
TARGETTIME	TARGET
TASKATTERR	ERROR
VALUE	TASKVALUE

Restrictions

Most task attributes can be accessed by ALGOL, APLB, COBOL(68), COBOL74, PL/I, and WFL. However, a few of the task attributes are not available from one or more of these sources. For example, WFL cannot access event-valued task attributes. ALGOL and COBOL74 cannot use the STARTTIME and FETCH task attributes, which are specific to WFL. If such language restrictions apply to an attribute, they are discussed in the "Restrictions" part of the task attribute description.

A limited subset of the task attributes can be accessed from commands in a CANDE or Menu-Assisted Resource Control (MARC) session. Such restrictions are not documented in the task attribute descriptions. For lists of the task attributes that can be accessed from CANDE and MARC sessions, refer to the discussion of tasking from interactive sources in the *A Series Task Management Programming Guide.*

Explanation

The "Explanation" part of the task attribute description summarizes the function of the task attribute. In many cases, relevant background information, helpful hints, or cautions are also provided.

Examples

Some of the task attribute descriptions include an "Examples" part, generally because they are unusual in some way. For examples of how to access most types of task attributes from programs, refer to "Programmer Access to Task Attributes" in Section 1, "Accessing Task Attributes."

Run-Time Errors

The "Run-Time Errors" part of the task attribute description discusses task attribute access errors that occur when the program is executed rather than when it is compiled. In addition, some errors that are closely related to the task attribute are discussed. For example, the errors for exceeding resource limits are documented.

Run-time errors are usually fatal for nonprivileged processes. However, they are not fatal for privileged processes or message control systems (MCSs), unless specifically stated in the text.

The index at the end of this manual includes page references for all the error messages that are discussed in this manual.

Individual Descriptions

Unisys A Series task attributes are presented in alphabetical order throughout the rest of this section.

ACCEPTEVENT

Type	Event
Units	Not applicable
Range	HAPPENED, NOT HAPPENED
Default	NOT HAPPENED
Read Time	Anytime
Write Time	Never
Inheritance	None
Overwrite Rules	None (read-only)
Host Services	Not supported
Attribute Number	100
Synonym	None
Restrictions	Not available in WFL or APLB

Explanation

The ACCEPTEVENT task attribute accesses a predeclared event called the *accept event* that is associated with each process. The accept event is caused by the system whenever an operator enters an AX (Accept) system command for the process. A program can conveniently use this attribute in a statement that waits on several events, one of which is the ACCEPTEVENT task attribute, as in the following ALGOL example:

```
WAITANDRESET(EVNT1,EVNT2,MYSELF.ACCEPTEVENT);
```

A process can also attach its ACCEPTEVENT to an interrupt, in which case the interrupt is executed whenever an operator enters an AX command for the process.

A process can access only its own accept event. For example, a process cannot interrogate or wait on the value of the accept event of its parent. A process that attempts to do so receives a run-time error and terminates abnormally.

For more information about the AX command and about events, refer to the *A Series Task Management Programming Guide*.

Run-Time Errors

NON-LOCAL ACCEPTEVENT

A process attempted to access the ACCEPTEVENT task attribute of another process. The accessing process is discontinued with HISTORYCAUSE = 2 (PROGRAMCAU-SEV) and HISTORYREASON = 137 (NONLOCALACCEPTEVENTV).

ACCEPTEVENT ATTRIBUTE IS READONLY

A process attempted to assign an event variable to the ACCEPTEVENT task attribute. The process, if nonprivileged, is discontinued with HISTORYCAUSE = 2 (PROGRAMCAUSEV) and HISTORYREASON = 9 (ATTREADONLYV).

ACCESSCODE

Type	String
Units	Not applicable
Range	<accesscode assignment>
Default	Null string
Read Time	Anytime
Write Time	Anytime
Inheritance	From parent
Overwrite Rules	Object code file dominant
Host Services	Supported
Attribute Number	61
Synonym	None
Restrictions	None

Range

<accesscode assignment>

—<accesscode>— / —<accesscode password>————————————————|

<accesscode>
<accesscode password>

These are both nonquote identifiers.

Explanation

The ACCESSCODE task attribute affects the ability of a nonprivileged process to access files that have associated guard files. A guard file can specify that only processes with a certain accesscode are allowed to access the file. For information about guard files, refer to the *A Series Security Features Operations and Programming Guide.*

An accesscode password must be included in the value assigned to ACCESSCODE. However, the accesscode password is not included in the value returned when ACCESSCODE is read.

The system performs validation to determine whether the ACCESSCODE value for a process is compatible with the USERCODE task attribute value. When you assign ACCESSCODE to a task variable that is not in use, the system does not perform this validation until the task variable is used in a process initiation statement. When you assign an ACCESSCODE value to an in-use process, the system performs the validation immediately. The following is an outline of this validation:

1. If the process has a nonnull ACCESSCODE value, the system compares this value with the ACCESSCODELIST usercode attribute. If the ACCESSCODE value does not correspond to any of the accesscode/accesscode-password pairs in the ACCESSCODELIST, the system discontinues the process and displays a "SECURITY VIOLATION" message.

2. For a WFL job, the WFL compiler checks the usercode of the job to see if the ACCESSCODENEEDED usercode attribute is set. If it is, the WFL compiler gives a syntax error if the ACCESSCODE value of the job is null or does not correspond to any of the values in the ACCESSCODELIST usercode attribute. (A WFL job can receive an ACCESSCODE value at compile time either through inheritance or through an assignment in the job attribute list.)

If you want to change the USERCODE value of an in-use process to a different usercode that requires a different accesscode, you should generally make the USERCODE assignment first and the ACCESSCODE assignment second. Refer to the USERCODE description later in this section for details.

Examples

The following are examples of ACCESSCODE assignment and interrogation in WFL. The string variable STRVAR receives the value *TDOT*.

```
TVAR (ACCESSCODE = TDOT / ALTO);
STRVAR := TVAR (ACCESSCODE);
```

The following is an example of ACCESSCODE assignment in ALGOL:

```
REPLACE TVAR.ACCESSCODE BY "TDOT/ALTO.";
```

The following is an example of ACCESSCODE assignment in COBOL74:

```
CHANGE ACCESSCODE OF MYSELF TO "TDOT/ALTO.";
```

Run-Time Errors

ACCESSCODE ATTRIBUTE INCORRECT SYNTAX

An attempt was made to assign ACCESSCODE a value that did not follow the proper format of <*nonquote identifier*> / <*nonquote identifier*>. The process, if nonprivileged, is discontinued with HISTORYCAUSE = 2 (PROGRAMCAUSEV) and HISTORYREASON = 131 (INCORRECTSYNTAXV).

SECURITY VIOLATION

An attempt was made to assign an accesscode that does not exist, does not match
the accesscode password, or is not allowed for this usercode. The process, if
nonprivileged, is discontinued with HISTORYCAUSE = 2 (PROGRAMCAUSEV) and
HISTORYREASON = 29 (SECURITYERRORV). The following entry is made in the
system log: "INVALID TASK ATTRIBUTE: ACCESSCODE".

ACCUMIOTIME

Type	Real
Units	See below
Range	0 to about 4.31E68
Default	0
Read Time	Anytime
Write Time	Never
Inheritance	None
Overwrite Rules	None (read-only)
Host Services	Supported
Attribute Number	14
Synonym	PROCESSIOTIME
Restrictions	None

Explanation

The ACCUMIOTIME task attribute records the accumulated I/O time for the process.

The process is discontinued if the value of the ACCUMIOTIME task attribute reaches the same value as the MAXIOTIME task attribute. Refer to the MAXIOTIME description later in this section for details.

Units

When accessed from WFL, the ACCUMIOTIME value is expressed in units of seconds. When accessed from other languages, the value is expressed in units of 2.4 microseconds.

ACCUMPROCTIME

Type	Real
Units	See below
Range	0 to about 4.31E68
Default	0
Read Time	Anytime
Write Time	Never
Inheritance	None
Overwrite Rules	None (read-only)
Host Services	Supported
Attribute Number	13
Synonym	PROCESSTIME
Restrictions	None

Explanation

The ACCUMPROCTIME task attribute records the accumulated processor time for the task.

The process is discontinued if the value of the ACCUMPROCTIME task attribute reaches the same value as the MAXPROCTIME task attribute. Refer to the MAXPROCTIME description later in this section for details.

Units

When accessed from WFL, the ACCUMPROCTIME value is expressed in units of seconds. When accessed from other languages, the value is expressed in units of 2.4 microseconds.

APPLYLIST

Type	Boolean
Units	Not applicable
Range	TRUE, FALSE
Default	None
Read Time	Anytime
Write Time	Before initiation
Inheritance	None
Overwrite Rules	None
Host Services	Not supported
Attribute Number	116
Synonym	None
Restrictions	None

Explanation

The APPLYLIST task attribute, if set, causes the system to apply task equations that were previously placed in the MYPPB task attribute of the process for temporary storage.

The MYPPB value can store task equations intended to be applied to a process, or task equations intended to be applied to an object code file. Setting APPLYLIST to TRUE causes the system to apply only those equations in MYPPB that are intended for a process. For further information, refer to the discussion of the MYPPB task attribute later in this section.

Run-Time Errors

MYPPB IS EMPTY, NOTHING TO APPLY

An attempt was made to set the APPLYLIST attribute to TRUE while there were no attribute assignments stored in the MYPPB task attribute. The assignment is ignored, but the assigning process continues executing normally.

CANNOT APPLY : PPB IS FOR CODEFILE

This warning occurs if the APPLYLIST attribute is set to TRUE when the MYPPB task attribute of the compiler process stores only attributes intended for the resulting object code file. The assignment is ignored, but the assigning process continues executing normally.

AUTORESTORE

Type	Boolean
Units	Not applicable
Range	TRUE, FALSE
Default	See below
Read Time	Anytime
Write Time	Anytime
Inheritance	From parent
Overwrite Rules	Standard
Host Services	Not supported
Attribute Number	123
Synonym	None
Restrictions	None

Explanation

The AUTORESTORE task attribute specifies how the system should respond if the process attempts to open a disk file that is not present on the requested family.

If AUTORESTORE is TRUE when a process encounters a NO FILE condition for a disk file, then the system might initiate an independent runner called ARCHIVE/AUTORESTORE to copy the missing file from backup tape to disk. The system starts ARCHIVE/AUTORESTORE if all of the following conditions are true:

- The AUTORESTORE system option has a value of either YES or DONTCARE. An operator can use the AUTORESTORE (Archiving Autorestore Option) system command to assign this option.

- The reference to the file would normally produce a "NO FILE" RSVP message if the file is not resident. Thus, for example, interrogating the RESIDENT file attribute does not cause an automatic restore to take place.

- The archive directory references a backup tape that contains a backup copy of the requested file. The archive directory records the location of files backed up through the WFL *ARCHIVE* command.

- The FILENAME file attribute of the requested file specifies the same usercode as the USERCODE attribute of the requesting process.

- If the file is a cataloged file, then the generation of the file being requested matches the file listed in the archive directory.

- The process is not attempting to open a logical file that has the file attribute DUPLICATED = TRUE.

If the system does initiate an ARCHIVE/AUTORESTORE, the process requesting the file remains in an active state. On the other hand, ARCHIVE/AUTORESTORE becomes suspended and appears in the W (Waiting Mix Entries) system command display. The

RSVP message identifies the backup tape that the operator should mount. When the operator mounts the requested tape, ARCHIVE/AUTORESTORE copies the missing file back to disk. The process that originally tried to use the file then resumes execution.

If the AUTORESTORE task attribute is FALSE, or if any of the other conditions previously discussed are not true, then the system does not initiate ARCHIVE/AUTORESTORE. Instead, the system suspends the process and displays a "NO FILE <file name>" or a "NO FILE <file name> FIND ON <backup description>" RSVP message.

For an overview of the system archiving and AUTORESTORE features, refer to the *A Series System Administration Guide*.

Default

If the AUTORESTORE system option is set to NEVER or DONTCARE, then the default value of the AUTORESTORE task attribute is FALSE. If the AUTORESTORE system option is set to YES, then the default value of the AUTORESTORE task attribute is TRUE.

If the value of the AUTORESTORE system option is changed while the process is running, the change has no effect on the value of the task attribute AUTORESTORE.

AUTOSWITCHTOMARC

Type	Boolean
Units	Not applicable
Range	TRUE, FALSE
Default	FALSE
Read Time	Anytime
Write Time	Anytime
Inheritance	None
Overwrite Rules	Object code file dominant
Host Services	Not supported
Attribute Number	102
Synonym	None
Restrictions	None

Explanation

The AUTOSWITCHTOMARC task attribute affects only processes that are initiated by a MARC session and open a remote file. For these processes, AUTOSWITCHTOMARC specifies whether the originating screen is automatically displayed when the process terminates.

If AUTOSWITCHTOMARC is TRUE, the originating screen is displayed immediately upon termination of the process. If AUTOSWITCHTOMARC is FALSE, the remote file screen continues to be displayed after process termination, until the user presses the XMIT or SPCFY key.

If this task attribute is assigned more than once, only the last assignment before process termination has effect.

For more information about MARC tasking and remote files, refer to the discussion of tasking from interactive sources in the *A Series Task Management Programming Guide*.

BACKUPFAMILY

Type	String
Units	Not applicable
Range	See below
Default	See below
Read Time	Anytime
Write Time	See below
Inheritance	See below
Overwrite Rules	Object code file dominant
Host Services	Not supported
Attribute Number	63
Synonym	None
Restrictions	None

Explanation

The BACKUPFAMILY task attribute specifies the family where print and punch backup files created by the process are located.

The BACKUPFAMILY task attribute affects only backup files with a BACKUPKIND file attribute value that is equated to DLBACKUP by the SB (Substitute Backup) system command. For an illustration of this restriction, refer to the examples at the end of this subsection.

If the BACKUPFAMILY value of an in-use process is changed, the change affects only backup files created after the change is made.

The effect of the BACKUPFAMILY task attribute can be overridden for individual backup files by the FAMILYNAME print attribute. For an introduction to printing issues, refer to the discussion of controlling process I/O usage in the *A Series Task Management Programming Guide*.

Default

The BACKUPFAMILY value defaults to the current DL BACKUP family defined by the DL (Disk Location) system command.

Write Time and Range

A WFL job can include a BACKUPFAMILY assignment in the job attribute list at the start of the job, but nowhere else in the job. When BACKUPFAMILY is assigned in WFL, the value must conform to the syntax for < simple name > as defined under "Format of the Descriptions" in this section.

Aside from WFL jobs, only MCSs can make assignments to BACKUPFAMILY. When BACKUPFAMILY is assigned by an MCS, the value must be in standard form. For an explanation of standard form, refer to the description of the DISPLAYTOSTANDARD function in the *A Series DCALGOL Programming Reference Manual*. The system extracts the first identifier from the standard form value and uses this as the BACKUPFAMILY.

Inheritance

A process inherits its parent's BACKUPFAMILY value if the parent has a non-null value and the process is running on the same host as its parent.

A process initiated from a MARC session receives the BACKUPFAMILY value associated with that session.

Examples

Suppose an operator has used the SB (Substitute Backup) system command to create the following SB settings for the system:

```
SB
DISK   = DLBACKUP
PACK   = PACK
TAPE   = TAPE
PETAPE   = PETAPE
TAPE9   = TAPE9
TAPE7   = TAPE7
```

Suppose also that an operator has used the DL (Disk Location) system command to create the following DL BACKUP setting for the system:

```
DISK LOCATION:
    BACKUP      ON DBFAM
```

The following WFL job creates a backup file:

```
100 ?BEGIN JOB;
110   BACKUPFAMILY = SYSPK;
120   FILE F(KIND=PRINTER,BACKUPKIND=DISK);
130   OPEN(F);
140   LOCK(F);
150 ?END JOB
```

Line 120 of the WFL job specifies a BACKUPKIND value of DISK; but the SB setting equates DISK to DLBACKUP. The DL BACKUP setting in turn is DBFAM. Thus, by default the printer backup file would have been created on DBFAM. However, the BACKUPFAMILY statement at line 110 overrides the DL BACKUP family and causes the backup file to be created on SYSPK instead.

Now suppose that line 120 of the WFL job is changed to specify a BACKUPKIND of PACK for the backup file. The following is the modified WFL job:

```
100 ?BEGIN JOB;
110   BACKUPFAMILY = SYSPK;
120   FILE F(KIND=PRINTER,BACKUPKIND=PACK);
130   OPEN(F);
140   LOCK(F);
150 ?END JOB
```

This version of the job specifies a BACKUPKIND value of PACK. The SB setting equates PACK to PACK, and the backup file is created on the family called PACK. The system ignores the BACKUPFAMILY assignment in the WFL job because BACKUPFAMILY affects only backup files that are redirected to the DL BACKUP family by an SB substitution.

Run-Time Error

BACKUPFAMILY ATTRIBUTE MAY ONLY BE SET BY AN MCS OR TASKING PROGRAM

A process that was not an MCS or tasking program attempted to assign a value to BACKUPFAMILY. The assigning process, if nonprivileged, is discontinued with HISTORYCAUSE = 2 (PROGRAMCAUSEV) and HISTORYREASON = 54 (ONLYMCSTASKINGV).

BDNAME

Type	String
Units	Not applicable
Range	\<backup prefix\>
Default	Null string
Read Time	Anytime
Write Time	Anytime
Inheritance	From parent
Overwrite Rules	Standard
Host Services	Supported
Attribute Number	29
Synonym	BACKUPPREFIX
Restrictions	None

Range

\<backup prefix\>
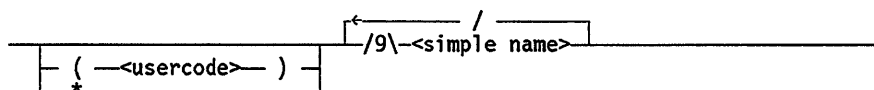
Explanation

The BDNAME task attribute causes backup files declared by the process to be
permanently saved under the file name prefix specified by the BDNAME value and
prevents the backup files from being automatically queued for printing. The user can
print out the backup files later by using a WFL *PRINT* statement.

If BDNAME is used by a nonprivileged process, backup files are saved under the
usercode of the process that declares the file. An error results if a nonprivileged process
attempts to assign a BDNAME value that includes a usercode different from the process
usercode, or an asterisk (*) in place of a usercode.

A privileged process can include a different usercode or an asterisk (*) at the start of the
BDNAME value and thus create backup files that do not have the same usercode as the
process.

The titles of the backup files follow the normal backup file titling conventions, except
that a usercode or asterisk (*) and the BDNAME value replaces the usual prefix of *BD
or *BP. For a discussion of backup file titling conventions, refer to the discussion of
controlling process I/O usage in the *A Series Task Management Programming Guide*.

File names can be a maximum of 12 nodes long, not counting the usercode. However,
the BDNAME value should not be that long because the system adds two or more nodes

to the BDNAME value when constructing the file title. In most cases, the system adds three nodes to the title.

If the BDNAME value is changed after initiation, only backup files opened after the change are affected.

Note that the BDNAME task attribute affects only backup files declared by the process. Any backup files written to by the process, but declared by another process, are not affected.

When originally implemented, the BDNAME task attribute had effect only if the BDBASE option of the OPTION task attribute was set. This is no longer the case; whether the BDBASE option is set or not set has no effect on the BDNAME task attribute.

The BDNAME task attribute has no effect on the job summary. For information about saving a copy of the job summary on disk, refer to the description of the JOBSUMMARYTITLE task attribute in this section.

The effects of the BDNAME task attribute can also be achieved through the use of several print attributes. For information about the interaction of BDNAME and these print attributes, refer to the discussion of controlling process I/O usage in the *A Series Task Management Programming Guide*.

Run-Time Errors

BDNAME ATTRIBUTE IS READONLY ON ACTIVE TASK

A process attempted to change the BDNAME value of another in-use process. The assigning process, if nonprivileged, is discontinued with HISTORYCAUSE = 2 (PROGRAMCAUSEV) and HISTORYREASON = 33 (READONLYONACTIVEV).

BDNAME ATTRIBUTE INCORRECT SYNTAX

BDNAME was assigned a value that does not conform to the backup prefix format. The assigning process is discontinued with HISTORYCAUSE = 2 (PROGRAMCAUSEV) and HISTORYREASON = 131 (INCORRECTSYNTAXV).

FILE <internal name> OPEN ERROR: TOO MANY NAMES

This error occurs when the backup file is opened if the BDNAME value caused the backup file title to have more than the allowed number of nodes. The process is discontinued with HISTORYCAUSE = 8 (SOFTIOERRCAUSEV) and HISTORYREASON = 18 (GTR14ERR).

BRCLASS

Type	Mnemonic
Units	Not applicable
Range	See "Explanation" below
Default	NOBR
Read Time	Anytime
Write Time	Anytime
Inheritance	From parent
Overwrite Rules	Object code file dominant
Host Services	Not supported
Attribute Number	83
Synonym	None
Restrictions	None

Explanation

The BRCLASS task attribute controls how the process responds to a BR (Breakout) system command. The operator can use the BR command to initiate a checkpoint for an in-use process. For a general discussion of checkpointing, refer to the discussion of restarting jobs and tasks in the *A Series Task Management Programming Guide*.

This attribute is meaningful only if the CHECKPOINTABLE attribute is TRUE. Refer to the CHECKPOINTABLE description in this section for details.

The following are the possible values and their meanings:

Mnemonic Value	Integer Value	Meaning
NOBR	0	The operator is not allowed to initiate a checkpoint for this process.
ONCEONLY	1	The operator can initiate a checkpoint for this process. The process is not allowed to continue after the checkpoint. The recovery files created by an operator BR (Breakpoint) system command are removed as soon as the RERUN statement has completed. This restriction prevents a process from being restarted more than once from this checkpoint.
MULTIPLE	2	The operator can initiate a checkpoint for this process. The process is allowed to continue execution after the checkpoint.

> *Note:* *The MULTIPLE value has effect only if it is set for the parent WFL job as well as for the checkpointed process.*

The BRCLASS attribute is reset to NOBR when the process terminates.

Example

In the following WFL job, the job attribute list assigns the job a BRCLASS value of
MULTIPLE. This value is inherited by OBJECT/PROGDATA, which becomes eligible
for multiple operator checkpoints.

```
?BEGIN JOB;
  BRCLASS = MULTIPLE;

  RUN OBJECT/PROGDATA;
?END JOB
```

Run-Time Error

BRCLASS ATTRIBUTE INCORRECT SYNTAX

An attempt was made to assign BRCLASS either an invalid mnemonic or a value
less than 0 or greater than 2. The assigning process, if nonprivileged, is discontinued
with HISTORYCAUSE = 2 (PROGRAMCAUSEV) and HISTORYREASON = 131
(INCORRECTSYNTAXV).

CHARGE

Type	String
Units	Not applicable
Range	<charge code>
Default	Null string
Read Time	Anytime
Write Time	Before initiation
Inheritance	See below
Overwrite Rules	Standard
Host Services	Supported
Attribute Number	42
Synonym	CHARGECODE
Restrictions	None

Range

<charge code>

```
     ┌←──────── / ─────────┐
─────┴┬─/14\─<simple name>─┴──────────────────────────────────────┤
```
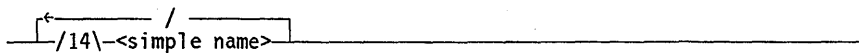
Explanation

The CHARGE task attribute contains the charge code of the process. The system logs the charge code information for each process. This information can be used by a log analysis program that computes billing charges at a site. For further information about billing, refer to the *A Series System Administration Guide*.

When a process is initiated, the system examines the USERCODE task attribute of the process and examines the usercode definition in the USERDATAFILE to determine whether the CHARGEREQ usercode attribute is set. If not, any CHARGE task attribute is accepted. If CHARGEREQ is set, the system performs the following steps to determine whether the CHARGE task attribute value is legal for the process. Remember when reading these steps that the system applies any inherited value to the process before making the following checks:

- If the CHARGE value of the process is null, the system discontinues the process.

- If the CHARGE value of the process is not null, the system compares the value with the CHARGECODE usercode attribute. If the CHARGE value does not correspond to any of the values stored in the CHARGECODE usercode attribute, the system discontinues the process.

- For a WFL job, the WFL compiler checks the usercode of the job to see if the CHARGEREQ usercode attribute is set. If it is, the WFL compiler gives a syntax error if the CHARGE value of the job is null. The WFL compiler also gives a syntax error if the CHARGE value of the job is not null and does not correspond to any of the values in the CHARGECODE usercode attribute. (A WFL job can receive a CHARGE value at compile time either through inheritance or through an assignment in the job attribute list.)

If it is necessary to change the USERCODE value of an in-use process to a different usercode that requires a different charge code, the USERCODE assignment should be made first and the CHARGE assignment second. Refer to the USERCODE description later in this section for details.

Inheritance

A process inherits the CHARGE value of its parent.

The system administrator can assign one or more charge codes to the CHARGECODE attribute of a usercode. If the system administrator also sets the USEDEFAULTCHARGE attribute of the usercode, then MARC or CANDE sessions receive the first charge code from the CHARGECODE usercode attribute at log-on time. Otherwise, MARC or CANDE requests the user to enter a charge code. Processes initiated from a MARC or CANDE session inherit the CHARGE value of the session.

A WFL job inherits a charge code from the usercode definition if all the following conditions are true:

- The job attribute list includes a USERCODE assignment or has inherited the usercode of the initiating source (such as an ODT that has a terminal usercode).

- The job attribute list did not include a CHARGE assignment and the job was submitted from a source that had no CHARGE value associated with it. (An ODT is an example of such a source.)

- The system administrator has assigned CHARGECODE and USEDEFAULT-CHARGE attributes to the usercode.

Run-Time Errors

CHARGECODE ATTRIBUTE INCORRECT SYNTAX

An attempt was made to assign CHARGE a value that was not in simple name format. If the assigning process is nonprivileged, it is discontinued with HISTORYCAUSE = 2 (PROGRAMCAUSEV) and HISTORYREASON = 131 (INCORRECTSYNTAXV).

CHARGECODE READONLY ON ACTIVE TASK, NOT CHANGED

An attempt was made to change the CHARGE value after initiation. This is a warning message rather than an error message. The process continues normally, but the requested change is not made.

INVALID CHARGECODE

The charge code assigned at initiation does not exist or is not allowed for this usercode. The new process (not the assigning process) is discontinued with HISTORYCAUSE = 2 (PROGRAMCAUSEV) and HISTORYREASON = 14 (INVALIDACCESSCODEV).

CHECKPOINTABLE

Type	Boolean
Units	Not applicable
Range	TRUE, FALSE
Default	FALSE
Read Time	Anytime
Write Time	Never
Inheritance	None
Overwrite Rules	None (read-only)
Host Services	Not supported
Attribute Number	82
Synonym	None
Restrictions	None

Explanation

The CHECKPOINTABLE task attribute specifies whether a checkpoint can be initiated for this process.

A value of TRUE indicates that a checkpoint can be initiated for this process. This value does not guarantee that the checkpoint will be executed successfully. The checkpoint can fail because of factors that are not reflected by the CHECKPOINTABLE value.

A value of FALSE indicates that the task is not allowed to execute a checkpoint.

The value of this attribute is computed at the time it is accessed.

The system evaluates the following conditions once. If any are true, the system sets the CHECKPOINTABLE attribute to FALSE for the life of the process:

- The process is an MCS or a process initiated by an MCS. This category includes processes initiated from sessions.

- The process is a frozen library. (For information about libraries, refer to the *A Series Task Management Programming Guide*.)

- The process was not initiated by a RUN statement in a WFL job.

- The code was not compiled by one of the following compilers:

  - ALGOL, DCALGOL, DMALGOL, or BDMSALGOL

  - COBOL(68), COBOL74, or BDMSCOBOL

In addition, at every access of the attribute the system checks to see whether the process has any offspring. If so, CHECKPOINTABLE returns a value of FALSE.

Another task attribute related to checkpointing, called BRCLASS, is discussed elsewhere in this section. For more information about checkpointing, refer to the discussion of restarting jobs and tasks in the *A Series Task Management Programming Guide.*

CLASS

Type	Integer
Units	Not applicable
Range	0 through 1023
Default	See below
Read Time	Anytime
Write Time	See below
Overwrite Rules	See below
Inheritance	See below
Host Services	Supported
Attribute Number	34
Synonym	QUEUE
Restrictions	None

Explanation

For WFL jobs or descendants of WFL jobs, the CLASS task attribute specifies the number of the job queue from which the WFL job is initiated. For processes not descended from WFL jobs, CLASS stores a value of zero.

The CLASS task attribute is only one of many factors affecting the job queue chosen for a WFL job. The system compares any user-specified CLASS value with the job queue definitions and terminates the WFL job if the specified CLASS is not appropriate. The system also terminates the WFL job if its CLASS value is not allowed by the CLASSLIST and ANYOTHERCLASSOK attributes of the WFL job's usercode.

If a CLASS value is not explicitly assigned, the system selects a queue for the WFL job. The job queue selection depends on such factors as the system default queue specification, the usercode definition, and any resource limits set for the job queue.

For a detailed explanation of job queues, refer to the *A Series System Administration Guide*.

Write Time

The CLASS task attribute can be assigned only in WFL jobs. Within a WFL job, CLASS can be assigned only in the job attribute list.

Overwrite Rules

This attribute can be assigned only in the job attribute list of a WFL job. For information about job attribute lists, refer to the *A Series Work Flow Language (WFL) Programming Reference Manual*.

Inheritance

A WFL job inherits a CLASS value from the usercode definition if all the following conditions are true:

- The job attribute list includes a USERCODE assignment or the job has inherited the usercode of the initiating source (such as an ODT that has a terminal usercode).

- The job attribute list did not include a CLASS assignment, and the job was submitted from a source that had no CLASS value associated with it. (An ODT is an example of such a source.)

- The system administrator has assigned a CLASS value to the usercode.

Descendants of WFL jobs inherit the CLASS value of the job. However, because only WFL jobs go through the job queue mechanism, the CLASS value has no effect on the descendants.

Example

The following is an example of a CLASS assignment in the job attribute list of a WFL job:

```
?BEGIN JOB;
  CLASS = 2;

  RUN OBJECT/X;

?END JOB
```

CODEVISIBILITY
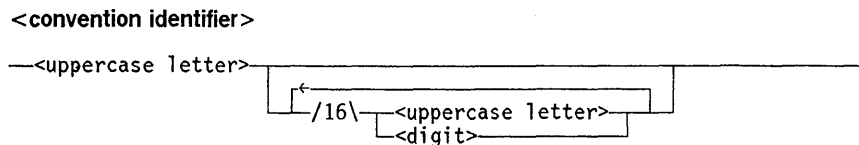
> *Note:* The CODEVISIBILITY task attribute has no meaning on systems
> running the Mark 3.9 system software release or a later release.
> The system displays a deimplementation warning message when a
> process attempts to use this attribute.

CONVENTION

Type	String
Units	Not applicable
Range	<convention identifier>
Default	See below
Read Time	Anytime
Write Time	Anytime
Inheritance	See below
Overwrite Rules	Standard
Host Services	Supported
Attribute Number	120
Synonym	None
Restrictions	None

Range

<convention identifier>

```
—<uppercase letter>
                     ┌←────────────────────────────────┐
              └─┌─/16\─┬─<uppercase letter>─┬─┘
                       └─<digit>─────────────┘
```

Explanation

The CONVENTION task attribute specifies the date, time, and currency conventions used by a process.

This task attribute affects only processes that use the CENTRALSUPPORT library to handle conventions for localization. When a process invokes a conventions procedure in the CENTRALSUPPORT library, the process can optionally use parameters to specify the convention that is desired. If the process does not request a convention in the procedure parameters, the CONVENTION task attribute of the user process determines the convention that is used.

Changes made to the value of this attribute take effect immediately. That is, subsequent calls to the conventions procedures in CENTRALSUPPORT use the new value of CONVENTION.

For further information about the CENTRALSUPPORT library, refer to the *A Series MultiLingual System (MLS) Administration, Operations, and Programming Guide.*

Default and Inheritance

A process inherits the CONVENTION value of its parent.

The default convention for A Series systems is ASERIESNATIVE. If you purchase your system through a Unisys international subsidiary, they may have already altered the CENTRALSUPPORT library to provide a different default convention. The system administrator can establish a different default convention value for the whole system by using the SYSOPS (System Options) system command.

The system administrator can selectively override the system default convention by including a CONVENTION attribute in usercode definitions in the USERDATAFILE. This CONVENTION value does not directly affect processes, but it is inherited by MARC and CANDE sessions with that usercode. The user can also use the MARC or CANDE *CONVENTION* command to change the convention of a session. Processes initiated from the session inherit the current convention of the session.

The CONVENTION attribute of a usercode also is inherited by WFL jobs that are assigned that usercode in the job attribute list.

Example

Processes that differ only in the conventions they use can benefit from this task attribute.

For example, a company might have a program that needs to print invoices for customers in several different countries. The invoices have to be printed using the conventions of each country. The following ALGOL statements run the program three times, assigning a different CONVENTION value to each run:

```
REPLACE T1.CONVENTION BY "UNITEDKINGDOM1.";
CALL DONOTHING [T1];
REPLACE T2.CONVENTION BY "FRANCELISTING.";
CALL DONOTHING [T2];
REPLACE T3.CONVENTION BY "EUROPEANSTANDARD.";
CALL DONOTHING [T3];
```

Each of these processes calls the appropriate CENTRALSUPPORT library procedures to format date, time, and currency information while generating invoices appropriate for each country.

CORE

Type	Integer
Units	Words
Range	0 to 1048575
Default	See below
Read Time	Anytime (except in WFL)
Write Time	See below
Inheritance	None
Overwrite Rules	Standard
Host Services	Not supported
Attribute Number	2
Synonym	COREESTIMATE
Restrictions	See below

Explanation

The CORE task attribute provides an estimate of the amount of main memory that a process needs for code and data areas in order to execute efficiently. The system schedules a new process if the CORE value exceeds the amount of available memory. You can override the default core estimate by assigning a different value to this task attribute.

For more information, refer to the discussion of controlling process memory usage in the *A Series Task Management Programming Guide.*

Default

The default value of CORE is taken from compiler and operating system core estimates that are stored in the object code file. For information about these estimates, refer to the discussion of process memory usage in the *A Series Task Management Programming Guide.*

Write Time

The CORE task attribute can be written at any time. However, the CORE value is used only at initiation time. Assignments made to CORE after initiation have no effect on the process.

Restrictions

In WFL, CORE can be assigned separate data core and code core values or a single total core value. Other sources can assign CORE only a single value, which is a data core estimate.

The CORE task attribute cannot be read in WFL.

Examples

The following WFL statement initiates the program OBJECT/PROG and assigns CORE a data estimate of 3000 and a code estimate of 1300:

```
RUN OBJECT/PROG;
    CORE = (3ØØØ,13ØØ);
```

The following WFL statement initiates the program OBJECT/PROG and assigns CORE a total memory estimate of 4300:

```
RUN OBJECT/PROG;
    CORE = 43ØØ;
```

Run-Time Error

CORE ATTRIBUTE INCORRECT SYNTAX

An attempt was made to assign CORE a value outside the allowed range. The assigning process, if nonprivileged, is discontinued with HISTORYCAUSE = 2 (PROGRAMCAUSEV) and HISTORYREASON = 131 (INCORRECTSYNTAXV).

DATABASE

Type	String
Units	Not applicable
Range	<database equation>
Default	Null string
Read Time	Never
Write Time	Before initiation
Inheritance	From parent
Overwrite Rules	Standard
Host Services	Not supported
Attribute Number	73
Synonym	None
Restrictions	Available only in WFL

Range

<database equation>

```
—<simple name>— ( — TITLE — = —<title>— ) ————————————|
```

Explanation

The DATABASE task attribute causes a process to use a different database than it otherwise would. This task attribute can be accessed only in WFL jobs.

In the DATABASE value, the simple name is the internal name by which the process refers to the original database. The title is the title of the database that is to be used instead.

Example

The following example shows this attribute being used in a WFL job:

```
RUN USERPROG;
   DATABASE TESTDB(TITLE=REPORT/MANAGER);
```

Run-Time Errors

DATABASE ATTRIBUTE - RESTRICTED ACCESS

An attempt was made to assign a value to the DATABASE attribute of an in-use process. The assigning process, if nonprivileged, is discontinued with HISTORYCAUSE = 2 (PROGRAMCAUSEV) and HISTORYREASON = 128 (RESTRICTEDACCESSV).

DATABASE ATTRIBUTE IS WRITEONLY

An attempt was made to read the DATABASE attribute of a process. The inquiring process, if nonprivileged, is discontinued with HISTORYCAUSE = 2 (PROGRAMCAUSEV) and HISTORYREASON = 129 (ATTWRITEONLYV).

DECKGROUPNO

Type	Integer
Units	Not applicable
Range	0 to 549755813887
Default	0
Read Time	Anytime; accurate while in use
Write Time	Never
Overwrite Rules	None (read-only)
Inheritance	None
Host Services	Not supported
Attribute Number	33
Synonym	None
Restrictions	None

Explanation

The DECKGROUPNO task attribute stores an index that is assigned by WFL to each task initiated by a WFL job. The first task initiated is assigned a DECKGROUPNO of 1, the second task a DECKGROUPNO of 2, and so on. WFL uses this information internally to determine which local data specifications are associated with which tasks.

A process initiated from any source but WFL has a DECKGROUPNO of 0.

For information about local data specifications, refer to the *A Series Work Flow Language (WFL) Programming Reference Manual*.

Example

The following WFL example includes a number of statements that display the value of DECKGROUPNO at different points during job execution. The comments at the right of the example show the values displayed by these statements.

```
?BEGIN JOB WFL/TEST;
  TASK T1, T2, T3;
  DISPLAY STRING(T1(DECKGROUPNO),*);      % Displays 0
  PROCESS RUN OBJECT/ALGOL/ERROR [T1];
     DISPLAY STRING(T1(DECKGROUPNO),*);   % Displays 1
  PROCESS RUN OBJECT/ALGOL/ERROR [T2];
     DISPLAY STRING(T2(DECKGROUPNO),*);   % Displays 2
  PROCESS RUN OBJECT/ALGOL/ERROR [T3];
     DISPLAY STRING(T3(DECKGROUPNO),*);   % Displays 3
?END JOB
```

DEPTASKACCOUNTING

Type	Mnemonic
Units	Not applicable
Range	See "Explanation" below
Default	See below
Read Time	Anytime
Write Time	Before initiation
Inheritance	See below
Overwrite Rules	Standard
Host Services	Not supported
Attribute Number	124
Synonym	None
Restrictions	None

Explanation

The DEPTASKACCOUNTING task attribute specifies whether the system should generate log entries and system messages when the process is initiated and when the process terminates. You can use DEPTASKACCOUNTING to improve overall system performance by reducing the number of log entries the system generates. The best way to achieve this effect is by establishing a system-wide DEPTASKACCOUNTING default, as described later under "Default and Inheritance."

The DEPTASKACCOUNTING task attribute can be assigned to any process. However, the system enforces the value of this task attribute only for processes that meet all the following criteria:

- The process is a task (that is, a dependent process).

- The process has the same usercode as its parent.

- The process is not initiated directly from a CANDE or MARC session or from a WFL job.

The following are the possible values of DEPTASKACCOUNTING:

Mnemonic Value	Integer Value	Meaning
UNSPECIFIED	0	This value has no effect on logging or message displays.
ANONYMOUS	1	The system does not generate Major Type 1, Minor Type 2 (BOT Entry) or Major Type 1, Minor Type 4 (EOT Entry) log entries for this process. If the system generates any other log entries for this process, the system places a Major Type 0, Minor Type 1 (Establish Identity) log entry before the first of these other entries. These logging effects apply equally to the system log and the job log.
		When the process terminates, the resource usage statistics of the process are added to those of the parent and are reflected in the Major Type 1, Minor Type 2 (EOJ Entry) or Minor Type 4 (EOT Entry) log entry that the system issues for the parent. For details about which fields in the parent's log entry can reflect statistics from an ANONYMOUS offspring, refer to the *A Series System Software Support Reference Manual*.
		Further, no BOT or EOT messages are sent to the originating station, and the process does not appear in the C (Completed Mix Entries) system command display.
		This value also affects enforcement of the FILEACCOUNTING task attribute. Refer to the FILEACCOUNTING task attribute description in this section.
IDENTIFIED	2	The system generates BOT and EOT log entries for the process. The system sends BOT and EOT messages to the originating terminal, and the process termination is recorded in the C (Completed Mix Entries) display.
		Note that an operator can use the LOGGING (Logging Options) system command to prevent logging of any BOT and EOT log entries. In this case, even processes with DEPTASKACCOUNTING = IDENTIFIED do not receive BOT or EOT log entries.

Default and Inheritance

A process inherits the DEPTASKACCOUNTING value of its parent.

The system administrator can use the ACCOUNTING (Resource Accounting) system command to specify a system-wide default for DEPTASKACCOUNTING. The system administrator can also associate a default value with a usercode by including a DEPTASKACCOUNTING usercode attribute in the usercode definition in the USERDATAFILE.

When a process is initiated, the system assigns the DEPTASKACCOUNTING task attribute the maximum of its current value (whether assigned or inherited), the system default value, and the usercode value. The integer values for each DEPTASKACCOUNTING mnemonic were previously listed under the "Explanation" subheading.

For example, suppose that DEPTASKACCOUNTING has a value of ANONYMOUS in the task variable, a value of IDENTIFIED at the system level, and a value of UNSPECIFIED at the usercode level. At initiation time, the process is assigned a DEPTASKACCOUNTING value of IDENTIFIED by the system, because IDENTIFIED has a higher numeric value (2) than ANONYMOUS or UNSPECIFIED.

On a system running InfoGuard software with a security class of S2, the system sets DEPTASKACCOUNTING to IDENTIFIED for all processes when they are initiated. This rule overrides all of the other factors affecting the DEPTASKACCOUNTING value.

DESTNAME

Type	String
Units	Not applicable
Range	<name>
Default	SITE
Read Time	Anytime
Write Time	Before initiation
Inheritance	See below
Overwrite Rules	Standard
Host Services	Supported
Attribute Number	44
Synonym	BACKUPDESTINATION
Restrictions	None

Explanation

The DESTNAME task attribute specifies a destination station for printer or punch output created by the process. This attribute is useful at sites where some of the printers are connected to data comm lines.

This attribute can be set to any of the following values:

- Any station name in the DATACOMINFO file data comm definition for the system.

- SITE. This value specifies that there is no destination station for the process. Other factors, such as the default printer pool definition, determine the routing of printer and punch files.

Setting this attribute to something other than SITE causes printer files to be built under the directory *REMLPnn/=, and punch files to be created under the directory *REMCPnn/=. The nn in the titles is the MCS number defined by the data comm subsystem for the MCS that controls the destination station. The remainder of the file name includes the job number, mix number, and so on, as described in the process I/O usage discussion in the *A Series Task Management Programming Guide*.

If COMS controls the destination station, and ReprintS is installed on the system, then the files are printed by the Print System. If remote job entry (RJE) controls the destination station, then RJE prints the files automatically. Otherwise, the files remain on disk until removed or printed by application software supplied by the site.

An alternate method of specifying the destination station for a process is the DESTSTATION task attribute. DESTSTATION specifies the logical station number (LSN) of the destination station. Assigning a valid station name to DESTNAME causes DESTSTATION to receive the corresponding LSN. Similarly, assigning a valid LSN to DESTSTATION causes DESTNAME to be updated with the corresponding station name.

Yet another method of routing printer output is the use of the DESTINATION file attribute. You can assign this file attribute to a particular printer file, or you can assign a default DESTINATION value to the PRINTDEFAULTS task attribute. If the DESTINATION value conflicts with the DESTNAME value, DESTINATION takes precedence. Note that DESTINATION does not change the value of DESTNAME; it simply prevents the DESTNAME value from being used.

For information about remote printing, refer to the *A Series Print System (PrintS/ReprintS) Administration, Operations, and Programming Guide.*

Inheritance

A process inherits the DESTNAME value of its parent.

A process initiated from a MARC or CANDE session inherits the DESTNAME value of the session. If the CANDEDESTNAME usercode attribute is set for a usercode, then MARC and CANDE use this value as the DESTNAME for sessions with that usercode. (For information about setting CANDEDESTNAME, refer to the *A Series Security Administration Guide.*) The DESTNAME value for the current session can be changed using the MARC or CANDE *DESTNAME* command.

Run-Time Errors

BACKUPDESTINATION ATTRIBUTE INCORRECT SYNTAX

An attempt was made to assign DESTNAME a value that was not in title format. (Note that BACKUPDESTINATION is a synonym for DESTNAME.) The current values of DESTNAME and DESTSTATION remain unchanged. The assigning process, unless privileged, is discontinued with HISTORYCAUSE = 2 (PROGRAMCAUSEV) and HISTORYREASON = 131 (INCORRECTSYNTAXV).

DESTNAME ATTRIBUTE IS READ ONLY ON ACTIVE TASK

An attempt was made to assign DESTNAME for an in-use process. The assigning process, if it is nonprivileged, is discontinued with HISTORYCAUSE = 2 (PROGRAMCAUSEV) and HISTORYREASON = 33 (READONLYONACTIVEV).

INVALID DESTINATION

The process was initiated with a DESTNAME value that does not correspond to any existing station or pseudostation. Note that no error is given for assigning such a DESTNAME value to a task variable. When the assignment is first made, DESTNAME is changed to the requested value and DESTSTATION is changed to zero. When the task variable is later used to initiate a process, the new process suffers the error. The process is discontinued with HISTORYCAUSE = 2 (PROGRAMCAUSEV) and HISTORYREASON = 46 (BADTASKATTRIBUTEV). The INVALID DESTINATION error message can also be displayed for a bad DESTSTATION task attribute assignment; refer to the description of DESTSTATION in this section.

UNABLE TO OBTAIN STATION NAME

An attempt was made to read DESTNAME when DESTNAME was set to the name of a nonexistent station. This error is not fatal.

DESTSTATION

Type	Integer
Units	Not applicable
Range	Valid LSNs
Default	0
Read Time	Anytime
Write Time	Before initiation
Inheritance	See below
Overwrite Rules	Standard
Host Services	Supported
Attribute Number	46
Synonym	None
Restrictions	None

Explanation

The DESTSTATION task attribute specifies a destination station for printer or punch output created by the process. This attribute is useful at sites where some of the printers are connected to data comm lines.

DESTSTATION serves the same purpose as the DESTNAME task attribute. The difference is that DESTSTATION specifies the logical station number (LSN) of the destination station rather than the station name. Assigning a valid LSN to DESTSTATION causes DESTNAME to be updated with the corresponding station name. Similarly, assigning a valid station name to DESTNAME causes DESTSTATION to receive the corresponding LSN.

DESTSTATION can be set to the LSN of any station on the system or to 0. If DESTSTATION is 0, there is no destination station for the process. In that case, other factors, such as the default printer pool definition, determine the routing of printer and punch files.

Inheritance

A process inherits its parent's DESTSTATION value. A process initiated from a MARC or CANDE session inherits the DESTNAME value of the session, and this DESTNAME, in turn, determines the DESTSTATION value.

Run-Time Errors

DATACOMM MUST BE ACTIVE TO SET DESTSTATION

An attempt was made to set DESTSTATION to a nonzero value while the number of data comm users was zero. The assigning process, if nonprivileged, is discontinued

with HISTORYCAUSE = 2 (PROGRAMCAUSEV) and HISTORYREASON = 134 (DATACOMMNOTACTIVEV).

DESTSTATION ATTRIBUTE IS READ ONLY ON ACTIVE TASK

An attempt was made to assign DESTSTATION for an in-use process. The assigning process, if nonprivileged, is discontinued with HISTORYCAUSE = 2 (PROGRAMCAUSEV) and HISTORYREASON = 33 (READONLYONACTIVEV).

INVALID DESTINATION

An attempt was made to set DESTSTATION to a value that is not a valid LSN. The DESTSTATION value is set to zero, and the DESTNAME value remains unchanged. The assigning process, if nonprivileged, is discontinued with HISTORYCAUSE = 2 (PROGRAMCAUSEV) and HISTORYREASON = 133 (INVALIDLSNV). The INVALID DESTINATION error message can also result indirectly from a bad DESTNAME task attribute assignment; refer to the description of DESTNAME in this section.

DISKLIMIT

Type	Integer
Units	Disk segments
Range	0 to 549755813887
Default	0 (unlimited)
Read Time	Anytime
Write Time	Anytime
Inheritance	See below
Overwrite Rules	See below
Host Services	Not supported
Attribute Number	59
Synonym	None
Restrictions	None

Explanation

Note: *The DISKLIMIT task attribute is being replaced by the disk resource control system, and is being deimplemented in a future release. For a description of the disk resource control system, refer to the* A Series Disk Subsystem Administration and Operations Guide. *Also refer to the descriptions of the TEMPFILELIMIT and TEMPFILEMBYTES task attributes in this section.*

The DISKLIMIT task attribute limits the number of disk segments the process can request. The process is discontinued if it requests more disk segments than are allowed by the DISKLIMIT.

Disk segment requests occur when a process creates a new disk file, or increases the size of an existing disk file to the point where a new area must be allocated for the file. (Note, however, that simply opening an existing disk file does not result in a disk segment request.) All types of disk segment requests, with the exception of overlay storage and printer backup files, are charged to the requesting process.

When a task terminates, the number of disk segments requested by the task are subtracted from the job's DISKLIMIT value.

If DISKLIMIT has not been set, there is no limit on disk usage. In this case, reading DISKLIMIT returns a value of 0. However, explicitly assigning 0 to DISKLIMIT sets a limit of 0 on disk segment usage.

Inheritance

A process inherits the current DISKLIMIT value of its parent.

If a job queue has a default value for the DISKLIMIT queue attribute, that value is inherited by the DISKLIMIT task attribute of WFL jobs run from that queue.

If a job queue has a limit value for the DISKLIMIT queue attribute, then WFL jobs that specify a greater DISKLIMIT value in the job attribute list are not allowed in that job queue.

Overwrite Rules

Standard overwrite rules apply, with the following exceptions:

- When a task is initiated, the DISKLIMIT value is the minimum of the value inherited from the parent and any value resulting from standard overwrite rules.

- For DISKLIMIT assignments to an in-use process, the maximum value that can result is the job's current DISKLIMIT value, minus the disk segments the in-use process has already requested. Attempts to assign a higher value result in this maximum value being assigned. No error or warning is issued.

Run-Time Error

DISK LIMIT EXCEEDED

The process attempted to request more disk segments than were allowed by DISKLIMIT. The process is discontinued with HISTORYCAUSE = 3 (RESOURCECAUSEV) and HISTORYREASON = 11 (DISKLIMITEXCEEDEDV).

DISPLAYONLYTOMCS

Type	Boolean
Units	Not applicable
Range	TRUE, FALSE
Default	FALSE
Read Time	Anytime
Write Time	See below
Inheritance	None
Overwrite Rules	Object code file dominant
Host Services	Not supported
Attribute Number	103
Synonym	None
Restrictions	None

Explanation

The DISPLAYONLYTOMCS task attribute specifies whether any DISPLAY messages created by the process are included in the system messages. The operator can use the MSG (Display Messages) system command to list recent system messages. If DISPLAYONLYTOMCS is FALSE, then DISPLAY messages appear in the MSG command output, as well as at the session that initiated the process. If DISPLAYONLYTOMCS is TRUE, then DISPLAY messages appear only at the session that initiated the process. A DISPLAYONLYTOMCS value of TRUE allows a process to communicate with an end user without distracting the operator.

The DISPLAYONLYTOMCS task attribute does not affect the logging of DISPLAY messages in either the job log or the system log. DISPLAY messages will be included in these logs unless the operator has used selective logging features to suppress the logging of DISPLAY messages. (For a description of selective logging features, refer to the *A Series System Software Support Reference Manual*.)

For information about DISPLAY messages, refer to the discussion of tasking from interactive sources in the *A Series Task Management Programming Guide*.

The A Series systems also provide methods for suppressing other types of messages. These methods include the SUPPRESSWARNING task attribute (discussed later in this section) and the MSC SUPPRESS form of the MSC command, which is discussed in the *A Series Menu-Assisted Resource Control (MARC) Operations Guide*.

Write Time

This task attribute can be assigned at any time for a process that is descended from a session. An example of such a process is one initiated by a CANDE *RUN* command.

However, if a process is not descended from a session, this task attribute cannot be
assigned after initiation. An example of such a process is one initiated from an ODT,
such as by a ??RUN (Run Code File) primitive system command.

ELAPSEDLIMIT

Type	Real
Units	Seconds
Range	0 to about 4.31E68
Default	0 (no limit)
Read Time	Anytime
Write Time	Anytime
Inheritance	See below
Overwrite Rules	Standard
Host Services	Supported
Attribute Number	57
Synonym	None
Restrictions	None

Explanation

The ELAPSEDLIMIT task attribute specifies the maximum elapsed time for a process. If the ELAPSEDTIME task attribute value reaches the same value as ELAPSEDLIMIT, the process is discontinued. Refer to the ELAPSEDTIME description later in this section for details.

Inheritance

Although ELAPSEDLIMIT is not inherited from the parent, the ELAPSEDLIMIT value of a process indirectly limits the elapsed time for all its descendants. This is true because when a process terminates, any in-use descendants of that process are discontinued with a "PARENT PROCESS TERMINATED" error.

If the operator defines a default value for the ELAPSEDLIMIT attribute of a job queue, the value is inherited by WFL jobs run from that job queue. If the operator defines a limit value for the ELAPSEDLIMIT attribute of a job queue, then WFL jobs that specify a greater ELAPSEDLIMIT in the job attribute list are not allowed in that job queue. For an introduction to job queue defaults and limits, refer to the discussion of tasking from programming languages in the *A Series Task Management Programming Guide*.

Run-Time Error

ELAPSED TIME LIMIT EXCEEDED

The process ran for longer than the time specified by ELAPSEDLIMIT. The process is discontinued with HISTORYCAUSE = 3 (RESOURCECAUSEV) and HISTORYREASON = 10 (ELAPSEDEXCEEDEDV).

ELAPSEDTIME

Type	Real
Units	See below
Range	0 to about 4.31E68
Default	None
Read Time	Anytime
Write Time	Never
Inheritance	None
Overwrite Rules	None (read-only)
Host Services	Supported
Attribute Number	15
Synonym	None
Restrictions	None

Explanation

The ELAPSEDTIME task attribute records the total amount of time that has passed since the initiation of the process. The process is discontinued if the value of the ELAPSEDTIME task attribute reaches the same value as the ELAPSEDLIMIT task attribute. Refer to the ELAPSEDLIMIT description in this section for details.

The ELAPSEDTIME value is unaffected by any DR (Date Reset) or TR (Time Reset) system commands entered while the process is in use. However, the ELAPSEDTIME value of a WFL job is set to zero when the job is restarted after a halt/load.

Units

When accessed from WFL, the ELAPSEDTIME value is expressed in units of seconds. When accessed from other languages, the value is expressed in units of 2.4 microseconds.

ERROR

Type	Real (string in WFL)
Units	Not applicable
Range	See "Explanation" below
Default	0
Read Time	Anytime
Write Time	Never
Overwrite Rules	None (read only)
Inheritance	None
Host Services	Not supported
Attribute Number	25
Synonym	TASKATTERR
Restrictions	None

Explanation

The ERROR task attribute indicates whether an error resulted from the most recent attempt to access a task attribute of this process. If an error did result, the ERROR value also indicates which task attribute was being accessed.

If read in WFL, the ERROR task attribute returns a string value. If the most recent task attribute access had an error, then the string is the name of the task attribute that was being accessed. If the most recent task attribute access did not cause an error, the ERROR task attribute returns a null string.

If read in other languages, the ERROR task attribute returns a real value. If the most recent task attribute access caused an error, the ERROR value is the negative of the attribute number of the attribute in error. (The USERCODE task attribute is an exception, as discussed in the following table.) If the most recent task attribute access did not cause an error, the ERROR value is 0.

The ERROR value has the following fields, which can be accessed at the bit level:

Field	Meaning
[46:01]	If set, the last task attribute access caused an error. Otherwise, it did not cause an error.
[27:20]	If the value in [07:08] is 8 (meaning a USERCODE assignment error), then this field contains a USERDATA error code. For a list of the most common USERDATA errors that can be stored in this field, refer to Table 2-3, "USERDATA Errors." For a complete list, and general information about USERDATA errors, refer to the *A Series Security Administration Guide*.
[07:08]	If [46:01] is set, this field contains the number of the last task attribute that was accessed. The task attributes are listed by number in Table 2-4, "Task Attributes by Number."

For details about how to access these fields, refer to "Accessing Task Attributes at the Bit Level" in Section 1, "Accessing Task Attributes."

The value of the ERROR task attribute is automatically erased when the task attribute is read by any process. Most MCSs read this task attribute for processes initiated from sessions. Therefore, if you initiate a process from a session, you can expect the ERROR task attribute to be blank even if a task attribute error has occurred.

In a memory dump or a program dump, you might see an ERROR value even though no task attribute error occurred. This is because the ERROR task attribute contains the attribute number of the task attribute most recently assigned, even if no error occurred. Also, the ERROR value is used by the system software as scratch storage while a job is being restarted. Both these types of values are visible only in dumps; a program that reads the task attribute finds a value of 0.

For more information about task attribute errors, refer to "Task Attribute Errors" in Section 1, "Accessing Task Attributes."

The following table lists and defines the USERDATA error numbers that can occur in field [27:20] of the ERROR task attribute value.

Table 2–3. USERDATA Errors

Error Code	Definition
8	No *SYSTEM/USERDATA file present.
9	No entry exists with the requested usercode.
10	The password supplied was invalid, or none was supplied when one was required.
16	This usercode is not a viable usercode; its entry has no system node.
17	This usercode has been marked SUSPENDED.
35	The usercode/password syntax was incorrect.
36	No usercode was specified.
45	The password has expired.
51	The password associated with the usercode has expired, and ENFORCEEXPIREDPW is true for the usercode.

Table 2–4 lists the numbers that can be returned in field [07:08] of the ERROR task attribute value, and the names of the corresponding task attributes. Note that some numbers are intentionally omitted because no task attributes correspond to those numbers.

Table 2–4. Task Attributes by Number

Number/Name	Number/Name	Number/Name
0 NAME	33 DECKGROUPNO	72 ITINERARY
1 MIXNUMBER	34 CLASS	73 DATABASE
2 CORE	35 COMPILETYPE	74 LIBRARY
3 PRIORITY	37 MYPPB	75 VISIBILITY
4 MAXPROCTIME	38 ORGUNIT	78 TIMESTARTED
5 MAXIOTIME	39 MAXCARDS	79 STARTTIME
6 TARGET	40 MAXLINES	81 JOBSUMMARY
7 STACKSIZE	41 JOBNUMBER	82 CHECKPOINTABLE
8 USERCODE	42 CHARGE	83 BRCLASS
9 TASKVALUE	44 DESTNAME	84 SW1
10 HISTORY	45 SOURCESTATION	85 SW2
11 TYPE	46 DESTSTATION	86 SW3
12 STATUS	47 SOURCEKIND	87 SW4
13 ACCUMPROCTIME	48 RESTARTED	88 SW5
14 ACCUMIOTIME	49 MAXWAIT	89 SW6
15 ELAPSEDTIME	50 STACKLIMIT	90 SW7
16 EXCEPTIONTASK	52 FETCH	91 SW8
17 LOCKED	53 RESOURCE	92 INHERITMCSSTATUS
18 STOPPOINT	55 FAMILY	94 TADS
19 PARTNER	56 WAITLIMIT	95 LANGUAGE
20 STATION	57 ELAPSEDLIMIT	96 CODEVISIBILITY
21 EXCEPTIONEVENT	58 TASKLIMIT	97 JOBSUMMARYTITLE
22 OPTION	59 DISKLIMIT	98 NOJOBSUMMARYIO
23 VALIDITYBITS	60 TANKING	99 PRINTDEFAULTS
24 FILECARDS	61 ACCESSCODE	100 ACCEPTEVENT
25 ERROR	62 SUBSYSTEM	101 LIBRARYUSERS
27 PARTNEREXISTS	63 BACKUPFAMILY	102 AUTOSWITCHTOMARC
28 RESTART	64 HOSTNAME	103 DISPLAYONLYTOMCS
29 BDNAME	66 HISTORYTYPE	104 INITPBITCOUNT
30 STACKHISTORY	67 HISTORYCAUSE	105 INITPBITTIME
31 SUBSPACES	68 HISTORYREASON	106 OTHERPBITCOUNT
32 TASKFILE	70 HSPARAMSIZE	107 OTHERPBITTIME

Table 2–4. Task Attributes by Number (cont.)

Number/Name	Number/Name	Number/Name
108 LIBRARYSTATE	116 APPLYLIST	122 MCSNAME
109 TASKWARNINGS	117 TASKERROR	123 AUTORESTORE
110 SUPPRESSWARNING	118 TEMPFILELIMIT	124 DEPTASKACCOUNTING
111 FILEACCESSRULE	119 TEMPFILEMBYTES	125 FILEACCOUNTING
112 SAVEMEMORYLIMIT	120 CONVENTION	
113 TASKSTRING	121 SOURCENAME	

Run-Time Error

ERROR ATTRIBUTE IS READONLY

An attempt was made to assign a value to the ERROR task attribute. The assigning process, if nonprivileged, is discontinued with HISTORYCAUSE = 2 (PROGRAMCAUSEV) and HISTORYREASON = 9 (ATTREADONLYV).

EXCEPTIONEVENT

Type	Event
Units	Not applicable
Range	HAPPENED, NOT HAPPENED
Default	NOT HAPPENED
Read Time	Anytime
Write Time	See below
Inheritance	None
Overwrite Rules	See below
Host Services	Not supported
Attribute Number	21
Synonym	None
Restrictions	Not available in WFL or APLB

Explanation

The EXCEPTIONEVENT task attribute accesses a predeclared event called the *exception event* that is associated with each process. When the STATUS task attribute of a process changes value, the system causes the exception event of the exception task of that process. By default, the parent is the exception task of a dependent process. Therefore, the exception event is a convenient means of informing the parent when one of its offspring has terminated or otherwise changed status.

The system also causes the exception event of a permanent library or control library whenever the value of the LIBRARYUSERS task attribute changes to zero.

The operator can also cause the exception event of a process by using the HI (Cause EXCEPTIONEVENT) system command.

The EXCEPTIONEVENT task attribute can be used in any ALGOL or COBOL74 statement that operates on an event. For example, a process can wait on the EXCEPTIONEVENT task attribute or can cause it.

A process can access the exception event of itself or of an ancestor process. The process cannot access the exception event of a descendant, sibling, or cousin process.

For a discussion of exception tasks, ancestors, siblings, cousins, and descendants, refer to the discussion of interprocess relationships in the *A Series Task Management Programming Guide*.

Write Time

A process can cause or reset the EXCEPTIONEVENT at any time. However, a process can never assign an event variable to EXCEPTIONEVENT. For example, the following ALGOL statement compiles successfully, but produces a run-time error:

```
T.EXCEPTIONEVENT := EVNT;
```

Overwrite Rules

The statements that access EXCEPTIONEVENT can be applied only to an in-use process.

Run-Time Errors

EXCEPTIONEVENT ATTRIBUTE IS READONLY

A process attempted to assign an event variable to the EXCEPTIONEVENT task attribute. The assigning process, if nonprivileged, is discontinued with HISTORYCAUSE = 2 (PROGRAMCAUSEV) and HISTORYREASON = 9 (ATTREADONLYV).

NON ANCESTRAL TASK REFERENCE

A process attempted to access the exception event of a descendant, sibling, or cousin process. The process, if nonprivileged, is discontinued with HISTORYCAUSE = 2 (PROGRAMCAUSEV) and HISTORYREASON = 130 (NONANCESTRALEXCEPTEVENTV).

EXCEPTIONTASK

Type	Task
Units	Not applicable
Range	See below
Default	See below
Read Time	Anytime
Write Time	Anytime
Inheritance	None
Overwrite Rules	Standard
Host Services	Not supported
Attribute Number	16
Synonym	None
Restrictions	Not available in WFL or APLB

Explanation

The EXCEPTIONTASK task attribute specifies the exception task for a process. When the STATUS task attribute of a process changes value, the system causes the exception event of the exception task for that process. (Note that the "exception task" is not necessarily a task; it could be a job.) A program can use the EXCEPTIONTASK task attribute to assign the process that is to be used as the exception task, or to access task attributes of the exception task.

For further information, refer to the discussion of interprocess relationships in the *A Series Task Management Programming Guide*.

Range

A process can assign any ancestral, sibling, or cousin process as the exception task. Descendant processes cannot be assigned as the exception task. (For a discussion of ancestral, sibling, cousin, and descendant processes, refer to the discussion of interprocess relationships in the *A Series Task Management Programming Guide*.)

An independent process has no exception task. When any process attempts to access the exception task of an independent process, the attempt is treated as a reference to the MYSELF task variable of the accessing process.

For remote tasks, the exception task is always the parent process. No other process can be assigned as the exception task. For information about remote tasks, refer to the discussion of tasking across multihost networks in the *A Series Task Management Programming Guide*.

Default

For a task, the parent is the default exception task. For a job, the job is its own default exception task. For a task initiated by a session, the controlling MCS is the default exception task.

Run-Time Errors

UP LEVEL TASK ASSIGNMENT

An attempt was made to assign a descendant process as the exception task. The assigning process, if nonprivileged, is discontinued with HISTORYCAUSE = 2 and HISTORYREASON = 113.

NON ANCESTRAL TASK REFERENCE

A sibling or cousin process is assigned as the exception task, and an attempt was made to access the exception event of the exception task using a statement such as "CAUSE (MYSELF.EXCEPTIONTASK.EXCEPTIONEVENT)".
The accessing process is discontinued, even if it is privileged, with HISTORYCAUSE = 2 (PROGRAMCAUSEV) and HISTORYREASON = 130 (NONANCESTRALEXCEPTEVENTV).

FAMILY

Type	String
Units	Not applicable
Range	<family specification>
Default	See below
Read Time	Anytime
Write Time	Anytime
Inheritance	See below
Overwrite Rules	Standard
Host Services	Supported
Attribute Number	55
Synonym	None
Restrictions	None

Range

<family specification>

```
—<target family>— = —<primary family>————————————————————→

→┬─ OTHERWISE —<alternate family>─┬──————————————————————┤
 └─ ONLY ─────────────────────────┘
```

<target family>
<primary family>
<alternate family>

These are each nonquote identifiers.

Explanation

The FAMILY task attribute can assign one or two substitute families to be used whenever the process references the target family. The substitute families are called the *primary family* and the *alternate family*. The alternate family is optional.

The process searches for and creates files on the substitute families whenever it would have used the target family. The following rules determine whether both substitute families, or only the primary family, are searched:

- When an existing file is being opened or executed, if the file cannot be found on the primary family, the alternate family is searched. If the TITLE file attribute does not include a usercode, then the file is searched for first under the usercode of the process and then as a nonusercoded file on each of the substitute families.

- When a file is being created, changed, removed, or secured, only the primary family is searched. The alternate family is not used.

- In the COPY and ADD statements, only the primary family is used for both sources and destinations; the alternate family is not used.

The most typical use of this task attribute is to establish a default family for files that do not have a family specified. Such files default to DISK if the FAMILY task attribute is not used. However, if the FAMILY task attribute is used, and the target family specified is DISK, then such files default to the substitute family in the FAMILY value. The following is an example of a FAMILY value that establishes ORDSPACK as the default family for a process:

```
DISK = ORDSPACK OTHERWISE DISK
```

The target family, primary family, and alternate family must be disk families. Tapes cannot be specified in the value of the FAMILY attribute.

Default

The default FAMILY setting is null, which means that no substitution takes place. The family specified by the TITLE or FAMILYNAME file attribute is used. If no family name is assigned to either of these file attributes, then DISK is used by default.

Inheritance

A process inherits the FAMILY value of its parent.

A process initiated from a MARC or CANDE session inherits the FAMILY value associated with the session. At log-on time, the session receives the FAMILY usercode attribute associated with the usercode in the USERDATAFILE. The session FAMILY can be changed using a MARC or CANDE *FAMILY* command.

If the job attribute list of a WFL job includes a USERCODE assignment, but no FAMILY assignment, then the job inherits any FAMILY usercode attribute that is defined for the usercode in the USERDATAFILE.

If a FAMILY value is assigned to a job queue, that value is inherited by WFL jobs run from that queue. A WFL job is not allowed in a job queue if the job attribute list specifies a FAMILY value different from that of the job queue. However, the job can assign a different FAMILY value after initiation.

Examples

Consider the following ALGOL program, which declares and opens two different disk files:

```
BEGIN
  FILE F(KIND=DISK,DEPENDENTSPECS=TRUE,TITLE="F ON TOOLS.");
  FILE G(KIND=DISK,DEPENDENTSPECS=TRUE);
  OPEN (F);
  OPEN (G);
END.
```

The following WFL statement would run the program and cause it to search for file F on ORDSPACK and then on DISK if necessary, and to search for file G on DISK:

```
RUN OBJECT/FILEOPEN;FAMILY TOOLS = ORDSPACK OTHERWISE DISK;
```

The following WFL statement would run the program and cause it to search for file F on TOOLS and for file G on ORDSPACK, and then on DISK if necessary:

```
RUN OBJECT/FILEOPEN;FAMILY DISK = ORDSPACK OTHERWISE DISK;
```

Run-Time Errors

FAMILY ATTRIBUTE INCORRECT SYNTAX

An attempt was made to assign FAMILY a value that does not follow the syntax for family specification. The assigning process, if nonprivileged, is discontinued with HISTORYCAUSE = 2 (PROGRAMCAUSEV) and HISTORYREASON = 131 (INCORRECTSYNTAXV).

REQUIRES *PK <family name> <file name>

This error occurs if the FAMILY value causes the process to search for a nonexistent family. In this message, <family name> is the name of the family being searched for, and <file name> is the value of the FILENAME attribute of the requested file. The process waits until an operator takes action. Refer to the *A Series System Operations Guide* for information on how to respond to waiting processes.

FETCH

Type	String
Units	Not applicable
Range	<fetch specification>
Default	Null string
Read Time	Never
Write Time	Before initiation
Inheritance	None
Overwrite Rules	See below
Host Services	Not supported
Attribute Number	52
Synonym	None
Restrictions	Available only in WFL

Range

<fetch specification>

A string of up to 256 EBCDIC characters.

Explanation

The FETCH task attribute stores instructions for the operator. The programmer can assign a string of text to FETCH. The operator can use the PF (Print Fetch) system command to display the FETCH value.

If a WFL job contains a FETCH specification, and the system option NOFETCH is reset, then the job cannot be initiated until the operator enters an OK (Reactivate) system command. The operator can set or reset the NOFETCH system option with the OP (Options) system command.

Overwrite Rules

The FETCH task attribute can be assigned only in the job attribute list in a WFL job. For the syntax of this assignment, refer to the *A Series Work Flow Language (WFL) Programming Reference Manual.*

Example

The following is an example of a WFL job that contains a FETCH specification. This specification asks the operator to mount several tapes before allowing the job to proceed.

```
?BEGIN JOB FILEIT;
    FETCH = "THIS JOB NEEDS THREE TAPE DRIVES";
    RUN NIGHTLY/UPDATE;
?END JOB
```

FILEACCESSRULE

Type	Mnemonic
Units	Not applicable
Range	See "Explanation" below
Default	DEFAULT
Read Time	Anytime
Write Time	See below
Inheritance	See below
Overwrite Rules	Standard
Host Services	Supported
Attribute Number	111
Synonym	None
Restrictions	None

Explanation

The FILEACCESSRULE task attribute specifies whether file access security checking is based on the identity of the process that declares the file or the process that opens the file. This task attribute is relevant only in cases where the declaring process and the opening process are different because a logical file is being shared among processes. For these cases, the value of the FILEACCESSRULE task attribute of the accessing process determines which type of security checking is used.

The following are the possible values and their meanings:

Mnemonic Value	Integer Value	Meaning
DEFAULT	0	This value is synonymous with DECLARER.
ACTOR	1	File access security checking is based on the identity of the process that accesses the file. Only a privileged process or an MCS can assign this value to FILEACCESSRULE.
DECLARER	2	File access security checking is based on the identity of the process that declares the file.

For a further discussion of file access security, refer to the discussion of shared files in the *A Series Task Management Programming Guide*.

Write Time

The ACTOR value can be assigned only after the process is initiated. The DEFAULT and DECLARER values can be assigned at any time.

Inheritance

A process inherits the FILEACCESSRULE value of its parent.

Run-Time Errors

FILEACCESSRULE ATTRIBUTE INCORRECT SYNTAX

A process attempted to assign FILEACCESSRULE a value not in the possible range of values. The assigning process, if nonprivileged, is discontinued with HISTORYCAUSE = 2 (PROGRAMCAUSEV) and HISTORYREASON = 131 (INCORRECTSYNTAXV).

PRIVILEGED REQUIRED TO SET FILEACCESSRULE = ACTOR

A process that was neither privileged nor an MCS attempted to assign the FILEACCESSRULE attribute the value ACTOR. The assigning process is discontinued with HISTORYCAUSE = 2 (PROGRAMCAUSEV) and HISTORYREASON = 136 (PRIVILEGEREQUIREDV).

SETTING FILEACCESSRULE TO ACTOR IS RESTRICTED TO ACTIVE TASKS

A process attempted to assign a value of ACTOR to the FILEACCESSRULE task attribute of a task variable that is not in use. This message can also occur if the ACTOR value is assigned through run-time task equation or is inherited from a FILEACCESSRULE assignment in the object code file. This error is nonfatal, but the requested assignment is ignored.

FILEACCOUNTING

Type	Mnemonic
Units	Not applicable
Range	See "Explanation" below
Default	See below
Read Time	Anytime
Write Time	Before initiation
Inheritance	See below
Overwrite Rules	Standard
Host Services	Not supported
Attribute Number	125
Synonym	None
Restrictions	None

Explanation

The FILEACCOUNTING task attribute specifies whether the system should generate log entries when the process opens or closes a file. You can use FILEACCOUNTING to improve overall system performance by reducing the number of log entries the system generates. The best way to achieve this effect is by establishing a system-wide FILEACCOUNTING default, as described later under "Default and Inheritance."

The following are the possible values of FILEACCOUNTING:

Mnemonic Value	Integer Value	Meaning
UNSPECIFIED	0	This value has no effect on logging.
ANONYMOUS	1	The system does not generate Major Type 1, Minor Type 5 (File Open) or Major Type 1, Minor Type 6 (File Close) log entries for this process. The system keeps general statistics on the file usage of the process, and issues a summary of these statistics as the Major Type 1, Minor Type 25 (File Statistics) log entry when the process terminates. However, if the system is enforcing a DEPTASKACCOUNTING value of ANONYMOUS for the process, then at termination time the system does not generate this log entry. Instead, the system adds the file usage statistics of the process to the file usage statistics of the parent. (Refer to the discussion of the DEPTASKACCOUNTING task attribute in this section.)

continued

continued

Mnemonic Value	Integer Value	Meaning
IDENTIFIED	2	The system generates File Open and File Close log entries for this process. The system does not create any File Statistics log entry for the process, nor does it add file statistics for the process to the parent's statistics.
		Note that an operator can use the LOGGING (Logging Options) system command to prevent logging of any File Open and File Close log entries. In this case, even processes with FILEACCOUNTING = IDENTIFIED do not receive File Open or File Close log entries.

Default and Inheritance

A process inherits the FILEACCOUNTING value of its parent.

The system administrator can use the ACCOUNTING (Resource Accounting) system command to specify a system-wide default for FILEACCOUNTING. The system administrator can also associate a default value with a usercode by including a FILEACCOUNTING usercode attribute in the usercode definition in the USERDATAFILE.

When a process is initiated, the system assigns the FILEACCOUNTING task attribute attribute the maximum of its current value (whether assigned or inherited), the system default value, and the usercode value. The integer values for each FILEACCOUNTING mnemonic were previously listed under the "Explanation" subheading.

For example, suppose that FILEACCOUNTING has a value of ANONYMOUS in the task variable, a value of IDENTIFIED at the system level, and a value of UNSPECIFIED at the usercode level. At initiation time, the process is assigned a FILEACCOUNTING value of IDENTIFIED by the system, because IDENTIFIED has a higher numeric value (2) than ANONYMOUS or UNSPECIFIED.

On a system running InfoGuard software with a security class of S2, the system sets FILEACCOUNTING to IDENTIFIED for all processes when they are initiated. This rule overrides all of the other factors affecting the FILEACCOUNTING value.

FILECARDS

Type	String
Units	Not applicable
Range	<file equation list>
Default	Null string
Read Time	See below
Write Time	Anytime
Inheritance	See below
Overwrite Rules	See below
Host Services	Supported
Attribute Number	24
Synonym	FILE
Restrictions	None

Range

<file equation list>

```
    ┌←─────────────────────────── ; ──────────────────────┐
──┴─ FILE ─<file internal name>─<file attribute assignment list>┴─┤
```

<file internal name>

```
──<simple name>──────────────────────────────────────────────┤
```

<file attribute assignment list>

```
         ┌←──────────────── , ────────────────┐
─ ( ─┴─<file attribute>─ = ─<file attribute value>┴─ ) ─────────┤
```

<file attribute>
<file attribute value>

For descriptions of all the file attributes and the values they can be assigned, refer to the *A Series File Attributes Programming Reference Manual.*

Explanation

The FILECARDS task attribute can be used to assign file attributes to one or more of the files declared by the process. Assignments to the FILECARDS task attribute are sometimes referred to as *file equations*. This task attribute is most frequently assigned by the parent in order to cause a task to use a file different from the one it otherwise would use.

The <file internal name> variable corresponds to the internal name of the file as it is declared in the process. The internal name is the value of the INTNAME file attribute.

If INTNAME is not assigned for the file, then it receives the file identifier as its value. The FILENAME file attribute has no effect on the internal name.

Thus, the following ALGOL file declarations both declare files with an internal name of CARD:

```
FILE CARD(FILENAME="INPUT/DATA.");
FILE LINE(INTNAME="CARD.",FILENAME="INPUT/DATA.");
```

If the FILECARDS value assigns attributes to a file that is not declared in the process, no error results, but the file attribute assignments are never used.

The file attributes assigned by FILECARDS are assigned to the logical file the first time the process references the file. A process is said to reference a file whenever it accesses a file attribute or opens a file. The FILECARDS file attribute assignments are merged with those in the file declaration. Where there is a conflict, the values assigned through FILECARDS override those assigned in the declaration. The file attributes assigned by FILECARDS can, in turn, be overridden by file attribute assignment statements later in the process.

FILECARDS can be assigned either before or during process execution. A given FILECARDS assignment has no affect on files that the process has already referenced at the time the FILECARDS assignment is made.

Note that, for a file declared within a procedure, the system creates a new logical file each time the process enters that procedure, and deallocates the logical file each time the process exits the procedure. The system applies the FILECARDS values to the logical file the first time the process references the file after each time the process enters the procedure.

Read Time

The FILECARDS task attribute can be read at any time from ALGOL. However, the value returned is encoded in an internal form that does not resemble the original FILECARDS assignments. The FILECARDS task attribute returns a null value if read from COBOL74 and cannot be read from WFL at all.

Inheritance

Internal processes inherit the FILECARDS value of the parent.

Overwrite Rules

In ALGOL or COBOL74, if the FILECARDS attribute of a task variable is assigned more than once, each assignment is merged with the previous value of the FILECARDS attribute. A file attribute assignment in the existing value is overwritten only in the following cases:

- If the new assignment specifies a different value for the same attribute of the same file.

- If a null string is assigned to FILECARDS. In this case, the FILECARDS value is restored to null.

In WFL, a FILECARDS assignment is merged with the existing FILECARDS value if the assignment includes an asterisk (*) or if the <file internal name> construct is a string primary. If no asterisk is included, and the <file internal name> construct is a name constant, then the previous FILECARDS value is discarded.

When a process is initiated, the FILECARDS values assigned through assignments to the task variable, object code file assignments, and inheritance from the parent are merged together into a single FILECARDS value. If these sources assign conflicting values to the same file attribute of the same file, then standard overwrite rules determine which file attribute assignment takes precedence.

Examples

In WFL, the syntax for assigning FILECARDS is distinguished by several special features, which are illustrated in the following example:

```
500 RUN OBJECT/DELTA ON PACK;
600    FILE OUT(KIND=DISK,TITLE=(BARNES)ACCUM/DATA ON ORDSPACK);
700    FILE IN=(JACOB)INPUT/DATA ON ORDSPACK;
800    FILECARDS CARD(KIND=READER);
```

The RUN statement at line 500 initiates a task. The statements at lines 600, 700, and 800 are all assignments to the FILECARDS attribute of that task. Although FILECARDS is a string-valued task attribute, in WFL the FILECARDS value is not enclosed in quotation marks ("). The assignment at line 600 shows how multiple file attributes can be assigned to the same file. The assignment at line 700 shows an abbreviated syntax that can be used if TITLE is the only attribute being assigned to a file. Line 800 shows the same syntax as line 600, except that FILECARDS is used instead of its synonym FILE.

The CANDE and MARC syntaxes for assigning FILECARDS are the same as the WFL syntax, except that FILECARDS must be referred to by its synonym, FILE.

The ALGOL syntax for assigning FILECARDS also differs from that used to assign other string-valued task attributes. The value is terminated by 48"00" instead of by a period (.). The following is an example:

```
REPLACE CTASK.FILECARDS BY
    "FILE CARD (KIND=DISK, TITLE=ALGOL/TASK);"
    "FILE CODE (KIND=DISK, TITLE=OBJECT/ALGOL/TASK);" 48"ØØ";
```

The following ALGOL statement resets the FILECARDS value to a null string:

```
REPLACE T.FILECARDS BY 48"ØØ" ;
```

The following COBOL74 statements assign attributes to two files. The second assignment does not overwrite the first assignment, but rather is merged with it:

```
CHANGE ATTRIBUTE FILECARDS OF TASK-VAR-1 TO
    "FILE CARD(KIND=DISK,TITLE=JUNK/JUNK);".
CHANGE ATTRIBUTE FILECARDS OF TASK-VAR-1 TO
    "FILE LINE(KIND=DISK,TITLE=JUNK/JUNK3);".
```

Run-Time Errors

FILECARDS ATTRIBUTE IS READONLY ON ACTIVE TASK

An attempt was made to assign the FILECARDS value of an in-use process. The assigning process, if nonprivileged, is discontinued with HISTORYCAUSE = 2 (PROGRAMCAUSEV) and HISTORYREASON = 33 (READONLYONACTIVEV).

FILECARDS ATTRIBUTE INCORRECT SYNTAX

There were one or more syntax errors in the file attribute assignments in the FILECARDS value. The assigning process, if nonprivileged, is discontinued with HISTORYCAUSE = 2 (PROGRAMCAUSEV) and HISTORYREASON = 131 (INCORRECTSYNTAXV).

HISTORY

Type	Real
Units	Not applicable
Range	See "Explanation" below
Default	0
Read Time	Anytime
Write Time	Never
Inheritance	None
Overwrite Rules	None (read-only)
Host Services	Not supported
Attribute Number	10
Synonym	None
Restrictions	None

Explanation

The HISTORY task attribute records the type of termination a process had. If termination was abnormal, HISTORY also stores information about why the abnormal termination occurred. The HISTORY value is divided into the following fields:

Field	Meaning
[47:01]	The operating system sometimes sets this bit for internal purposes.
[46:01]	If this bit is set, and field [07:08] stores a value of 4, then initiation of the process failed.
[45:01]	If this bit is set, the process cannot be discontinued.
[43:20]	The operating system sometimes stores information in this field for internal purposes.
[23:08]	If the process was discontinued or is suspended, this field stores the specific reason. This field corresponds to the value of the HISTORYREASON task attribute. Refer to the HISTORYREASON description in this section for details.
[15:08]	If this process was discontinued or is suspended, this field stores the general reason. This field corresponds to the value of the HISTORYCAUSE task attribute. Refer to the HISTORYCAUSE description in this section for details.
[07:08]	This field stores information about the process state. If the process has terminated, this field also records the general type of termination. This field corresponds to the value of the HISTORYTYPE task attribute. Refer to the HISTORYTYPE description in this section for details.

For details about how to access these fields, refer to "Accessing Task Attributes at the Bit Level" in Section 1, "Accessing Task Attributes."

HISTORYCAUSE

Type	Mnemonic
Units	Not applicable
Range	See "Explanation" below
Default	0
Read Time	Anytime
Write Time	Never
Inheritance	None
Overwrite Rules	None (read-only)
Host Services	Supported
Attribute Number	67
Synonym	None
Restrictions	None

Explanation

The HISTORYCAUSE task attribute specifies what general type of condition caused the process to terminate abnormally or be suspended. The HISTORYCAUSE value is the same as field [15:08] of the HISTORY task attribute.

If the process did not terminate abnormally and is not suspended, the HISTORYCAUSE value is 0. No mnemonic is associated with this value.

If the process terminated abnormally, then the HISTORYTYPE value is DSEDV, and the following are the possible HISTORYCAUSE values and their meanings:

Mnemonic Value	Integer Value	Meaning
(none)	0	The process has not been initiated, is still in use, or terminated normally.
OPERATORCAUSEV	1	The process was discontinued by a system command such as DS (Discontinue).
PROGRAMCAUSEV	2	The process was deliberately terminated for one of the following reasons:
		• A value of TERMINATED was programmatically assigned to the STATUS task attribute.
		• The process attempted an action that is not allowed by the operating system.
RESOURCECAUSEV	3	The process was terminated for exceeding a resource limit, such as MAXPROCTIME or MAXIOTIME.

continued

continued

Mnemonic Value	Integer Value	Meaning
FAULTCAUSEV	4	The process was terminated because it requested a machine operation that could not be executed, such as dividing by zero or reading past the end of an array.
SYSTEMCAUSEV	5	The process was terminated because it violated a system parameter, such as overlay row size or the amount of memory allowed.
DCERRCAUSEV or DCERRV	6	The process was terminated because of a data comm error.
IOERRCAUSEV or IOERRV	7	The process was terminated because of a physical I/O error.
SOFTIOERRCAUSEV or SOFTIOERRV	8	The process was terminated because of a logical I/O error.
NEWIOERRCAUSEV or NEWIOERRV	9	The process was terminated because of an error in opening a file.
UNIMPLEMENTEDCAUSEV or UNIMPLEMENTEDV	10	The process was terminated because it attempted to use a feature that has not been implemented.
UNSPECIFIEDCAUSEV	11	The process was terminated because of an error of an unknown type.
EBDMSERRCAUSEV or EBDMSERRV	12	The process was terminated because of a Data Management System II (DMSII) error.
NETWORKCAUSEV	13	The process was terminated because of a BNA-related error. For example, the process might have failed initiation because of a missing host or a missing object code file on a remote host.
SOFTIOERR2CAUSEV or SOFTIOERR2V	14	The process was terminated because of a logical I/O error.

If the process is suspended, then the HISTORYTYPE value is STEDV, and the following are the possible HISTORYCAUSE values and their meanings:

Mnemonic Value	Integer Value	Meaning
OPERATORCAUSEV	1	The process was suspended by the ST (Stop) system command.
PROGRAMCAUSEV	2	The process was suspended for one of the following reasons: ● A resource needed by the process is missing. ● The STATUS task attribute was programmatically assigned a value of SUSPENDED.

continued

continued

Mnemonic Value	Integer Value	Meaning
SYSTEMCAUSEV	5	The process was suspended because of a shortage of available memory.
NETWORKCAUSEV	13	The process was suspended because of a BNA condition.

For a list of process termination messages and their relationship to HISTORYCAUSE values, refer to the discussion of process history in the *A Series Task Management Programming Guide.*

HISTORYREASON

Type	Mnemonic
Units	Not applicable
Range	See "Explanation" below
Default	0
Read Time	Anytime
Write Time	Never
Inheritance	None
Overwrite Rules	None (read-only)
Host Services	Supported
Attribute Number	68
Synonym	None
Restrictions	Not available in WFL; however, for a description of how to extract the same information from the HISTORY task attribute, refer to "Accessing Task Attributes at the Bit Level" in Section 1.

Explanation

The HISTORYREASON task attribute indicates the specific reason why a process terminated abnormally or was suspended. The HISTORYREASON value corresponds to field [23:08] of the HISTORY task attribute.

Most HISTORYREASON integer values have mnemonics associated with them. Each mnemonic briefly describes one reason this HISTORYREASON integer value could have occurred. You can determine which mnemonic applies in a particular case by using the HISTORYREASON integer value with the HISTORYTYPE and HISTORYCAUSE values.

If the process did not terminate abnormally and is not suspended, the HISTORYREASON value is 0. No mnemonic is associated with this value.

One standard method of reading mnemonic-valued task attributes might yield confusing results if applied to HISTORYREASON. The following is an ALGOL example of this method:

```
IF T.HISTORYREASON = VALUE(DIVIDEBYZEROV) THEN ...
```

The mnemonic DIVIDEBYZERO is associated with a HISTORYREASON value of 1. The expression shown in the example evaluates to TRUE whenever HISTORYREASON has a value of 1. However, a HISTORYREASON value of 1 indicates a DIVIDEBYZERO error only if HISTORYTYPE = DSEDV and HISTORYCAUSE = FAULTCAUSEV.

The following is a better method of reading HISTORYREASON. This example evaluates to TRUE only if a DIVIDEBYZERO error occurred:

```
IF T.HISTORYTYPE = VALUE(DSEDV)
   AND T.HISTORYCAUSE = VALUE(FAULTCAUSEV)
   AND T.HISTORYREASON = VALUE(DIVIDEBYZEROV) THEN ...
```

The following pages list the possible HISTORYREASON values for each combination of HISTORYTYPE and HISTORYCAUSE values. For HISTORYREASON values that have mnemonics, the mnemonics are listed under the column heading "Mnemonic Value." For HISTORYREASON values that do not have mnemonics, a short explanatory phrase is listed under the column heading "History Reason (No Mnemonic)."

HISTORYTYPE = 3 (STEDV), HISTORYCAUSE = 1 (OPERATORCAUSEV)

Integer Value	Mnemonic Value
0	(No mnemonic. This value means the ST (Stop) system command was entered from an ODT.)
1	REMOTELYCAUSEDV

HISTORYTYPE = 3 (STEDV), HISTORYCAUSE = 2 (PROGRAMCAUSEV)

Integer Value	Mnemonic Value
1	RESPONSEREQUIRED

HISTORYTYPE = 3 (STEDV), HISTORYCAUSE = 13 (NETWORKCAUSEV)

Integer Value	Mnemonic Value
3	SUSPENDEDV

HISTORYTYPE = 4 (DSEDV), HISTORYCAUSE = 1 (OPERATORCAUSEV)

Integer Value	Mnemonic Value
0	RSVPV
1	CLEARUNITV
2	JUSTDSEDV

HISTORYTYPE = 4 (DSEDV), HISTORYCAUSE = 2 (PROGRAMCAUSEV)

Integer Value	Mnemonic Value
0	MISSINGCODEFILENAMEV
1	MISSINGCODEFILEV
3	INITACTIVETASKV
4	NOEXTERNALRUNV
5	VISITNONACTIVEV
6	ILLEGALVISITV

continued

continued

HISTORYTYPE = 4 (DSEDV), HISTORYCAUSE = 2 (PROGRAMCAUSEV)

Integer Value	Mnemonic Value
7	DYNCODEEOFV
8	BADD1STRETCHV
9	ATTREADONLYV
11	NOTSESSIONNUMBERV
12	NONANCESTRALTASKFILEV
13	NOTIMPLEMENTEDV
14	INVALIDACCESSCODEV
15	INCOMPATIBLEBOXESV
18	DEATHINFAMILYV
19	CRITICALBLOCKV
20	BADGOTOV
23	INVALIDPARAMETERV
25	INCOMPATIBLECODEV
26	NOTEXECUTABLEV
27	UNMATCHEDPARAMSV
28	INVCOMPILERVV
29	SECURITYERRORV
30	LIBMAINTV
31	ILLEGALTASKXFERV
32	BADRESIZEDEALLOCV
33	READONLYONACTIVEV
37	MISSINGINTRINSICV
38	INCOMPATIBLELEVELV
39	INFANTICIDEV
40	NOTBOUNDV
41	ILLEGALOWNARRAYV
42	DIMSIZERRORV
43	UPLEVELATTACHV
44	ILLEGALSWAPV
46	BADTASKATTRIBUTEV
47	MISSINGCARDDECKV
48	BADRESTARTV
49	BADEVENTUSAGEV
50	BADGIVELOCKV

continued

HISTORYTYPE = 4 (DSEDV), HISTORYCAUSE = 2 (PROGRAMCAUSEV)

Integer Value	Mnemonic Value
51	BADGETLOCKV
52	ONLYMCSMAYSETV
53	DCKEYINSIZEV
54	ONLYMCSTASKINGV
56	NONOWNERACCESSV
57	COMPILERSONLYV
58	TASKLIMITEXCEEDEDV
59	AXBADARRAYV
60	RUNTIMEWFLV
61	COMPILERERRORV
62	XSPARAMSV
63	SORTKILLV
64	MISSINGSYSTEMLIBV
65	ALREADYSELECTEDV
66	LIBMISSINGNAMEV
67	LIBTYPEMISMATCHV
68	LIBNOTINITIATEDV
69	CYCLICPROVISIONV
70	PREVIOUSLYFROZENLIBV
71	LIBIMPLEMENTATIONERRORV
72	FASTTASKFREEZEV
73	NONUNIQLIBV
74	SAVELIBTASKNEVERCALLEDV
75	LIBNEVERFROZEV
77	BADLIBTASKV
78	LIBFEATURENOTIMPLEMENTEDV
79	BADCOMPILERINDEXV
80	LIBNOTPROCESSEDORRUNV
81	LIBMUSTBESEPARATESTACKV
82	INVALIDPARAMV
83	FORTRANERRV
84	PLIRUNTIMEERRV
85	INTRINSICSERRV
86	MATHERRV

continued

continued

HISTORYTYPE = 4 (DSEDV), HISTORYCAUSE = 2 (PROGRAMCAUSEV)

Integer Value	Mnemonic Value
87	FORMATERRV
88	LIBDEIMPLEMENTATIONERRORV
89	LIBLEVELINCOMPATIBLEV
90	BADLIBTITLEV
91	CANTLINKTOASYSTEMLIBV
92	NOTASYSTEMLIBV
93	NOTLIBRARYCAPABLEV
94	LISTSERRORV
95	LIBPARENTNOTALIBV
96	BADLIBFUNCTIONV
97	INVALIDATTVALUEV
98	UNAUTHORIZEDLIBUSEV
99	FOREIGNTASKINITFAILV
100	PORTSERRORV
101	LIBCANCELERRV
102	INVALIDSAVECORELIMITV
103	NONVISTASKFILEV
104	BADINSCRIBEV
105	BADERASEV
106	CLIENTDIEDINACRV
107	BADPOBOXUSAGEV
108	INVALIDSTKNOV
109	BADTCPREQV
110	BYRESTRICTIONV
111	LIBWRONGMARKLEVELV
112	NOINITIATORV
113	UPLEVELTASKASSIGNV
114	FRAMEEXCEEDEDV
115	CODEFILEINCOMPATIBLEWITHMCPV
116	CODEFILENOTACTIVEV
117	BADPPBV
118	NOTAFASTTASKV
119	FASTTASKINITTASKV
120	FASTTASKNOPARAMSV

continued

continued

HISTORYTYPE = 4 (DSEDV), HISTORYCAUSE = 2 (PROGRAMCAUSEV)

Integer Value	Mnemonic Value
121	FASTTASKNOTADSV
122	FASTTASKBADEXCEPTIONTASKV
123	STACKHASFAMILYV
124	FASTTASKFAULTEDV
125	DATABASEDIEDV
126	LIBRARYDIEDV
127	STACKHASUNITATTACHEDV
128	RESTRICTEDACCESSV
129	ATTWRITEONLYV
130	NONANCESTRALEXCEPTEVENTV
131	INCORRECTSYNTAXV
132	ATTACCESSFAULTV
133	INVALIDLSNV
134	DATACOMMNOTACTIVEV
135	VALUETOOLARGEV
136	PRIVILEGEREQUIREDV
137	NONLOCALACCEPTEVENTV
138	INVSCHEDACTV
139	INVTIMESTATV
140	INVREACTIVATEV
141	INVSOURCEV
142	INVDUMPPARAMV
143	INVCPMACTIONV
144	INVPREFACTIONV
145	INVDISCONNECTV
146	INVDESTINATIONV
147	BLOCKHASNOSCWV
148	PRPROVIDERGONEV
149	LIBWRONGCODEFILEV
150	(No mnemonic. The stepparent of the process terminated.)

HISTORYTYPE = 4 (DSEDV), HISTORYCAUSE = 3 (RESOURCECAUSEV)

Integer Value	Mnemonic Value
0	PROCESSEXCEEDEDV
1	IOEXCEEDEDV
2	STACKEXCEEDEDV
3	PRINTEXCEEDEDV
4	PUNCHEXCEEDEDV
5	CARDREADEXCEEDEDV
6	MEMORYEXCEEDEDV
8	TAPEEXCEEDEDV
9	WAITEXCEEDEDV
10	ELAPSEDEXCEEDEDV
11	DISKLIMITEXCEEDEDV
12	STRINGPOOLEXCEEDEDV
13	FAMILYSIZEEXCEEDEDV
14	SAVECORELIMITEXCEEDEDV
15	CAUEXCEEDEDV
16	SEGLIMITEXCEEDEDV

HISTORYTYPE = 4 (DSEDV), HISTORYCAUSE = 4 (FAULTCAUSEV)

Integer Value	Mnemonic Value
1	DIVIDEBYZEROV
2	EXPOVERFLOWV
3	EXPUNDERFLOWV
4	INVALIDINDEXV
5	INTEGEROVERFLOWV
6	INACTIVEQV
7	MEMORYPROTECTV
8	INVALIDOPV
9	LOOPV
10	MEMORYPARITYV
11	SCANPARITYV
12	INVALIDADDRESSV
13	STACKOVERFLOWV
14	STRINGPROTECTV
16	FALSEASSERTV
17	SEQUENCEERRORV

continued

HISTORYTYPE = 4 (DSEDV), HISTORYCAUSE = 4 (FAULTCAUSEV)

Integer Value	Mnemonic Value
18	INVALIDPCWV
19	STACKUNDERFLOWV
21	LIBLINKERRORV
22	INVALIDINTV
23	MEMFAIL1V
26	MEMORYFAIL2V
30	PROCINTERNALV
35	PROCDIEDV
37	BCLPOINTERV
40	DISKPARITYV
41	EMODEVIOLATIONV
42	NOACTIVELINKV
43	PROCLINKPARITYV
45	BOTTOMOFSTACKV
46	RUNLIGHTOUTV
47	STACKSTRUCTUREV
48	BADMSCWV

HISTORYTYPE = 4 (DSEDV), HISTORYCAUSE = 5 (SYSTEMCAUSEV)

Integer Value	Mnemonic Value
1	NOMEMV
2	PARITYONPBITV
3	ARRAYTOOLARGEV
4	INCOMPATIBLEWFLJOBFILEV
8	(No mnemonic. The process was using an object code file or a data file on a disk unit that was closed by the *CLOSE PK* *<unit number>* *:DS* form of the CLOSE (Close Pack) system command.)
9	(No mnemonic. The MCP encountered an error while handling software interrupts.)

HISTORYTYPE = 4 (DSEDV), HISTORYCAUSE = 6 (DCERRCAUSEV)

Integer Value	History Reason (No Mnemonic)
10	Message size error
12	Unknown file or station
13	File subtraction error

HISTORYTYPE = 4 (DSEDV), HISTORYCAUSE = 7 (IOERRCAUSEV)

Integer Value	History Reason (No Mnemonic)
0	Either a train printer I/O error occurred and could not be resolved, or else the MCP procedure PATHRES did not not successfully complete. PATHRES performs functions such as loading disk controller firmware.
6	Direct I/O attribute error

HISTORYTYPE = 4 (DSEDV), HISTORYCAUSE = 8 (SOFTIOERRCAUSEV)

Integer Value	History Reason (No Mnemonic)
0	No error
1	Label parity error
2	Parity error on position
3	Invalid translation
4	Incompatible blocking
5	Illegal output reverse
6	Illegal input reverse
7	Short tape blocking
8	Illegal output file
9	No buffer space
10	No space in header
11	Duplicated file
12	Illegal direct I/O
14	Exceeded resources
15	No unit
16	Illegal optional file
17	Illegal final reel
18	Too many names
19	Failed entry
20	Illegal MYUSE value

continued

continued

HISTORYTYPE = 4 (DSEDV), HISTORYCAUSE = 8 (SOFTIOERRCAUSEV)

Integer Value	History Reason (No Mnemonic)
21	Illegal NEWFILE value
22	DCOPEN failed
23	No write ring
24	Failed volume entry
25	Illegal unlabeled volume
26	Illegal BLOCKSTRUCTURE or FILETYPE
27	Illegal reel number
28	Find routines failed
29	Illegal backward seek
30	Illegal read reverse
31	Illegal seek
32	Parity error on seek
33	Read on output file
34	Read on unopened file
35	Read reverse on unopened file
36	Seek on unopened file
37	Space forward on output file
38	Write on code file
39	Write on input file
40	Write on unopened file
41	Buffer in use
42	Up-level event
43	Security error
44	No room for buffer
45	Unknown error
46	Logic error
47	Already closed
48	No read before rewrite
49	No read before delete
50	Delete on non-I/O file
51	Illegal update file
52	Incompatible file organization

continued

HISTORYTYPE = 4 (DSEDV), HISTORYCAUSE = 8 (SOFTIOERRCAUSEV)

Integer Value	History Reason (No Mnemonic)
53	Close not called
54	File information block (FIB) stack transition error
55	Locking error
56	Kind list not allowed
57	Dialog communication failure with other host
59	File not removed on disk
60	File not cataloged
61	Checkpoint file title not changed
62	Write user label error
63	RELEASEHEADER error
64	Tried to write beyond end of file (EOF)
65	Rewrite on non-I/O file
66	Logical/physical file mismatch
67	Seek on output file
68	Tape position error
69	Distributed systems service (DSS) cannot handle this file
70	Access restricted to APL
71	Open after close with lock
72	Illegal write random
73	Illegal read random
74	Not closed
75	Unexpected I/O error
78	Data error
79	Deleted/duplicate record
80	Parity error
81	I/O not done
82	Invalid subfile
83	Broadcast read error
84	Subfile is closed
85	No available buffer
86	No available message

continued

continued

HISTORYTYPE = 4 (DSEDV), HISTORYCAUSE = 8 (SOFTIOERRCAUSEV)

Integer Value	History Reason (No Mnemonic)
87	Port not connected
88	End of file (EOF)
89	Illegal short block read
90	Break on output
91	Unit in rewind
92	Time limit exceeded
93	File not available
94	No file
95	Mismatched genealogy
96	Mismatched serial number
97	File not resident
98	Pack not present
99	Invalid access code
100	Foreign file open error
101	Port offer error
102	Illegal hostname for foreign file
103	Data might have been lost
104	Record count error
105	Block count error
106	Host not reachable
107	Write lockout
108	FRAMESIZE and INTMODE values incompatible
109	Binary I/O not allowed
110	End of page
111	BCL not allowed on this machine
112	No continuation pack for audited file
113	Cannot be audit file
114	I/O error occurred during flushing of buffers
115	Too many backup files
116	Maximum audit length exceeded
117	Unable to position file at end

continued

HISTORYTYPE = 4 (DSEDV), HISTORYCAUSE = 8 (SOFTIOERRCAUSEV)

Integer Value	History Reason (No Mnemonic)
118	Unsupported function
119	Bad use of use routines
120	Must have usercode to use DSS
121	Invalid port name
122	Requires direct I/O
123	SB must contain a disk/pack unit
124	EIO logic error
125	Invalid array index
126	Incompatible I/O length
127	SIZEVISIBLE/FRAMESIZE/INTMODE value conflict
128	I/O error occurred during closing of file
129	I/O support library error
130	I/O error
131	INQ_LIST allocation failed
132	End file not allowed
133	I/O error changing host control (HC) unit access mask register (AMR)
134	Unsupported protocol type
135	Protocol error
136	No resource to open port
137	YOURHOST is not in YOURHOSTGROUP
138	User is not an authorized user of the application group
139	Support library unavailable
140	Error in one or more port-subfiles open operations
141	Error in one or more port-subfiles close operations
142	Incompatible attribute value or values
143	Function not available
144	Unacceptable character set
145	Networking not supported

continued

continued

HISTORYTYPE = 4 (DSEDV), HISTORYCAUSE = 8 (SOFTIOERRCAUSEV)

Integer Value	History Reason (No Mnemonic)
146	TRANSLATE=FORCESOFT not allowed with binary I/O
147	I/O error clearing adapter or unit
148	Access restrictions not met
149	Cannot create restricted file
150	Security error on output tape open
151	Cannot write on guard file
152	Logical I/O not supported for this type of unit
153	Attribute already set in physical file
154	FAMILYOWNER conflicts with task usercode
155	Illegal I/O to coactive disk
156	Coactive unit not in output mode
157	Incompatible with this MCP version
158	DSS dialogue number too large for logical I/O
159	I/O error occurred during closing of file
160	Tape drive mode change operation failed
161	BYTES is not supported by this unit
162	Random add not allowed unless delete-capable
163	Not delete-capable
164	Record has not been read
165	Beyond extend area
166	Record not locked
167	Record position occupied
168	Sequential write not permitted to EFS direct file
170	Attempt to exceed family limit
171	Family integral limit exceeded
172	Attempt to exceed temporary file limit
174	Illegal write option specified
175	Invalid specification of ANYSIZEIO

continued

continued

HISTORYTYPE = 4 (DSEDV), HISTORYCAUSE = 8 (SOFTIOERRCAUSEV)

Integer Value	History Reason (No Mnemonic)
176	Area length exceeds maximum allowed
177	Logical file INTMODE incompatible with permanent file FRAMESIZE
178	Incompatible FILESTRUCTURE
179	Permanent file FILESTRUCTURE must be STREAM
180	Logical file MAXRECSIZE inconsistent with permanent file MAXRECSIZE
181	Logical file BLOCKSIZE inconsistent with permanent file BLOCKSIZE
182	MAXRECSIZE exceeds AREALENGTH
183	Logical file FRAMESIZE incompatible with permanent file area length
184	Unsupported parameter for this service
185	Local interprocess communication (IPC) not supported for this service
186	Unsupported translation for this service
187	DIOFILESTRUCTURE value requires FILESTRUCTURE to be set
188	Not in proper state for direct I/O to unit EIO
189	Cannot access a file of this FILEKIND
190	Open rejected by correspondent
191	Close rejected by correspondent
192	Endpoint not registered
193	Invalid respond option
194	Service invalid for provider
195	Provider restricted
196	Connect time limit exceeded
197	Correspondent does not support APPLICATIONCONTEXT value
198	Correspondent rejected DEFAULTPCONTEXT value
199	Invalid value or values for DEFINEDPCONTEXTSET
200	Warning—DEFINEDPCONTEXTSET values have changed

continued

continued

HISTORYTYPE = 4 (DSEDV), HISTORYCAUSE = 8 (SOFTIOERRCAUSEV)

Integer Value	History Reason (No Mnemonic)
201	Warning—port attribute ignored
202	Invalid associated data
203	Associated data too long
204	Invalid attribute values for respond
205	Invalid attribute values for AWAITOPEN
206	DEPENDENTSPECS must be TRUE when DIOFILESTRUCTURE=SECTORSTREAM
207	Warning—initiator close collision
208	Warning—responder close collision
209	Endpoint incompatible with service
210	Unsupported primitive
211	Open failure in KEYEDIO library
212	Read reverse is not supported by this unit
213	Specified MAXRECSIZE is not supported by this unit
214	MAXRECSIZE must equal BLOCKSIZE for this unit
215	MCP does not support DMSII use of this FILESTRUCTURE
216	Insufficient disk space
217	Operator entered OF (Optional File) system command
218	KEYEDIOII write error occurred
219	Unmatched DIOFILESTRUCTURE value
220	Invalid connect TIMELIMIT value
221	Error encoding data
222	No data available to be read
223	Error on broadcast write
224	No buffer available for write
226	Open data was received
227	Open response data was received
228	Close request data was received
229	Close abort data was received
230	Close response data was received

continued

continued

HISTORYTYPE = 4 (DSEDV), HISTORYCAUSE = 8 (SOFTIOERRCAUSEV)

Integer Value	History Reason (No Mnemonic)
231	More data to come
232	Fault in use routine
233	Logical file MINRECSIZE inconsistent with permanent file MINRECSIZE
233	Logical file MINRECSIZE inconsistent with permanent file MINRECSIZE
234	BASICSERVICE violation
235	Action not valid in this FILESTATE
236	Transparent LOCALSYNTAX cannot be supported for this subfile
237	Open aborted by correspondent
238	Open rejected—transient
241	Requested PROVIDERGROUP not defined
244	Write on read-only file
245	During a file open, either the CENTRALSUPPORT library could not be accessed, or CENTRALSUPPORT reported an error related to CCSVERSION validation, INTMODE/EXTMODE validation, or translation tables availability
248	Operation requires ownership of all available tokens
249	Tape was changed while assigned
250	Word oriented access not supported for KIND=CD
251	NETBIOS name in use

HISTORYTYPE = 4 (DSEDV), HISTORYCAUSE = 9 (NEWIOERRV)

Integer Value	History Reason (No Mnemonic)
20	Data error—no label
37	Remote backup disk error
38	Unknown station
39	Invalid set of attributes

HISTORYTYPE = 4 (DSEDV), HISTORYCAUSE = 10 (UNIMPLEMENTEDCAUSEV)

Integer Value	Mnemonic Value
1	DYNAMICOWNARRAYV

HISTORYTYPE = 4 (DSEDV), HISTORYCAUSE = 13 (NETWORKCAUSEV)

Integer Value	Mnemonic Value
1	DISCONNECTEDV
5	HOSTNOTREACHABLEV
12	TASKPROTOCOLERRORV

HISTORYTYPE = 4 (DSEDV), HISTORYCAUSE = 14 (SOFTIOERR2CAUSEV)

Integer Value	History Reason (No Mnemonic)
0	WORM file still being written
1	Inappropriate attributes for WORM file
2	Inappropriate FILENAME for WORM file
3	Inappropriate MAXRECSIZE for protected WORM file
4	File and directory name conflict on WORM
5	Too many WORM directory nodes
6	WORM directory node too big
7	Duplicate FILENAME on WORM
8	Security error creating WORM file
9	No room on WORM
10	Nonappend write to WORM file
11	Prior buffer write not at WORM sector boundary

Examples

Suppose that the following task attributes have the values shown:

```
HISTORYTYPE = 4 (DSEDV)
HISTORYCAUSE = 4 (FAULTCAUSEV)
HISTORYREASON = 1
```

In this context, a HISTORYREASON of 1 means DIVIDEBYZEROV. In other words, the process was discontinued because it attempted to divide by zero.

Now suppose that these task attributes have the following values:

```
HISTORYTYPE = 4 (DSEDV)
HISTORYCAUSE = 3 (RESOURCECAUSEV)
HISTORYREASON = 1
```

In this context, a HISTORYREASON of 1 means IOEXCEEDEDV. In other words, the process was discontinued because it used more I/O time than was allowed by its MAXIOTIME task attribute value.

HISTORYTYPE

Type	Mnemonic
Units	Not applicable
Range	See "Explanation" below
Default	NORMALV
Read Time	Anytime
Write Time	Never
Inheritance	None
Overwrite Rules	None (read-only)
Host Services	Supported
Attribute Number	66
Synonym	None
Restrictions	None

Explanation

The HISTORYTYPE indicates the type of termination that occurred for a process. The HISTORYTYPE value is identical to field [07:08] of the HISTORY task attribute. Possible values are as follows:

Mnemonic Value	Integer Value	Meaning
NORMALV	0	The process is still in-use or has not yet been initiated.
DUMPINGV	1	The process is performing a program dump.
QTEDV	2	The process was a print request that was terminated by an operator command.
STEDV	3	The process is suspended.
DSEDV	4	The process was discontinued (terminated abnormally).
NORMALEOTV	5	The process terminated normally.
SYNTAXERRORV	6	The process was a compilation that failed because of syntax errors in the source program.
UNKNOWNEOTV	7	The process was terminated by an unknown cause or by a cause related to job queues.
DSEDINEPILOGV	8	The process was a WFL job whose initiation failed because the job attribute list included an invalid task attribute assignment; or, the process has been discontinued while executing an epilog procedure.

HOSTNAME

Type	String
Units	Not applicable
Range	<simple name>
Default	None
Read Time	Anytime
Write Time	Before initiation
Inheritance	From parent
Overwrite Rules	See below
Host Services	Supported
Attribute Number	64
Synonym	None
Restrictions	None

Explanation

The HOSTNAME task attribute specifies the host system on which the process runs. If HOSTNAME is specified before initiation, the object code file is searched for and initiated on the requested host. If HOSTNAME is read after initiation, it returns the name of the host where the process is running.

For general information about initiating and controlling tasks on remote host systems, refer to the discussion of tasking across multihost networks in the *A Series Task Management Programming Guide*.

Overwrite Rules

Standard overwrite rules apply, except that HOSTNAME task attribute assignments should not be made to an object code file. If HOSTNAME is assigned to an object code file, the process is immediately discontinued as soon as it is initiated.

Run-Time Errors

HOSTNAME ATTRIBUTE INCORRECT SYNTAX

An attempt was made to assign HOSTNAME a value that did not follow the simple name syntax. The assigning process, if nonprivileged, is discontinued with HISTORYCAUSE = 2 (PROGRAMCAUSEV) and HISTORYREASON = 131 (INCORRECTSYNTAXV).

HOSTNAME ATTRIBUTE IS READONLY ON ACTIVE TASK

An attempt was made to assign a HOSTNAME value to an in-use process. The assigning process, if nonprivileged, is discontinued with HISTORYCAUSE = 2 (PROGRAMCAUSEV) and HISTORYREASON = 33 (READONLYONACTIVEV).

ILLEGAL HOST-TO-HOST TRANSFER OF TASK

An attempt was made to initiate a process with a compiled-in HOSTNAME task
attribute value. The initiating process is discontinued with HISTORYCAUSE = 2
(PROGRAMCAUSEV) and HISTORYREASON = 31 (ILLEGALTASKXFERV).

HSPARAMSIZE

Type	Integer
Units	See below
Range	–65535 to +65535
Default	0
Read	Anytime
Write Time	Never
Inheritance	None
Overwrite Rules	None (read-only)
Host Services	Not supported
Attribute Number	70
Synonym	None
Restrictions	None

Explanation

The HSPARAMSIZE task attribute records the total length of the parameters passed to this process. This attribute is mainly intended for use by the system software, but can also be read by application programs.

Units

If the value of HSPARAMSIZE is less than 0, the length is expressed in words. If the value of HSPARAMSIZE is greater than 0, the length is expressed in bytes.

INHERITMCSSTATUS

Type	Boolean
Units	Not applicable
Range	TRUE, FALSE
Default	See below
Read Time	See below
Write Time	See below
Inheritance	None
Overwrite Rules	See below
Host Services	Not supported
Attribute Number	92
Synonym	None
Restrictions	Not available in WFL

Explanation

The INHERITMCSSTATUS task attribute, if TRUE, enables a process to inherit the privileges and priority category of an MCS.

If the initiating process is not an MCS, then the INHERITMCSSTATUS task attribute has no effect.

Note: *Although tasking programs have many of the same privileges as an MCS, the INHERITMCSSTATUS task attribute cannot be used to cause tasking status to be inherited.*

For a discussion of MCS privileges and tasking status, refer to the *A Series Task Management Programming Guide.*

Default

INHERITMCSSTATUS defaults to FALSE for most processes. However, the INHERITMCSSTATUS task attribute defaults to TRUE for internal processes initiated by an MCS.

Read Time, Write Time, and Overwrite Rules

The INHERITMCSSTATUS of a task variable can be read or written at any time, but only by the following types of programs:

- Host Services system software

- Libraries with a nonzero linkage class

- Programs marked with one or more of the following types of security status: compiler status, MCS status, privileged status, or tasking status. (Note that it is the object code file, rather than the process usercode, that must have the special security status.)

For information about linkage classes and about the various types of security status, refer to the *A Series Task Management Programming Guide*.

Although you can assign INHERITMCSSTATUS to object code files, such assignments are ignored when the program is initiated.

Similarly, although you can assign INHERITMCSSTATUS through task equations in CANDE and MARC, such assignments have no effect. INHERITMCSSTATUS task equations in WFL result in the run-time error or warning "INHERITMCSSTATUS ATTRIBUTE – RESTRICTED ACCESS", which is described later under this heading.

Run-Time Error

INHERITMCSSTATUS ATTRIBUTE – RESTRICTED ACCESS

A program lacking the necessary code file privileges attempted to access the INHERITMCSSTATUS task attribute. The accessing process, if nonprivileged, is discontinued with HISTORYCAUSE = 2 (PROGRAMCAUSEV) and HISTORYREASON = 128 (RESTRICTEDACCESSV).

INITPBITCOUNT

Type	Real
Units	Presence-bit operations
Range	0 to about 4.31E68
Default	0
Read Time	Anytime
Write Time	Never
Inheritance	None
Overwrite Rules	None (read-only)
Host Services	Not supported
Attribute Number	104
Synonym	None
Restrictions	None

Explanation

The INITPBITCOUNT task attribute returns the number of initial presence-bit operations that have been performed for the process since its initiation.

For information about initial presence-bit operations, refer to the discussion of controlling process memory usage in the *A Series Task Management Programming Guide.*

INITPBITTIME

Type	Real
Units	See below
Range	0 to about 4.31E68
Default	0
Read Time	Anytime
Write Time	Never
Inheritance	None
Overwrite Rules	None (read-only)
Host Services	Not supported
Attribute Number	105
Synonym	None
Restrictions	None

Explanation

The INITPBITTIME task attribute returns the total time spent processing initial presence-bit operations for this process.

For information about initial presence-bit operations, refer to the discussion of controlling process memory usage in the *A Series Task Management Programming Guide.*

Units

In WFL, this value is returned in units of seconds. In all other languages, this value is returned in units of 2.4 microseconds.

ITINERARY

Type	String
Units	Not applicable
Range	<hostname list>
Default	Null string
Read Time	Anytime
Write Time	Never
Inheritance	See below
Overwrite Rules	None (read-only)
Host Services	Supported
Attribute Number	72
Synonym	None
Restrictions	None

Range

<hostname list>

```
      ┌←──────,──────┐
──────┴─<simple name>─┴──────────────────────────────────┤
```

Explanation

The ITINERARY task attribute contains a record of the remote hosts where ancestors of this process were initiated. The leftmost entry in the string is the hostname of the most recent remote ancestor of the process. The next entry in the string is the hostname of the host where the next most recent remote ancestor was initiated, and so forth.

The default value of null indicates that the process has no remote ancestors.

Inheritance

This attribute is inherited verbatim from parent to offspring when the parent and offspring are running on the same host. When the parent and offspring are on different hosts, the offspring inherits the parent's ITINERARY value with an added entry at the left that records the host where the parent is running.

Examples

The contents of the ITINERARY attribute for four related processes are shown in the following table:

Process	ITINERARY Value of Process
A	"."
B	"BLUE."
C	"BLUE."
D	"YELLOW, BLUE."

The relationship of the processes is as follows:

- Job A starts on host BLUE.

- Job A initiates task B on host YELLOW.

- Task B initiates task C on host YELLOW.

- Task C initiates task D on host RED. Note that RED does not appear in the ITINERARY value for D because the ITINERARY reflects only the ancestors of D.

JOBNUMBER

Type	Integer
Units	Not applicable
Range	0 to 9999
Default	See below
Read Time	Anytime
Write Time	See below
Inheritance	See below
Overwrite rules	Object code file dominant
Host Services	Supported
Attribute Number	41
Synonym	None
Restrictions	None

Explanation

For a task, the JOBNUMBER task attribute records the mix number of the job that owns the task. For a job, the JOBNUMBER task attribute value records the job's own mix number. The mix number is a number that uniquely identifies a process and which the system assigns to the process at initiation. A process can read the mix number by using the MIXNUMBER task attribute.

For further information about mix numbers and relationships between jobs and tasks, refer to the *A Series Task Management Programming Guide*.

Default and Inheritance

The JOBNUMBER value is 0 before initiation. At initiation, a job is automatically assigned a JOBNUMBER value by the system. When you initiate a task from a MARC or CANDE session, the task receives a JOBNUMBER value equal to the session number. All other tasks inherit the JOBNUMBER value of their parents.

Write Time

Only an MCS or other system software can assign the JOBNUMBER value.

Run-Time Errors

The following errors are fatal unless the accessing process is privileged.

JOBNUMBER ATTRIBUTE INCORRECT SYNTAX

An attempt was made to assign JOBNUMBER a value less than 0 or greater than 9999. The assigning process, if nonprivileged, is discontinued with HISTORYCAUSE = 2 (PROGRAMCAUSEV) and HISTORYREASON = 131 (INCORRECTSYNTAXV).

JOBNUMBER ATTRIBUTE INCORRECT SYNTAX

An attempt was made to assign JOBNUMBER a value less than 0 or greater than 9999. The assigning process, if nonprivileged, is discontinued with HISTORYCAUSE = 2 (PROGRAMCAUSEV) and HISTORYREASON = 131 (INCORRECTSYNTAXV).

JOBNUMBER ATTRIBUTE IS READONLY ON ACTIVE TASK

An attempt was made to assign a JOBNUMBER value to an in-use process. The accessing process, if nonprivileged, is discontinued with HISTORYCAUSE = 2 (PROGRAMCAUSEV) and HISTORYREASON = 33 (READONLYONACTIVEV).

JOBNUMBER ATTRIBUTE MAY ONLY BE SET BY AN MCS OR TASKING PROGRAM

A process that was not an MCS or tasking program attempted to assign a value to JOBNUMBER. The assigning process, if nonprivileged, is discontinued with HISTORYCAUSE = 2 (PROGRAMCAUSEV) and HISTORYREASON = 54 (ONLYMCSTASKINGV).

JOBNUMBER IS NOT A SESSIONNUMBER

An attempt was made to assign JOBNUMBER a value that was not a session number. The assigning process, if nonprivileged, is discontinued with HISTORYCAUSE = 2 (PROGRAMCAUSEV) and HISTORYREASON = 11 (NOTSESSIONNUMBERV).
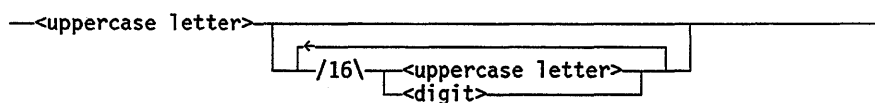
JOBSUMMARY

Type	Mnemonic
Units	Not applicable
Range	See "Explanation" below
Default	DEFAULT
Read Time	Anytime
Write Time	Anytime
Inheritance	None
Overwrite Rules	Standard
Host Services	Supported
Attribute Number	81
Synonym	None
Restrictions	None

Explanation

The JOBSUMMARY task attribute of a job determines whether the job produces a job summary printout. The following are the possible values and their meanings:

Mnemonic Value	Integer Value	Meaning
DEFAULT	0	If the NOSUMMARY option of the OPTION task attribute is set, then the effects are the same as if JOBSUMMARY had a value of CONDITIONAL. If NOSUMMARY is not set, job summary printing is controlled by the JOBSUMMARY option of the PS DEFAULT system command. This PS DEFAULT JOBSUMMARY option can specify a value of CONDITIONAL, SUPPRESSED, UNCONDITIONAL, or ABORTONLY. These values have the same effects as the corresponding JOBSUMMARY task attribute values.
CONDITIONAL	1	The job summary is printed only if one of the following conditions occur: backup files are produced, the job terminates abnormally, or a descendant compilation encounters a syntax error.

continued

continued

Mnemonic Value	Integer Value	Meaning
SUPPRESSED	2	The job summary is suppressed, except in the following circumstances:
		• The job is submitted from an ODT and has WFL syntax errors.
		• The job is discontinued because of a job queue conflict, such as requesting a nonexistent job queue, or specifying job attributes that conflict with job queue attributes.
		Any backup files associated with the job are printed, regardless of whether the job summary prints or not.
UNCONDITIONAL	3	The job summary is printed, regardless of how the job terminates or whether there are backup files.
ABORTONLY	4	The job summary is printed only if the job or one of its descendants terminates abnormally.

The JOBSUMMARY value is not used until the job terminates. If JOBSUMMARY is assigned more than once for an in-use job, only the last assignment before job termination has effect.

When a task initiated through a CANDE or MARC *RUN* command attempts to access its own JOBSUMMARY value, the system actually accesses the JOBSUMMARY value for the session. In other words, for a task initiated by the RUN command from a session, MYSELF.JOBSUMMARY is interpreted as MYJOB.JOBSUMMARY. Any assignments made by the offspring actually affect the job summary for the session.

Similarly, for WFL statements submitted through a CANDE or MARC *WFL* command, MYJOB(JOBSUMMARY) affects the job summary of the session. However, in such WFL statements, MYSELF(JOBSUMMARY) has no effect on the job summary of the session.

In MARC, you can also assign the JOBSUMMARY value for a session by using the MARC *JOBSUMMARY* command.

A task initiated from a job can read or modify its own JOBSUMMARY value. However, for a task, the JOBSUMMARY value has no effect, because a task has no job summary. The JOBSUMMARY value of the task's job determines whether a job summary is produced.

Run-Time Error

JOBSUMMARY ATTRIBUTE INCORRECT SYNTAX

An attempt was made to set JOBSUMMARY to a value less than 0 or greater than 3.
The assigning process, if nonprivileged, is discontinued with HISTORYCAUSE = 2
(PROGRAMCAUSEV) and HISTORYREASON = 131 (INCORRECTSYNTAXV).

JOBSUMMARYTITLE

Type	String
Units	Not applicable
Range	<title>
Default	Null string
Read Time	Anytime
Write Time	Anytime
Inheritance	None
Overwrite Rules	Standard
Host Services	Not supported
Attribute Number	97
Synonym	None
Restrictions	None

Explanation

The JOBSUMMARYTITLE task attribute specifies a title under which the job summary file should be saved.

If JOBSUMMARYTITLE is null when the job terminates, then a job summary file is created only if a job summary is to be printed. The job summary file is titled according to default conventions and is removed immediately after printing.

If the JOBSUMMARYTITLE value has a nonnull value when the job terminates, then the system creates a permanent job summary file with the value of JOBSUMMARYTITLE as its title. The job summary file is created even if no job summary is to be printed. However, even if a job summary is printed, the job summary file is preserved for possible later use.

If the JOBSUMMARYTITLE value includes an *ON* <*family name*> part, then the file is created on the specified family. Otherwise, the location of the job summary file is determined by the rules discussed for printer backup file media in the *A Series Task Management Programming Guide*.

If a statement assigns JOBSUMMARYTITLE a value that does not include a usercode, then the system automatically prefixes the new JOBSUMMARYTITLE value with the usercode under which the job was initiated.

> *Note:* *If the usercode of the job changes after initiation, and the job then assigns JOBSUMMARYTITLE a value that does not include a usercode, the system prefixes JOBSUMMARYTITLE with the original usercode of the job. If you want the job summary file to be created under the new usercode of the job, you must explicitly specify the desired usercode in the JOBSUMMARYTITLE assignment.*

Only a privileged process can assign JOBSUMMARYTITLE a usercode different from the usercode of the process. If a nonprivileged process assigns a usercode to JOBSUMMARYTITLE, the usercode must match the usercode of the process and the usercode of the job of the process. A nonprivileged process running without a usercode cannot assign a usercode to JOBSUMMARYTITLE.

The JOBSUMMARYTITLE attribute has meaning only for jobs. Whenever a task reads its own JOBSUMMARYTITLE value, a null value is returned. If a task assigns a value to its JOBSUMMARYTITLE value, then no error results but the value remains null.

When a task initiated from a CANDE or MARC session attempts to access its own JOBSUMMARYTITLE value, the system actually accesses the JOBSUMMARYTITLE value for the session. In other words, for a task initiated from a session, MYSELF.JOBSUMMARYTITLE is interpreted as MYJOB.JOBSUMMARYTITLE. Any assignments made by the offspring actually affect the job summary for the session. In MARC, you can also assign the JOBSUMMARYTITLE for a session by using the MARC *JOBSUMMARYTITLE* command.

The JOBSUMMARYTITLE value has no effect on the printing of the job summary. For information about controlling job summary printing, and general information about job summaries, refer to the *A Series Task Management Programming Guide*.

Run-Time Errors

JOBSUMMARYTITLE TASK ATTRIBUTE INCORRECT SYNTAX

A process attempted to assign JOBSUMMARYTITLE a value that does not conform
to the syntax of a title. The assigning process, if nonprivileged, is discontinued with
HISTORYCAUSE = 2 (PROGRAMCAUSEV) and HISTORYREASON = 131
(INCORRECTSYNTAXV).

SECURITY VIOLATION

A nonprivileged process attempted to assign JOBSUMMARYTITLE a usercode
that is not allowed for that process. The assigning process is discontinued with
HISTORYCAUSE = 2 (PROGRAMCAUSEV) and HISTORYREASON = 29
(SECURITYERRORV). The message "INVALID TASK ATTRIBUTE:
JOBSUMMARYTITLE" is written in the log.

TASK ATTRIBUTE ACCESS FAULT

A disk error resulted from an attempt to read or assign the JOBSUMMARYTITLE
of a process. The reading or assigning process, if nonprivileged, is discontinued with
HISTORYCAUSE = 2 (PROGRAMCAUSEV) and HISTORYREASON = 132
(ATTACCESSFAULTV).

LANGUAGE

Type	String
Units	Not applicable
Range	<language identifier>
Default	See below
Read Time	Anytime
Write Time	Anytime
Inheritance	See below
Overwrite Rules	Standard
Host Services	Supported
Attribute Number	95
Synonym	None
Restrictions	None

Range

<language identifier>

```
—<uppercase letter>┬─────────────────────────────────────┤
                   └─┬─/16\─┬─<uppercase letter>─┬─┘
                     │      └─<digit>─────────────┘
```

Explanation

The LANGUAGE task attribute is used by the MultiLingual System (MLS) to determine the language of output messages for a process. The LANGUAGE value affects all messages displayed for the process by the system, including BOT, EOT, and RSVP messages. The LANGUAGE value also establishes a default language to be applied to any MESSAGESEARCHER statements executed by ALGOL programs.

You should be especially careful not to misspell the LANGUAGE value, because the system does not notify you of any spelling errors. The system accepts any combination of letters and digits that conforms to the language identifier syntax. If the LANGUAGE value does not correspond to any language that is available on the system, the process messages are displayed in the system default language.

Refer to the *A Series MultiLingual System (MLS) Administration, Operations, and Programming Guide* for information about MLS.

Default and Inheritance

A process inherits the LANGUAGE value of its parent.

The default value of LANGUAGE is ENGLISH. A different default can be established for the whole system by using the LANGUAGE option of the SYSOPS (System Options) system command.

The system administrator can associate a language with a usercode by including a LANGUAGE usercode attribute in the usercode definition in the USERDATAFILE. This language value does not directly affect processes, but it is inherited by MARC or CANDE sessions with that usercode. You can also change the language of a session after log-on by using the MARC or CANDE *LANGUAGE* command. Processes initiated from the session inherit the current language of the session.

The LANGUAGE attribute of a usercode is also inherited by WFL jobs that are assigned that usercode in the job attribute list.

Run-Time Errors

LANGUAGE ATTRIBUTE INCORRECT SYNTAX

An attempt was made to assign a LANGUAGE value that did not conform to the language identifier syntax. The assigning process, if nonprivileged, is discontinued with HISTORYCAUSE = 2 (PROGRAMCAUSEV) and HISTORYREASON = 131 (INCORRECTSYNTAXV).

TOO MANY LANGUAGES IN USE BY SYSTEM

An attempt was made to assign a language value that would bring the total number of languages in use on the system to greater than 256. The assigning process, if nonprivileged, is discontinued with HISTORYCAUSE = 2 (PROGRAMCAUSEV) and HISTORYREASON = 99 (FOREIGNTASKINITFAILV).

LIBRARY

Type	String
Units	Not applicable
Range	<library equation>
Default	Null string
Read Time	See below
Write Time	Before initiation
Inheritance	See below
Overwrite Rules	See below
Host Services	Not supported
Attribute Number	74
Synonym	None
Restrictions	None

Range

<library equation>

```
     ┌←─────────────────────┐ ;  ┌─────────────────────────────┐
──────┴─ LIBRARY ─<internal name>─<library attribute assignment>─┴──────┤
```

<internal name>

<simple name>

<library attribute assignment>

```
        ┌←──────────────────┐ ,  ┌─────────────────────────┐
── ( ───┴─<library attribute>── = ─<library attribute value>─┴─ ) ───────┤
```

<library attribute>
<attribute value>

For a list of possible library attributes and their values, refer to the *A Series Task Management Programming Guide*.

Explanation

The LIBRARY task attribute assigns library attributes to a library used by the process. The LIBRARY task attribute overrides any conflicting assignments in the library declaration.

One typical use of this attribute is to cause a user process to call on a different library than it otherwise would. The library to be used can be specified through assignments to the LIBACCESS, FUNCTIONNAME, and TITLE library attributes. Another use of the LIBRARY task attribute is to pass a parameter to the library through the LIBPARAMETER library attribute.

The internal name specified in the LIBRARY value should equal the value of the INTNAME library attribute. This library attribute can be assigned by the user process. If not assigned, this library attribute defaults to the name of the identifier used in the library declaration.

Read Time

The LIBRARY task attribute can be read at any time from ALGOL. However, the value returned is encoded in an internal form that does not resemble the original LIBRARY assignments. The LIBRARY task attribute returns a null value if read from COBOL74 and cannot be read from WFL at all.

Inheritance

Internal processes inherit the LIBRARY value of the parent.

Overwrite Rules

In ALGOL or COBOL74, if the LIBRARY attribute of a task variable is assigned more than once, each assignment is merged with the previous value of the LIBRARY attribute. A library attribute assignment in the existing value is overwritten only in the following cases:

- If the new assignment specifies a different value for the same attribute of the same library.

- If a null string is assigned to LIBRARY. In this case, the LIBRARY value is restored to null.

In WFL, a LIBRARY assignment is merged with the existing LIBRARY value if the assignment includes an asterisk or if the library internal name is a string primary. If no asterisk is included, and the library internal name is a name constant, then the previous LIBRARY value is discarded.

When a process is initiated, the LIBRARY values assigned through assignments to the task variable, object code file assignments, and inheritance from the parent are merged together into a single LIBRARY value. If these sources assign conflicting values to the same library attribute of the same library, then standard overwrite rules determine which library attribute assignment takes precedence.

Examples

The following is an example of a LIBRARY assignment in CANDE and in WFL:

```
RUN OBJECT/DAILY/UPDATE;
    LIBRARY UPDATER (LIBACCESS=BYTITLE, TITLE=OBJECT/UPDATE/MODS);
    LIBRARY GENROUTINES (TITLE=OBJECT/GENROUTINES/TESTVERSION);
```

The following is an example of a LIBRARY assignment in ALGOL:

```
REPLACE T.LIBRARY BY
  "LIBRARY L (LIBACCESS = BYFUNCTION,FUNCTIONNAME=MYSUPPORT);"
  "LIBRARY GENROUTINES (TITLE=OBJECT/GENROUTINES/TEST);" 48"ØØ";
```

The following ALGOL statement resets the LIBRARY value to a null string:

```
REPLACE T.LIBRARY BY 48"ØØ";
```

The following COBOL74 statements assign attributes to two libraries. Both assignments are made to the same task variable, TASK-VAR-1. The second assignment does not overwrite the first assignment, but rather is merged with it:

```
CHANGE ATTRIBUTE LIBRARY OF TASK-VAR-1 TO
   "LIBRARY L (LIBACCESS=BYFUNCTION,FUNCTIONNAME=""MYSUPPORT"");".
CHANGE ATTRIBUTE LIBRARY OF TASK-VAR-1 TO
   "LIBRARY UPDATER (LIBACCESS = BYFUNCTION);".
```

Run-Time Errors

LIBRARY ATTRIBUTE IS READONLY ON ACTIVE TASK

An attempt was made to assign the LIBRARY value of an in-use process. The assigning process, if nonprivileged, is discontinued with HISTORYCAUSE = 2 (PROGRAMCAUSEV) and HISTORYREASON = 33 (READONLYONACTIVEV).

LIBRARY ATTRIBUTE INCORRECT SYNTAX

There were one or more syntax errors in the library attribute assignments in the LIBRARY value. The assigning process, if nonprivileged, is discontinued with HISTORYCAUSE = 2 (PROGRAMCAUSEV) and HISTORYREASON = 131 (INCORRECTSYNTAXV).

LIBRARYSTATE

Type	Real
Units	Not applicable
Range	See "Explanation" below
Default	0
Read Time	Anytime
Write Time	Never
Inheritance	None
Overwrite Rules	None (read-only)
Host Services	Not supported
Attribute Number	108
Synonym	None
Restrictions	None

Explanation

The LIBRARYSTATE task attribute records whether a process is a library. If the process is a frozen library, or if it was initiated by the library linkage mechanism and has not yet frozen, then bit [0:1] of the LIBRARYSTATE value is set to 1. (The library linkage mechanism initiates a library program if a user process attempts to import an object from that library and an instance of the library does not already exist.) If the process is not frozen and was not initiated by the library linkage mechanism, then bit [0:1] of the LIBRARYSTATE value is set to 0.

This task attribute can be useful for programs that are designed to run in either of two ways, as an ordinary process or a library process. The process can read the LIBRARYSTATE value to determine if it was initiated by the library mechanism; if so, the process can execute a FREEZE statement and become a frozen library process. If the process was not initiated by the library linkage mechanism, it can skip the FREEZE statement and take other actions.

Example

An ALGOL program can use a statement such as the following to determine if it was invoked as a library and take appropriate action:

```
IF BOOLEAN(MYSELF.LIBRARYSTATE)
    THEN FREEZE(TEMPORARY)
    ELSE NONLIBACTOR;
```

This IF statement freezes the process if it was invoked as a library. Otherwise, it calls a procedure named NONLIBACTOR, which is declared elsewhere in the program.

LIBRARYUSERS

Type	Integer
Units	User processes
Range	See below
Default	0
Read Time	Anytime
Write Time	Never
Inheritance	None
Overwrite Rules	None (read-only)
Host Services	Not supported
Attribute Number	101
Synonym	None
Restrictions	None

Explanation

For library processes, the LIBRARYUSERS task attribute returns the number of user processes that are currently linked to the library.

When the LIBRARYUSERS value of a permanent library or control library changes to zero, the system causes the exception event of that library.

If LIBRARYUSERS is read for a process that is not a library, it returns a zero.

Range

The value of LIBRARYUSERS is limited to the number of processes that a given system is capable of running. This number is anywhere from about 1000 for a B 5900 system to about 4000 for an A 15 system.

LOCKED

Type	Boolean
Units	Not applicable
Range	TRUE, FALSE
Default	FALSE
Read Time	Anytime
Write Time	Anytime
Inheritance	None
Host Services	Not supported
Attribute Number	17
Synonym	None
Restrictions	Not available in APLB

Explanation

The LOCKED task attribute provides a means to regulate the timing of two or more processes that access a shared object.

If LOCKED has a value of FALSE, then any process can change the value to TRUE and continue normally. However, if LOCKED has a value of TRUE, then any process that attempts to set LOCKED to TRUE stops executing until some other process sets the value of LOCKED to FALSE. If more than one process is waiting to set LOCKED to TRUE, then when another process sets LOCKED to FALSE, one of the waiting processes sets LOCKED back to TRUE and resumes execution. The programmer cannot predict which of the waiting processes resumes execution first. However, the highest priority process has the best chance. The other waiting processes continue to wait until the next time a process sets LOCKED to FALSE.

Implicitly, the LOCKED attribute functions by accessing the available state of a predeclared event. This attribute is used mainly in WFL jobs because they cannot access events directly. For a detailed discussion of events, refer to the *A Series Task Management Programming Guide*.

MAXCARDS

Type	Integer
Units	Punch cards
Range	0 to 549755813887
Default	0 (unlimited)
Read Time	Anytime
Write Time	Anytime
Inheritance	See below
Overwrite Rules	See below
Host Services	Supported
Attribute Number	39
Synonym	PUNCHLIMIT
Restrictions	None

Explanation

The MAXCARDS task attribute specifies the maximum number of cards that can be punched by a process and its descendants. If a process and its descendants attempt to punch more cards than are allowed by this attribute, the process is discontinued.

The system uses two different but complementary methods to keep track of the number of cards that have been punched by a parent process and its descendants:

- For each process, the system maintains a punch count that records the total number of cards that have been punched for all the punch files declared by that process. (This punch count is stored internally and is not visible to the user.) The system updates this punch count whenever the process or any of its descendants writes to a punch file declared by the process. The system discontinues the process if the punch count reaches a greater value than MAXCARDS.

- If a task declares a punch file, and writes to it, the system does not update the punch count for the parent of the task. However, when the task terminates, the system subtracts the task's punch count from the parent's MAXCARDS value and updates the parent's MAXCARDS value accordingly. The system discontinues the parent if the new MAXCARDS value is lower than the parent's punch count.

Range

If a value less than 0 is assigned, the value is changed to 0. If a value greater than the maximum value is assigned, the value is changed to the maximum value, 549755813887.

Inheritance

A process inherits its parent's MAXCARDS value if the parent's MAXCARDS value is not unlimited.

If a job queue has a default value for the CARDS queue attribute, then that value is inherited by the MAXCARDS task attribute of WFL jobs run from that queue.

If a job queue has a limit value for the CARDS queue attribute, then WFL jobs that specify a higher MAXCARDS value in the job attribute list cannot be accepted into that job queue.

Overwrite Rules

Standard overwrite rules apply, with the following exceptions:

- When a task is initiated, the MAXCARDS value is the minimum of the value inherited from the parent and any value resulting from standard overwrite rules.

- For MAXCARDS assignments to an in-use process, the maximum value that can result is the job's current MAXCARDS value, minus the number of cards the in-use process has already punched. Attempts to assign a higher value result in this maximum value being assigned. No error or warning is issued.

Run-Time Error

PUNCH LIMIT EXCEEDED

The process attempted to punch more cards than were allowed by the MAXCARDS value. The process is discontinued with HISTORYCAUSE = 3 (RESOURCECAUSEV) and HISTORYREASON = 4 (PUNCHEXCEEDEDV).

MAXIOTIME

Type	Real
Units	Seconds
Range	0 through 1319412.9 (15 days, 6 hours, 30 minutes, 12.9 seconds)
Default	0 (unlimited)
Read Time	Anytime
Write Time	Anytime
Inheritance	See below
Overwrite Rules	See below
Host Services	Supported
Attribute Number	5
Synonym	IOTIME
Restrictions	None

Explanation

The MAXIOTIME task attribute specifies the maximum amount of I/O time that a process can use. When the ACCUMIOTIME task attribute reaches the same value as the MAXIOTIME task attribute, the process is discontinued.

When a task terminates, the system decrements the MAXIOTIME value of the task's parent by the amount of I/O time recorded by the ACCUMIOTIME attribute of the task. Refer to the ACCUMIOTIME description in this section for details.

Inheritance

A process inherits the MAXIOTIME value of its parent.

If a job queue has a default value for the IOTIME queue attribute, then that value is inherited by the MAXIOTIME task attribute of WFL jobs run from that queue.

If a job queue has a limit value for the IOTIME queue attribute, then WFL jobs that specify a higher MAXIOTIME value in the job attribute list cannot be accepted into that job queue.

Overwrite Rules

Standard overwrite rules apply, with the following exceptions:

- When a task is initiated, the MAXIOTIME value is the minimum of the value inherited from the parent and any value resulting from standard overwrite rules.

- For MAXIOTIME assignments to an in-use process, the maximum value that can result is the job's current MAXIOTIME value, minus the amount of I/O time the process has already used. Attempts to assign a higher value result in this maximum value being assigned. No error or warning is issued.

Run-Time Errors

EXC I/O TIME

The process used more I/O time than is allowed by the MAXIOTIME task attribute. The process is discontinued with HISTORYCAUSE = 3 (RESOURCECAUSEV) and HISTORYREASON = 1 (IOEXCEEDEDV).

TASK ATTRIBUTE ACCESS FAULT

An attempt was made to assign MAXIOTIME a value greater than its maximum value. The assigning process, if nonprivileged, is discontinued with HISTORYCAUSE = 2 (PROGRAMCAUSEV) and HISTORYREASON = 39 (INFANTICIDEV).

MAXLINES

Type	Integer
Units	Lines printed
Range	0 to 274877906943
Default	0 (unlimited)
Read Time	Anytime
Write Time	Anytime
Inheritance	See below
Overwrite Rules	See below
Host Services	Supported
Attribute Number	40
Synonym	PRINTLIMIT
Restrictions	None

Explanation

The MAXLINES task attribute specifies the maximum number of lines that can be
printed by a process and its descendants. If a process and its descendants attempt to
print more lines than are allowed by this attribute, the process is discontinued.

The system uses two different, complementary methods to keep track of the number of
lines that have been printed by a parent process and its descendants.

1. For each process, the system maintains a print count that records the total number
    of lines that have been printed for all the printer files declared by that process.
    (This print count is stored internally and is not visible to the user.) The system
    updates this print count whenever the process or any of its descendants writes to a
    print file declared by the process. The system discontinues the process if the print
    count reaches a value greater than MAXLINES.

2. If a task declares a print file and then writes to it, the system does not update the
    print count for the parent of the task. However, when the task terminates, the
    system subtracts the task's print count from the parent's MAXLINES value and
    updates the parent's MAXLINES value accordingly. The system discontinues the
    parent if the new MAXLINES value is lower than the parent's print count.

Range

If a value less than 0 is assigned, the value is changed to 0. If a value greater than
274877906943 is assigned, the value is changed to 274877906943.

Inheritance

A process inherits its parent's MAXLINES value if the parent's MAXLINES value is not
unlimited.

If a job queue has a default value for the LINES queue attribute, then that value is inherited by the MAXLINES task attribute of WFL jobs run from that queue.

If a job queue has a limit value for the LINES queue attribute, then WFL jobs that specify a higher MAXLINES value in the job attribute list cannot be accepted into that job queue.

Overwrite Rules

Standard overwrite rules apply, with the following exceptions:

- When a task is initiated, the MAXLINES value is the minimum of the value inherited from the parent and any value resulting from standard overwrite rules.

- For MAXLINES assignments to an in-use process, the maximum value that can result is the job's current MAXLINES value, minus the number of lines the in-use process has already written. Attempts to assign a higher value result in this maximum value being assigned. No error or warning is issued.

Run-Time Error

PRINT LIMIT EXCEEDED

The process attempted to print more lines than were allowed by the MAXLINES value. The process is discontinued with HISTORYCAUSE = 3 (RESOURCECAUSEV) and HISTORYREASON = 3 (PRINTEXCEEDEDV).

MAXPROCTIME

Type	Real
Units	Seconds
Range	0 through 1319412.9 (15 days, 6 hours, 30 minutes, 12.9 seconds)
Default	0 (unlimited)
Read Time	Anytime
Write Time	Anytime
Inheritance	See below
Overwrite Rules	See below
Host Services	Supported
Attribute Number	4
Synonym	None
Restrictions	None

Explanation

The MAXPROCTIME task attribute specifies the maximum amount of processor time that can be used by a process. If the ACCUMPROCTIME task attribute value reaches the same value as MAXPROCTIME, the process is discontinued.

When a task terminates, the system decrements the MAXPROCTIME value of the task's parent by the amount of processor time recorded by the ACCUMPROCTIME attribute of the task. Refer to the ACCUMPROCTIME description in this section for details.

Inheritance

A process inherits its parent's MAXPROCTIME value if the parent's MAXPROCTIME value is not unlimited.

If a job queue has a default value for the PROCESSTIME queue attribute, then that value is inherited by the MAXPROCTIME task attribute of WFL jobs run from that queue.

If a job queue has a limit value for the PROCESSTIME queue attribute, then WFL jobs that specify a higher MAXPROCTIME value in the job attribute list cannot be accepted into that job queue.

Overwrite Rules

Standard overwrite rules apply, with the following exceptions:

- When a task is initiated, the MAXPROCTIME value is the minimum of the value inherited from the parent and any value resulting from standard overwrite rules.

- For MAXPROCTIME assignments to an in-use process, the maximum value that can result is the job's current MAXPROCTIME value, minus the amount of processor time the process has already used. Attempts to assign a higher value result in this maximum value being assigned. No error or warning is issued.

Run-Time Errors

EXC PROC TIME

The process used more processor time than is allowed by the MAXPROCTIME task attribute. The process is discontinued with HISTORYCAUSE = 3 (RESOURCECAUSEV) and HISTORYREASON = 0 (PROCESSEXCEEDEDV).

TASK ATTRIBUTE ACCESS FAULT

An attempt was made to assign MAXPROCTIME a value greater than its maximum value. The assigning process, if nonprivileged, is discontinued with HISTORYCAUSE = 2 (PROGRAMCAUSEV) and HISTORYREASON = 39 (INFANTICIDEV).

MAXWAIT

Type	Real
Units	Seconds
Range	0 to about 4.31E68
Default	0 (Unlimited)
Read Time	Anytime
Write Time	Anytime; effective only if in use
Inheritance	See below
Overwrite Rules	Standard
Host Services	Supported
Attribute Number	49
Synonym	None
Restrictions	None

Explanation

The MAXWAIT task attribute specifies the maximum number of seconds a process can wait on a BDMS *LOCK* or *SECURE* statement in a program. Each of these BDMS statements finds and locks a record in a Data Management System II (DMSII) data set. These statements cause a process to wait if the requested record is currently locked by another process.

Note: *The default MAXWAIT value of 0 specifies an unlimited wait. However, an explicit assignment of 0 to MAXWAIT specifies that the process is not to wait at all.*

If the time limit specified by MAXWAIT is exceeded, the LOCK operation fails and the database status word stores a DMERROR of DEADLOCK and a DMERRORTYPE of 2.

For information about the BDMS *LOCK* and *SET* statements, refer to the appropriate programming language reference manuals. For the format of the database status word, refer to the *A Series DMSII Application Program Interfaces Programming Guide*.

This attribute should not be confused with the WAITLIMIT task attribute, which specifies the number of seconds a process can wait on an event. Refer to the WAITLIMIT description in this section for details.

Inheritance

A task inherits the MAXWAIT value of its parent if the parent's MAXWAIT value is not unlimited.

Example

The following is a BDMSALGOL example:

```
MYSELF.MAXWAIT := 6Ø;
LOCK FIRST STUDENT: RSLT;
IF BOOLEAN(RSLT) THEN
    IF RSLT.DMERROR = DEADLOCK THEN
        IF RSLT.DMERRORTYPE = 2 THEN
            DISPLAY("RECORD NOT UPDATED - LOCKED BY ANOTHER PROCESS");
```

In this example, STUDENT is the name of a data set and RSLT is a real variable.

MCSNAME

Type	String
Units	Not applicable
Range	<title>
Default	See below
Read Time	Anytime
Write Time	Never
Inheritance	See below
Overwrite Rules	None (read-only)
Host Services	Supported
Attribute Number	122
Synonym	None
Restrictions	None

Explanation

The MCSNAME task attribute records the name of the message control system (MCS) that initiated this process family, if it was initiated by an MCS. For example, processes initiated from a MARC session have an MCSNAME that refers to COMS. Processes initiated from a CANDE session, even if that session is in a COMS window, have an MCSNAME that refers to CANDE.

The exact spelling of the MCSNAME corresponds to the file name of the MCS object code file. No *ON <family>* part is included. An asterisk (*) might or might not appear at the start of the MCSNAME. For example, the MCSNAME for COMS might be *SYSTEM/COMS*. The MCSNAME for CANDE might be *SYSTEM/CANDE*.

Default

Before a process is initiated, the default MCSNAME value is a null string. When an MCS sets the SOURCESTATION task attribute of a process, the operating system sets the MCSNAME attribute to the name of the setting MCS.

Inheritance

A process inherits the MCSNAME value of its parent.

If a WFL job is initiated from a CANDE or MARC session or from a task descended from such a session, the WFL job inherits the MCSNAME of the session.

MIXNUMBER

Type	Integer
Units	Not applicable
Range	–9999 to 9999
Default	See below
Read Time	Anytime
Write Time	Never
Inheritance	None
Overwrite Rules	None (read-only)
Host Services	Supported
Attribute Number	1
Synonym	STACKNO
Restrictions	None

Explanation

The MIXNUMBER task attribute returns the mix number of a process. The mix number uniquely identifies the process in system messages, log entries, and system commands that affect the process.

A positive MIXNUMBER value indicates an in-use process or a suspended process. A negative value indicates a terminated process. A zero indicates that the process has not yet been initiated.

For a further discussion of mix numbers, refer to the *A Series Task Management Programming Guide*.

Default

The system assigns the MIXNUMBER task attribute of a new process the next available mix number that is not in use.

MYPPB

Type	String
Units	Not applicable
Range	<task equation list>
Default	Null
Read Time	See below
Write Time	Before initiation
Inheritance	None
Overwrite Rules	Standard
Host Services	Not supported
Attribute Number	37
Synonym	TASK
Restrictions	Not available in WFL

Range

<task equation list>

```
 ┌←─────────────── ; ───────────────┐
─┴─┬─<task attribute assignment>─┬──┴────────────────────────────┤
   ├─<file equation>─────────────┤
   ├─<library equation>──────────┤
   └─<database equation>─────────┘
```

<task attribute assignment>
<file equation>
<database equation>
<library equation>
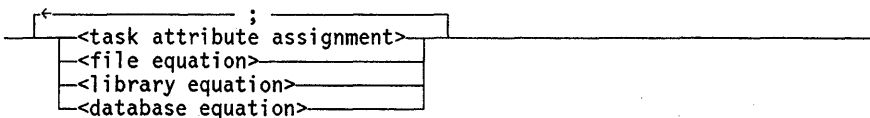
For the syntax of these assignments, refer to the *A Series Work Flow Language (WFL) Programming Reference Manual*.

Explanation

The MYPPB task attribute provides temporary storage for task equations that are not meant to take effect immediately.

The MYPPB task attribute can be assigned only through calls on the HANDLEATTRIBUTES procedure of the WFLSUPPORT system library. The user program can control the behavior of HANDLEATTRIBUTES through several parameters, including one that enables or disables compiler mode and another that specifies a DISPOSITION.

HANDLEATTRIBUTES can assign task equations intended to be applied to a process, task equations intended to be applied to an object code file, or both. The task equations intended for a process include

- Compiler task equations assigned in compiler mode with a DISPOSITION of AIATTACHV.

- Noncompiler task equations assigned in noncompiler mode with a DISPOSITION of AIATTACHV.

The MYPPB task equations that are intended for a process are applied when one of the following happens:

- The task variable is used in a process initiation statement.

- The APPLYLIST task attribute of the task variable is assigned a value of TRUE.

The task equations intended for an object code file include all noncompiler task equations assigned in compiler mode with a DISPOSITION of AIATTACHV or AIAPPLYV. The system does not apply these task equations, even when APPLYLIST is set to TRUE or the task variable is used in a process initiation statement. However, if the task variable is used to initiate a compiler, the compiler applies these task equations to the object code file it creates.

The HANDLEATTRIBUTES procedure can be invoked repeatedly to make assignments to the MYPPB attribute of the same task variable. In this case, the system merges the task equations provided by each HANDLEATTRIBUTES call with the task equations already stored in MYPPB. If a particular task equation conflicts with an existing task equation, the new task equation overwrites the old one.

For a description of the HANDLEATTRIBUTES procedure, refer to "Using WFLSUPPORT to Access Task Attributes" in Section 1, "Accessing Task Attributes."

Read Time

The MYPPB task attribute can be read at any time from ALGOL or COBOL74. However, the value returned is encoded in an internal form that does not resemble the original MYPPB assignments.

Run-Time Error

MYPPB ATTRIBUTE IS READONLY ON ACTIVE TASK

An attempt was made to assign the MYPPB attribute of an in-use process. The assigning process, if nonprivileged, is discontinued with HISTORYCAUSE = 2 (PROGRAMCAUSEV) and HISTORYREASON = 33 (READONLYONACTIVEV).

NAME

Type	String
Units	Not applicable
Range	<title>
Default	See below
Read Time	Anytime
Write Time	Before initiation
Inheritance	See below
Overwrite Rules	See below
Host Services	Supported
Attribute Number	0
Synonym	None
Restrictions	None

Explanation

The NAME task attribute specifies the name of the process. The name of the process is used in the following ways:

- Before initiating an external process, the initiating process typically assigns the NAME task attribute of the task variable of the external process. The NAME value specifies the title of the object code file that is to be initiated.

- The NAME value appears in messages and log entries generated for the process.

- Guard files can specify that only processes with a given NAME are allowed to access a particular file.

If the NAME task attribute of an external process does not include an *ON <family name>* part, then by default the system searches for the object code file on the family named DISK. However, if the FAMILY task attribute specifies substitute families for DISK, the substitute families are searched instead. Refer to the FAMILY description in this section for details.

Default and Inheritance

An internal process inherits the NAME value of its parent. For an external process, the NAME value defaults to the name of the declared external procedure specified in the initiation statement. For example, in ALGOL the following statements initiate a process whose NAME task attribute is DATADC:

```
TASK T;
PROCEDURE DATADC;
   EXTERNAL;
PROCESS DATADC [T];
```

For an internal process, the NAME value is automatically prefixed with the USERCODE value of the new process at initiation time. If NAME was explicitly assigned a value that included a different usercode at the start, this usercode is overwritten with the usercode of the new process. However, for an external process, NAME can specify an object code file with a different usercode than the process or a nonusercoded object code file.

Overwrite Rules

Standard overwrite rules are applied. However, you should be aware that the WFL *RUN <object code file title>* statement implicitly assigns the specified object code file title to the NAME task attribute. In the same way, a *PROCESS <subroutine identifier>* statement implicitly assigns the subroutine identifier to the NAME task attribute. Any NAME value previously assigned to the task variable is overridden by these implicit assignments. These implicit assignments can, in turn, be overridden by task equations included in the RUN or PROCESS statement. For example, the following WFL job initiates the program (STEVENS)OBJECT/TESTB ON DCOM:

```
?BEGIN JOB JOBBIT;
  TASK T(NAME=(WALLACE)OBJECT/OUTPUT ON DCOM);
  RUN (THELMA)OBJECT/NEWDATA ON DCOM [T];
    NAME = (STEVENS)OBJECT/TESTB ON DCOM;
?END JOB
```

The NAME task attribute is also implicitly assigned an object code file title by a MARC or CANDE *RUN* statement.

Run-Time Errors

NAME ATTRIBUTE INCORRECT SYNTAX

A process attempted to assign NAME a value that did not conform to the syntax of a title. The assigning process, if nonprivileged, is discontinued with HISTORYCAUSE = 2 (PROGRAMCAUSEV) and HISTORYREASON = 131 (INCORRECTSYNTAXV).

NAME ATTRIBUTE IS READONLY ON ACTIVE TASK

An attempt was made to assign a value to the NAME attribute of an in-use process. The assigning process, if nonprivileged, is discontinued with HISTORYCAUSE = 2 (PROGRAMCAUSEV) and HISTORYREASON = 33 (READONLYONACTIVEV).

NOJOBSUMMARYIO

Type	Boolean
Units	Not applicable
Range	TRUE, FALSE
Default	FALSE
Read Time	Anytime
Write Time	Anytime
Inheritance	None
Overwrite Rules	Standard
Host Services	Not supported
Attribute Number	98
Synonym	None
Restrictions	None

Explanation

The NOJOBSUMMARYIO task attribute specifies whether any information is to be written to the job log. When NOJOBSUMMARYIO is FALSE, log entries recording activities of the job and its tasks are written to the job log. When NOJOBSUMMARYIO is TRUE, no log entries are written to the job log. This setting conserves disk space and I/O time.

A job can change its NOJOBSUMMARYIO value repeatedly during job execution to prevent job logging of selected areas of the job. Whenever the value of NOJOBSUMMARYIO changes from TRUE to FALSE, an entry is made in the job log to indicate that job log information was not written for part of the job.

If NOJOBSUMMARYIO has a value of TRUE at job initiation and is never reset, the job log contains only the BOJ entry or log-on entry.

The job summary information in the job file is used as the source for job summaries that are printed or saved on disk. Thus, any job summary information suppressed by the NOJOBSUMMARYIO attribute does not appear in printouts produced by the JOBSUMMARY attribute or in job summary files created by the JOBSUMMARYTITLE attribute.

NOJOBSUMMARYIO does not prevent information from being written to the system log.

When a task initiated from a CANDE or MARC session attempts to access its own NOJOBSUMMARYIO value, the system actually accesses the NOJOBSUMMARYIO value for the session. In other words, for a task initiated from a session, *MYSELF.NOJOBSUMMARYIO* is interpreted as *MYJOB.NOJOBSUMMARYIO*. Any assignments made by the offspring actually affect the job summary for the session. In MARC, you can also assign the NOJOBSUMMARYIO value for a session by using the MARC *NOJOBSUMMARYIO* command.

A task initiated from a job can read or modify its own NOJOBSUMMARYIO value. However, for a task the NOJOBSUMMARYIO value has no effect, because a task has no job summary. The NOJOBSUMMARYIO value of the task's job determines whether information from that task is written to the job log.

OPTION

Type	Option list
Units	Not applicable
Range	See "Explanation" below
Default	All options reset
Read Time	Anytime
Write Time	Anytime
Inheritance	See below
Overwrite Rules	Standard
Host Services	Supported
Attribute Number	22
Synonym	OPTIONS
Restrictions	None

Explanation

The OPTION task attribute assigns or returns the values of various options for the process. The options affect program dump contents, job summary printing, handling of backup files, and other areas.

The option value is a single word in which selected bits are associated with particular options. Most of the options have associated mnemonics that can be used to assign that bit. Any combination of options can be set at the same time. The following are the option mnemonics and the effects they have when set:

Option	Meaning
ARRAYS	All arrays of the stack are dumped if a program dump occurs. The ARRAYS option can be abbreviated as ARRAY.
AUTORM	If this option and/or the system option AUTORM is set, then any duplicate library conditions created by the process cause the removal of the old file. For details, refer to the discussion of shared files in the *A Series Task Management Programming Guide*.
BACKUP	If this option and/or the system option LPBDONLY is set, then any printer files created by the process are assigned to a backup disk. Similarly, if this option and/or the system option CPBDONLY is set, then any punch files created by the process are assigned to a backup disk. LPBDONLY and CPBDONLY can be assigned by way of the OP (Options) system command. For details about the effect of this option, refer to the *A Series Task Management Programming Guide*.
BASE	The base of the process stack and the Process Information Block (PIB) are dumped if a program dump occurs.

continued

continued

Option	Meaning
BDBASE	The process assumes some of the characteristics of a job, including the default printing of backup files at termination time. For details, refer to the discussion of interprocess relationships in the *A Series Task Management Programming Guide*.
CODE	The segment dictionary of the task is dumped if a program dump occurs.
DBS	The database stack is dumped if a program dump occurs.
DEBUG	If the process is a COBOL74 or FORTRAN process, it executes special compiled-in debugging code. For details, refer to the *A Series COBOL ANSI-74 Programming Reference Manual, Volume 1: Basic Implementation* and the *A Series FORTRAN Programming Reference Manual*.
DSED	A program dump occurs if the process is terminated by an external cause. For a definition of *external cause*, refer to the *A Series Task Management Programming Guide*.
FAULT	A program dump occurs if the process terminates abnormally from an internal cause. For a definition of *internal cause*, refer to the *A Series Task Management Programming Guide*.
FILES	Information about the files in the stack is dumped if a program dump occurs. The contents of disk file headers are included in the hex information output by this option. The FILES option can be abbreviated as FILE.
LIBRARIES	All libraries associated with the stack are dumped if a program dump occurs. The output from this option includes an analysis of all library-related information, including library templates and directories.
LONG	No arrays are segmented. This option affects only programs written in ALGOL, FORTRAN, or FORTRAN77.
NOSUMMARY	If the JOBSUMMARY task attribute has a value of DEFAULT, then the NOSUMMARY option causes the process to behave as if the JOBSUMMARY task attribute had a value of CONDITIONAL. Refer to the description of the JOBSUMMARY task attribute earlier in this section.
PRESENTARRAYS	Only those arrays that are present in memory are dumped if a program dump occurs. This option reduces the size of a program dump as well as the time the system takes to generate the program dump. Note that if the ARRAYS option is also set, it overrides PRESENTARRAYS and causes all arrays to be dumped. The PRESENTARRAYS option can be abbreviated as PRESENTARRAY.
PRIVATELIBRARIES	Any private libraries used by the process are dumped if a program dump occurs. This option causes an analysis of all library-related information, including library templates and directories.

continued

continued

Option	Meaning
(private process)	The descendants of the process are prevented from altering the task attributes of the process. Any descendant that attempts to assign an attribute of this process is discontinued. This option is typically assigned to message control systems (MCSs) to prevent tasks initiated by sessions from accessing the task attributes of the MCS. There is no mnemonic for this option, which must be assigned by bit number (see the following discussion of bits and their mnemonics).
SIBS	This option has no effect on the Mark 3.6 release and later releases.
SORTLIMITS	Setting this option protects a process that has invoked the SORT facility from being terminated if SORT runs out of memory or disk space. Instead, the process is suspended and the system displays an RSVP message asking the operator to enter an OK (Reactivate) system command to allow SORT to allocate more memory or disk space. For information about the SORT facility, refer to the *A Series System Software Utilities Operations Reference Manual*.
TODISK	Causes any program dumps generated by the process to be directed to a disk file. This is also the default behavior if the operating system option PDTODISK is set. For details about the effects of this option, refer to the *A Series Task Management Programming Guide*.
TOPRINTER	Causes any program dumps generated by the process to be directed to the task file. This is also the default behavior if the operating system option PDTODISK is not set. For details about the effects of this option, refer to the *A Series Task Management Programming Guide*.

The following are the meanings of the various bits in the OPTION value:

Bit	Corresponding OPTION Mnemonic
[47:01]	This bit is set by an OPTION assignment statement in WFL that uses an asterisk (*) to retain the previous option values. (See "Examples" following.)
[24:01]	TOPRINTER
[23:01]	TODISK
[22:01]	SORTLIMITS
[21:01]	DEBUG
[20:01]	PRIVATELIBRARIES
[19:01]	LIBRARIES
[18:01]	SIBS
[15:01]	DBS
[14:01]	private process
[12:01]	NOSUMMARY

continued

continued

Bit	Corresponding OPTION Mnemonic
[11:01]	PRESENTARRAYS, PRESENTARRAY
[10:01]	FILES
[09:01]	CODE
[08:01]	ARRAYS
[07:01]	BASE
[06:01]	BDBASE
[05:01]	AUTORM
[04:01]	BACKUP
[02:01]	DSED
[01:01]	FAULT
[00:01]	LONG

The operator can change the value of the OPTION task attribute with the DUMP (Dump Memory) system command or the DS (Discontinue) system command. Both these commands can include option lists that set program dump options of the OPTION task attribute.

On the other hand, program dump statements in programs do not modify the value of the OPTION task attribute, even if these statements specify dump options. Any dump options specified in a program dump statement thus do not affect later program dumps generated by the process. For information about program dump statements, refer to the *A Series Task Management Programming Guide*.

Inheritance

An internal process inherits the OPTION value of its parent. External processes do not inherit OPTION values.

Examples

The following are ALGOL examples of several methods of setting the OPTION value:

```
100 TVAR.OPTION := 0;
200 TVAR.OPTION := 2**VALUE(ARRAYS) + 2**VALUE(FILES);
300 TVAR.OPTION := 1"10100000000";
400 TVAR.OPTION := 1280;
500 TVAR.OPTION := * & 1[VALUE(ARRAYS):1] & 1[VALUE(FILES):1];
600 TVAR.OPTION := * & 0[VALUE(FILES):1];
```

In this example, the statement at line 100 resets all the options. The statement at line 200 sets the ARRAYS and FILES options and resets all the other options. The statements at lines 300 and 400 have the same effect. The statement at line 500 has a similar effect, except that it does not reset any options that were set previously. The

statement at line 600 resets a single option (FILES) while leaving the other options unchanged. Where the VALUE function is used in these examples, it returns the bit position of the specified mnemonic.

The following COBOL74 example assigns the FAULT option to the OPTION task attribute and leaves any other options unchanged. Note that if line 600 were omitted, the example would reset all options except FAULT:

```
100 WORKING-STORAGE SECTION.
200 01  OPTION-WORD  PIC 9(11)  BINARY.
300 01  VALUE-ONE    PIC 9(11)  BINARY  VALUE 1.
400 PROCEDURE DIVISION.
500 P-1.
600     MOVE ATTRIBUTE OPTION OF MYSELF TO OPTION-WORD.
700     MOVE VALUE-ONE TO OPTION-WORD [0:VALUE FAULT:1].
800     CHANGE ATTRIBUTE OPTION OF MYSELF TO OPTION-WORD.
```

The following is an example of an OPTION task attribute assignment in WFL:

```
OPTION = (*,ARRAYS,FILES);
```

This example assigns the ARRAYS and FILES options and leaves unchanged any options that were already set. If the asterisk (*) is not included, then all options are reset except the ones specifically assigned by the statement.

Run-Time Error

NON-OWNER ACCESS OF A PRIVATE TASK

A descendant of a private process has attempted to make an assignment to a task attribute of the private process. (A private process is one whose OPTION task attribute has the "private process" option set.) The descendant process, if nonprivileged, is discontinued with HISTORYCAUSE = 2 (PROGRAMCAUSEV) and HISTORYREASON = 56 (NONOWNERACCESSV).

ORGUNIT

Type	Integer
Units	Not applicable
Range	See "Explanation" below
Default	See below
Read Time	Anytime; accurate after initiation
Write Time	Never
Inheritance	See below
Overwrite Rules	None (read-only)
Host Services	Not supported
Attribute Number	38
Synonym	None
Restrictions	None

Explanation

The ORGUNIT task attribute records the Logical Station Number (LSN) or physical unit number of the unit that initiated this process. For example, for a process initiated by a CANDE *RUN* command, this task attribute records the LSN of the terminal where the RUN command was entered. The offspring of a process also inherit the ORGUNIT value of that process.

The following fields are defined in the ORGUNIT value:

Field	Meaning
[15:01]	If set, the job was started from a remote terminal. If reset, the job was started from another source, such as an ODT, a card reader, or the operating system.
[14:15]	If [15:01] is set, this field contains the LSN of the originating terminal. If [15:01] is reset, this field contains the physical unit number of the originating device.
	The ORGUNIT value is 0 for processes initiated by the ??RUN (Run Code File) primitive system command, for processes initiated by independent runners, and for remote processes.

For details about how to access these fields, refer to "Accessing Task Attributes at the Bit Level" in Section 1, "Accessing Task Attributes."

One typical use of ORGUNIT is to examine bit 15 to determine whether a process was initiated from a remote terminal. The process can use this information to decide whether to open a remote file to communicate with the user. A process can more precisely determine the type of source from which it was initiated by reading the SOURCEKIND task attribute.

If the process was initiated from a remote terminal, it might be useful for the process to read ORGUNIT to extract the LSN. By assigning field [14:15] of the ORGUNIT value

to the STATION task attribute, the process can make it possible to open a remote file at the originating station. (An alternate method of learning the LSN is to read the SOURCESTATION task attribute.)

Note: *The LSN associated with any particular station can change over time. The ORGUNIT value is not updated to reflect such changes. An alternative to ORGUNIT is the SOURCENAME task attribute. SOURCENAME stores the originating station name, which is less volatile than the LSN.*

The physical unit returned by ORGUNIT can be a useful aid to assigning an ODT file, as shown under "Example" in this description.

Default

Before a process is initiated, the default ORGUNIT value is 0. At initiation time, ORGUNIT is automatically assigned the correct value.

Inheritance

A process inherits the ORGUNIT value of its parent.

If a WFL job is initiated from a CANDE or MARC session or from a task descended from such a session, the WFL job inherits the ORGUNIT of the session.

Example

The following ALGOL example shows two uses of ORGUNIT:

```
100 BEGIN
110 FILE TERM (MYUSE=IO,DEPENDENTSPECS=TRUE);
120
130 IF MYSELF.SOURCEKIND = VALUE(REMOTE) THEN
140    BEGIN
150      TERM.KIND := VALUE(REMOTE);
160      MYSELF.STATION := MYSELF.ORGUNIT.[14:15];
170    END;
180 IF MYSELF.SOURCEKIND = VALUE(ODT) THEN
190    BEGIN
200      TERM.KIND := VALUE(ODT);
210      TERM.UNITNO := MYSELF.ORGUNIT.[14:15];
220    END;
230
240 OPEN (TERM);
250 WRITE (TERM, //, "HI, HOW ARE YOU");
260 END.
```

This program examines the SOURCEKIND value to determine whether it was initiated from a remote terminal or an ODT. The statement at line 160 is equivalent to

```
MYSELF.STATION := MYSELF.SOURCESTATION;
```

For more information, refer to the description of the STATION task attribute in this section.

If the program was initiated at an ODT, the statement at line 210 assigns the physical unit number of the ODT to the UNITNO file attribute. This assignment allows the file to be automatically opened at the ODT, and saves the operator from having to enter a LABEL (Label ODT) system command. Note that this statement is not necessary if the MYUSE file attribute value is OUT instead of IO or IN. For further information about ODT files, refer to the *A Series I/O Subsystem Programming Guide*.

Note also that use of the UNITNO file attribute is restricted on systems running InfoGuard security enhancement software at the S2 level; refer to the *A Series Security Administration Guide* for details.

OTHERPBITCOUNT

Type	Real
Units	Presence-bit operations
Range	0 to about 4.31E68
Default	0
Read Time	Anytime
Write Time	Never
Inheritance	None
Overwrite Rules	None (read-only)
Host Services	Not supported
Attribute Number	106
Synonym	None
Restrictions	None

Explanation

The OTHERPBITCOUNT task attribute returns the count of noninitial presence-bit operations for the process since its initiation.

For information about noninitial presence-bit operations, refer to the *A Series Task Management Programming Guide*.

OTHERPBITTIME

Type	Real
Units	See below
Range	0 to about 4.31E68
Default	0
Read Time	Anytime
Write Time	Never
Inheritance	None
Overwrite Rules	None (read-only)
Host Services	Not supported
Attribute Number	107
Synonym	None
Restrictions	None

Explanation

The OTHERPBITTIME task attribute returns the total time spent processing noninitial presence-bit actions for this process.

For information about noninitial presence-bit operations, refer to the *A Series Task Management Programming Guide*.

Units

In most languages, this value is returned in units of 2.4 microseconds. However, in WFL this value is returned in units of seconds.

PARTNER

Type	Task
Units	Not applicable
Range	Any task variable
Default	See below
Read Time	Anytime
Write Time	Anytime
Inheritance	None
Overwrite Rules	Standard
Host Services	Supported
Attribute Number	19
Synonym	None
Restrictions	Not available in WFL or APLB

Explanation

The PARTNER task attribute accesses the task variable of the partner process.
The partner process is the one to which control is passed when an ALGOL process
executes a simple CONTINUE statement or a COBOL74 process executes an EXIT
PROGRAM statement. Also, the system automatically continues the partner process of
a synchronous task when the synchronous task terminates.

A process can use the PARTNER task attribute as a means to read or write the task
attributes of the partner process. For example, a process can determine the identity
of the partner process by reading the NAME task attribute of the PARTNER task
attribute. The following is an ALGOL example of such a statement:

```
REPLACE NAMEARR BY MYSELF.PARTNER.NAME;
```

A process can also use the PARTNER task attribute to assign a particular process to
be the partner process. However, setting the PARTNER task attribute to a process
other than the parent is not recommended. Such a practice causes each CONTINUE
statement to use more processor time and also leads to source code that is difficult to
understand and maintain.

If this process does not have a partner, then the PARTNER task attribute is treated as
a reference to the MYSELF task variable. When PARTNER refers to MYSELF, any
attempt to continue the partner process has no effect. Execution simply continues to the
next statement in the same process.

For more information about partner processes, refer to the *A Series Task Management
Programming Guide*. Also, see the description of the PARTNEREXISTS task attribute
in this section.

Default

For an independent process or an asynchronous dependent process, the default value of PARTNER is a reference to MYSELF. However, if the process initiates a synchronous dependent offspring, PARTNER changes to a reference to that offspring.

For a dependent process with no offspring, the default value of PARTNER is usually a reference to the initiator of the process. However, PARTNER defaults to MYSELF for such a process if any of the following conditions are true:

- The parent is a WFL job.

- The process was initiated from a MARC or CANDE session.

- The process is a remote process. That is, it was initiated from one BNA host system and runs on a different host system.

When a dependent process A initiates a dependent offspring B, the PARTNER task attribute of dependent process A remains unchanged. It does not become a reference to dependent offspring B unless explicitly assigned.

Run-Time Errors

ILLEGAL VISIT

A process executed a simple CONTINUE statement and the PARTNER task attribute did not refer to MYSELF or a continuable partner process. The process is discontinued with HISTORYCAUSE = 2 (PROGRAMCAUSEV) and HISTORYREASON = 6 (ILLEGALVISITV).

VISIT NONACTIVE TASK

A process attempted to use a CONTINUE statement to transfer control to a process that is not in use (that is, a process that has terminated or has not yet been initiated). The process that executed the CONTINUE statement is discontinued with HISTORYCAUSE = 2 (PROGRAMCAUSEV) and HISTORYREASON = 5 (VISITNONACTIVEV).

PARTNEREXISTS

Type	Boolean
Units	Not applicable
Range	TRUE, FALSE
Default	FALSE
Read Time	Anytime
Write Time	Never
Inheritance	None
Overwrite Rules	None (read-only)
Host Services	Not supported
Attribute Number	27
Synonym	None
Restrictions	None

Explanation

The PARTNEREXISTS task attribute indicates whether the partner process is a
continuable coroutine. The partner process is the process indicated by the PARTNER
task attribute. PARTNEREXISTS returns a value of TRUE only if all the following
conditions are true:

- The process is a synchronous process.

- The partner process is a separate process whose state is TO BE CONTINUED.

- The partner process is not a WFL job.

For more information about partner processes, refer to the *A Series Task Management
Programming Guide*. Also, see the description of the PARTNER task attribute in this
section.

PRINTDEFAULTS

Type	String
Units	Not applicable
Range	<print specification>
Default	Null string
Read Time	Anytime
Write Time	Anytime
Inheritance	See below
Overwrite Rules	Standard
Host Services	Not supported
Attribute Number	99
Synonym	None
Restrictions	Not readable in WFL

Range

<print specification>

```
  ┌←──────────,──────────┐
──┤ ┌─<print attribute phrase>─┤ ├─────────────────────────────┤
    └─<print modifier phrase>──┘
```

<print attribute phrase>
<print modifier phrase>
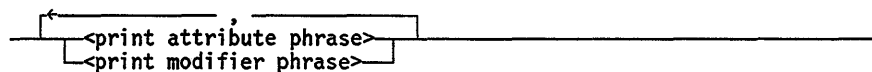
For a discussion of print attributes and print modifiers, refer to the *A Series Print System (PrintS/ReprintS) Administration, Operations, and Programming Guide.*

Explanation

The PRINTDEFAULTS task attribute specifies default values for print attributes and print modifiers. These default values are applied to any print attributes or print modifiers that are not explicitly assigned values by the process.

For an overview of all the task attributes related to printing, refer to the *A Series Task Management Programming Guide.*

Inheritance

A process inherits the PRINTDEFAULTS value of its parent.

If the system administrator has assigned a PRINTDEFAULTS attribute to a usercode, then MARC or CANDE sessions with that usercode receive that PRINTDEFAULTS value at log-on time. You can use the MARC *PRINTDEFAULTS* command to change the PRINTDEFAULTS value of a MARC session. You can use the CANDE *PDEF* command to change the PRINTDEFAULTS value of a CANDE session. Any processes

initiated from a MARC or CANDE session inherit the PRINTDEFAULTS value of the session.

The PRINTDEFAULTS attribute of a usercode is also inherited by WFL jobs that are assigned that usercode in the job attribute list. However, if the job attribute list also contains a PRINTDEFAULTS assignment, the PRINTDEFAULTS attribute of the usercode is ignored.

Examples

The following is an example of a PRINTDEFAULTS assignment in WFL:

```
TVAR (PRINTDEFAULTS = (DESTINATION = "LP4", USERBACKUPNAME = TRUE,
                       SAVEBACKUPFILE = TRUE));
```

The following is an example of a PRINTDEFAULTS assignment in COBOL74:

```
CHANGE ATTRIBUTE PRINTDEFAULTS OF TASK-VAR-1 TO
   "DESTINATION = ""LP4"", USERBACKUPNAME = TRUE.".
```

The following is an example of a PRINTDEFAULTS assignment in ALGOL:

```
REPLACE T.PRINTDEFAULTS BY
  "DESTINATION = """LP4""", USERBACKUPNAME = TRUE,"
  "SAVEBACKUPFILE = TRUE.";
```

Run-Time Errors

PRINTDEFAULTS ATTRIBUTE INCORRECT SYNTAX

A process attempted to assign PRINTDEFAULTS a value that did not conform to the print specification syntax. The assigning process, if nonprivileged, is discontinued with HISTORYCAUSE = 2 (PROGRAMCAUSEV) and HISTORYREASON = 131 (INCORRECTSYNTAXV).

WAITING FOR PRINTSUPPORT TO INITIALIZE

A process attempted to read or assign PRINTDEFAULTS, and the print support library is not available. The process is suspended until the print support library initializes. For information about initializing the print support library, refer to the *A Series Print System (PrintS/ReprintS) Administration, Operations, and Programming Guide.*

PRIORITY

Type	Integer
Units	Not applicable
Range	0 to 99
Default	50
Read Time	Anytime
Write Time	See below
Inheritance	See below
Overwrite Rules	See below
Host Services	Supported
Attribute Number	3
Synonym	DECLAREDPRIORITY
Restrictions	None

Explanation

The PRIORITY task attribute specifies the degree of precedence this process has when competing with other processes for system resources. In general, the higher the number assigned to PRIORITY, the faster the process runs.

The priority of a process is affected by other factors in addition to the PRIORITY task attribute. For details, refer to the discussion of priority in the *A Series Task Management Programming Guide*.

If a limit value is set for the PRIORITY attribute of a job queue, then WFL jobs that specify a higher PRIORITY value in the job attribute list cannot be accepted into that job queue.

For PRIORITY assignments made before initiation, for example, in the job attribute list of a WFL job, only values 1 through 99 are effective. A PRIORITY assignment of 0 is converted to the default of 50 at process initiation, unless inheritance or overwrite rules result in a different value.

Write Time

The PRIORITY task attribute can be assigned a value at any time. The PRIORITY value reflects assignments made after initiation; however, such assignments do not change the actual priority of the process. Only a PR (Priority) system command can effectively change the PRIORITY task attribute value after initiation.

Inheritance

A process inherits the PRIORITY value of its parent.

At the start of any CANDE or MARC session, CANDE or MARC reads the USERDATAFILE to determine if the usercode of the session has a PRIORITY attribute defined for it. If so, CANDE or MARC stores this PRIORITY value as the priority of the session. The session priority is inherited by processes initiated from that session.

If no PRIORITY attribute is defined for the usercode, the session receives no session priority and processes inherit the default PRIORITY value of 50.

If the PRIORITY usercode attribute value is changed after the start of the session, the session priority remains unchanged.

The PRIORITY attribute of a usercode is also inherited by WFL jobs that are assigned that usercode in the job attribute list, or that inherit the terminal usercode of an ODT.

If a default value is set for the PRIORITY job queue attribute, then that value is inherited by the PRIORITY task attribute of WFL jobs run from that queue.

Overwrite Rules

Standard overwrite rules apply, with the following exceptions:

- Processes initiated from CANDE or MARC sessions cannot be assigned a PRIORITY value any higher than the session priority (if there is one). If a task equation or code file PRIORITY assignment requests a higher PRIORITY value than the session priority, the process inherits the session priority.

- If a process is initiated from a MARC session, any PRIORITY value stored in the object code file is ignored. The object code file PRIORITY value is ignored even if it is lower than the session priority, or if there is no session priority.

- If a session priority exists for a CANDE session, then any PRIORITY value stored in the object code file is ignored.

- In general, tasks can be assigned a higher priority than their parents. However, descendants of WFL jobs cannot be assigned a higher PRIORITY value than the PRIORITY job queue limit (if there is one). When the descendant task is initiated, it receives a PRIORITY value equal to the lower of the following values: the requested PRIORITY value and the job queue PRIORITY limit.

RESOURCE

Type	Resource
Units	Not applicable
Range	<resource list>
Default	Unlimited
Read Time	Never
Write Time	Before initiation
Inheritance	See below
Overwrite Rules	Standard
Host Services	Supported
Attribute Number	53
Synonym	None
Restrictions	Available only in WFL

Range

<resource list>

```
        ┌←─────────────┐    ┌──────────────────┐
 ─ ( ───┼─/1\─ TAPE ───┼─ = ─<tape count>─┘ ) ──────────────────┤
        ├─/1\─ TAPE7 ──┤
        ├─/1\─ TAPE9 ──┤
        └─/1\─ TAPEPE ─┘
```

<tape count>
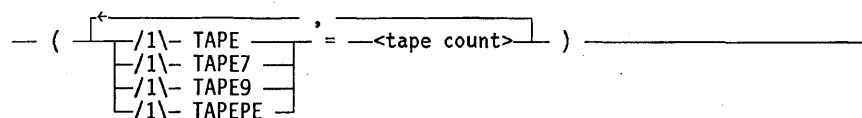
An integer in the range 0 to 255.

Explanation

The RESOURCE task attribute allows the programmer to specify the number and type of tape units required by the process and its descendants. The value TAPE7 refers to 7-track tape, TAPE9 refers to 9-track tape, and TAPEPE refers to phase-encoded tape. The value TAPE includes all kinds of magnetic tapes.

Note: *The values TAPE7, TAPE9, and TAPEPE will be deimplemented*
on a future release. A warning of the change is displayed when a
process uses the RESOURCE task attribute.

If the RESOURCE value of a process requests more tape units of any type than are in the overall tape pool, then process initiation is interrupted and the process appears in the W (Waiting Entries) system command display with a "WAITING FOR RESOURCE" RSVP message. The overall tape pool consists of tape units that have been acquired by the system and have not been opened by any process.

If a RESOURCE value is assigned to a task, then the RESOURCE value is compared with the local tape pool as well as the overall tape pool. The local tape pool is defined by

the most immediate ancestor process for which RESOURCE was explicitly set. The local tape pool is the number of tapes of each type that can be opened by that ancestor process and all its descendants at any given time. The local tape pool is decremented by one when any of these processes opens a tape file and incremented by one when any of them closes a tape file. If a task is initiated with a RESOURCE value that specifies more tape units than are available in the local tape pool, the task appears in the W display with a "WAITING FOR RESOURCE" RSVP message.

However, RESOURCE does not actually impose a limit on the number of tape files a process can attempt to open. A process can have a RESOURCE value of (TAPEPE = 0) and still open a TAPEPE file. The only effect of RESOURCE is to interrupt initiation of a process whose RESOURCE value requests more tapes than are available.

A process in this condition is neither scheduled nor suspended. Initiation is halted at a later stage than it is for a scheduled process, which has only a mix number and a PIB. A process whose initiation was suspended because of a missing tape resource has a mix number, a PIB, and also a process stack. However, the code segment dictionary does not yet exist and execution of the process has not begun.

The RESOURCE task attribute is useful for preventing deadlock conditions. For example, there could be four tape units and two processes, each of which needs to use three tape units. If these processes run simultaneously, and RESOURCE is not set for either one, then the processes might succeed in opening two tape units each. Once this has happened, neither process can proceed any further until the other one is terminated by an operator action, such as a DS (Discontinue) system command.

The RESOURCE task attribute has effect only if the system option RESOURCECHECK is set. This option can be set using the OP (Options) system command. If RESOURCECHECK is reset, then processes are initiated normally regardless of their RESOURCE value.

The RESOURCE task attribute can be accessed only from WFL. It can be assigned to a WFL job in the job attribute list or to tasks by assignments to the task variable or task equations. If RESOURCE is assigned in the job attribute list of a WFL job, then the job cannot be accepted into a job queue with a tape specification that specifies fewer tapes.

Inheritance

A process inherits the RESOURCE value of the closest ancestor that has a RESOURCE value set (if any).

A task cannot be assigned a RESOURCE value higher than what it would inherit from an ancestor. An attempt to assign the task a higher value causes task initiation to fail with the error "TAPE LIMIT EXCEEDED". However, the initiating process continues normally.

Example

The following WFL job includes a RESOURCE assignment for the job as a whole, as well as a RESOURCE assignment for each task. The example is based on the assumption

that the first task needs a maximum of three tapes during its execution, and the second
task needs a maximum of two.

```
?BEGIN JOB TAPEUSER;
    RESOURCE=(TAPE=5);
PROCESS RUN PROG/ONE;
    RESOURCE=(TAPE=3);
PROCESS RUN PROG/TWO;
    RESOURCE=(TAPE=2);
?END JOB
```

Run-Time Error

RESOURCE ATTRIBUTE IS WRITE ONLY

An attempt was made to read the RESOURCE attribute of a process. The
inquiring process, if nonprivileged, is discontinued with HISTORYCAUSE = 2
(PROGRAMCAUSEV) and HISTORYREASON = 129 (ATTWRITEONLYV).

RESTART

Type	Integer
Units	Process restarts
Range	0 to 131071; also see "Range" below
Default	0
Read Time	Anytime; actual after initiation
Write Time	Anytime
Inheritance	None
Overwrite Rules	Standard
Host Services	Not supported
Attribute Number	28
Synonym	None
Restrictions	None

Range

If an attempt is made to assign a value greater than 131071, RESTART is set to 131071. If an attempt is made to assign a value less than 0, RESTART is set to 1.

Explanation

The RESTART task attribute can be used to cause a process to be automatically reexecuted following an abnormal termination. The process is reexecuted if RESTART has a nonzero value at the time of the termination and the termination is due to an internal cause. For a definition of *internal cause*, refer to the *A Series Task Management Programming Guide*.

Reexecution begins with the first statement in the outer block of the process. The value assigned to RESTART determines how many times the process can be reexecuted. The value of RESTART is automatically decreased by 1 after each reexecution of the process.

For related information, refer to the discussion of restarting jobs and tasks in the *A Series Task Management Programming Guide*.

RESTARTED

Type	Boolean
Units	Not applicable
Range	TRUE, FALSE
Default	FALSE
Read Time	Anytime
Write Time	Anytime
Inheritance	None
Overwrite Rules	Standard
Host Services	Not supported
Attribute Number	48
Synonym	None
Restrictions	None

Explanation

The RESTARTED task attribute records whether the process has been restarted.

For a WFL job, RESTARTED is set to TRUE when the job automatically restarts after a halt/load. For a checkpointed process, RESTARTED is set to TRUE when the process is restarted by way of a WFL *RERUN* statement.

The value of RESTARTED is not affected by automatic retries that are caused by the RESTART task attribute. These two task attributes are completely unrelated.

Assigning a value to this attribute has no effect on the process. However, the new value is returned if the RESTARTED value is read later.

For further information, refer to the discussion of restarting jobs and tasks in the *A Series Task Management Programming Guide.*

SAVEMEMORYLIMIT

Type	Real
Units	Words
Range	0 to 274877906943
Default	0 (Unlimited)
Read Time	Anytime
Write Time	Anytime
Inheritance	See below
Overwrite Rules	See below
Host Services	Supported
Attribute Number	112
Synonym	SAVECORELIMIT
Restrictions	None

Explanation

The SAVEMEMORYLIMIT task attribute specifies the maximum amount of save
memory that a process can use. When the amount of save memory in use by a
process exceeds the value of the SAVEMEMORYLIMIT task attribute, the process is
discontinued.

The reason for setting a limit on save memory usage by a process is that save memory
cannot be overlaid. As the proportion of total memory set aside as save memory
increases, it becomes increasingly more difficult for the system to manage memory
efficiently. A process that uses abnormally large amounts of save memory can therefore
have an adverse effect on the performance of all other processes running on the system.
By setting a SAVEMEMORYLIMIT for a process, you can prevent this from happening.

Inheritance and Overwrite Rules

At initiation, a process receives a SAVEMEMORYLIMIT value that is the minimum
value received from the following sources:

- The parent's SAVEMEMORYLIMIT value

- The SAVEMEMORYLIMIT usercode attribute value, if one is defined for the
  usercode of this process

- The limit value for the SAVEMEMORYLIMIT job queue attribute, if the process is a
  WFL job submitted through a job queue that has such a limit defined

- Any SAVEMEMORYLIMIT value that was assigned to the task variable of the
  process before initiation

- Any SAVEMEMORYLIMIT value that was assigned to the object code file of the
  process before initiation

Note that a SAVEMEMORYLIMIT value of 0 means there is no limit on save memory usage. Thus, when determining the minimum, the system ignores any of these sources that has a zero value.

If none of these sources provides a nonzero value, and the process is a WFL job submitted through a job queue with a default SAVEMEMORYLIMIT value, then the job queue default value is inherited by the process.

Once a process is running, any assignment statements that increase the current SAVEMEMORYLIMIT value are ignored. No error is issued, but the requested change is not made.

Run-Time Error

USER SAVE MEMORY LIMIT EXCEEDED

The process attempted to use more save memory than was allowed by the SAVEMEMORYLIMIT value. The process is discontinued with HISTORYCAUSE = 3 (RESOURCECAUSEV) and HISTORYREASON = 14 (SAVECORELIMITEXCEEDEDV).

SOURCEKIND

Type	Real
Units	Not applicable
Range	See "Explanation" below
Default	See below
Read Time	Anytime; actual after initiation
Write Time	Never
Inheritance	See below
Overwrite Rules	None (read-only)
Host Services	Supported
Attribute Number	47
Synonym	None
Restrictions	None

Explanation

The SOURCEKIND task attribute records the type of device that initiated this process family.

One typical use of this attribute is to help decide what value to assign the KIND file attribute of input files used by a process: REMOTE, ODT, READER, and so on. The SOURCEKIND values, and their associated mnemonics, correspond to several of the possible values of the KIND file attribute.

The following are the possible values and their meanings:

Mnemonic Value	Integer Value	Meaning
(None)	0	There is no device type. For example, the process might have been initiated by a ??RUN (Run Code File) primitive system command, an independent runner, or a device on a remote BNA host system.
ODT	2	Operator display terminal (ODT).
REMOTE	3	Remote terminal.
PAPERREADER	4	Papertape reader.
READER	9	Card reader.

Default

Before a process is initiated, the default SOURCEKIND value is 0.

At initiation time, the system assigns SOURCEKIND the appropriate value. For example, processes initiated from CANDE or MARC sessions receive a SOURCEKIND of 3 (remote terminal).

Inheritance

A process inherits the SOURCEKIND value of its parent.

Example

The following ALGOL statement uses SOURCEKIND to determine what value to assign
to the KIND attribute of a file:

```
IF MYSELF.SOURCEKIND = VALUE(REMOTE)
   THEN TERM.KIND := VALUE(REMOTE)
   ELSE TERM.KIND := VALUE(READER);
```

SOURCENAME

Type	String
Units	Not applicable
Range	<name>
Default	See below
Read Time	Anytime
Write Time	Never
Inheritance	See below
Overwrite Rules	None (read-only)
Host Services	Supported
Attribute Number	121
Synonym	None
Restrictions	None

Explanation

The SOURCENAME task attribute records the name of the unit that initiated this process family. It corresponds to the name of the unit stored in the SOURCESTATION task attribute.

A process originating from a unit is assigned a SOURCENAME applicable to that unit. For example, a process initiated from SC 2 is assigned a SOURCENAME of SC2. A process originating from a remote station is assigned a SOURCENAME of the station designated by the SOURCESTATION task attribute. If the SOURCESTATION task attribute designates an invalid logical station number (LSN), then the system assigns SOURCENAME the value *STATION/LSNnnnn*, where *nnnn* represents the LSN.

If the process family was initiated from a pseudostation, the SOURCENAME returns the name of the pseudostation rather than the physical station. For example, if the process family originated from the COMS window CANDE/3 at physical station ST143, the SOURCENAME is *ST143/CANDE/3*.

Default

Before a process is initiated, the default SOURCENAME value is a null string. At initiation time, the system assigns the appropriate SOURCENAME value to the process. Processes initiated from a MARC or CANDE session receive a SOURCENAME value that records the name associated with the LSN from which the process originated. This SOURCENAME is applied when the MCS sets the SOURCESTATION task attribute.

Inheritance

A process inherits the SOURCENAME value of its parent.

If a WFL job is initiated from a CANDE or MARC session or from a task descended from such a session, the WFL job inherits the SOURCENAME of the session.

Example

The following WFL job runs a program that opens a remote file. The remote file has an internal name of REM. WFL equates the title of REM to the SOURCENAME value. The result is that the remote file is opened at the station where the WFL job originated.

```
?BEGIN JOB;
  RUN OBJECT/PROG;
    FILE REM(TITLE = #MYSELF(SOURCENAME));
?END JOB
```

SOURCESTATION

Type	Real
Units	Not applicable
Range	See "Explanation" below
Default	See below
Read Time	Anytime; actual after initiation
Write Time	See "Explanation" below
Inheritance	See below
Overwrite Rules	Object code file dominant
Host Services	Supported
Attribute Number	45
Synonym	None
Restrictions	None

Explanation

The SOURCESTATION task attribute records the unit that initiated this process family. The SOURCESTATION value is divided into the following fields:

Field	Meaning
[47:01]	This field affects the printing of job summaries by any WFL jobs that are initiated by this process. If the value is 0 (zero), the WFL job summary file is saved under a special title for later printing by a message control system (MCS). The title is built under the *REMLPnn/= directory, where nn is the MCS number of the MCS that controls the LSN specified in field [14:15]. On the other hand, if [47:01] has a value of 1, then the WFL job summary is printed in the normal manner.

> *Note:* *The system ignores a 0 value in this field if the rest of the SOURCESTATION value is also 0. If you wish to save job summaries, but you do not wish to set fields [46:01] or [14:15], you can make this field valid by setting another bit such as [45:01].*

An MCS can write to this field, but cannot read it. Other processes cannot read from or write to this field.

The DESTNAME and DESTSTATION task attributes completely override the effect of this field if they are assigned values. For further information, refer to the descriptions of these attributes in this section.

continued

continued

Field	Meaning
[46:01]	If the value is 0 (zero), the system forwards copies of all process messages to the MCS that controls the LSN specified in field [14:15]. Forwarded messages include DISPLAY messages, "BOT" and "EOT" messages, and so on.

> *Note:* *The system ignores a 0 value in this field if the rest of the SOURCESTATION value is also 0. If you wish to forward process messages, but you do not wish to set fields [47:01] or [14:15], you can make this field valid by setting another bit such as [45:01].*

An MCS can write to this field, but cannot read it. Other processes cannot read from or write to this field.

Field	Meaning
[14:15]	The system stores the physical unit number of the originating unit in this field. If the originating unit is a remote terminal, the controlling MCS typically overwrites this field with the LSN of the originating terminal. This field determines the MCS to which process messages are forwarded (refer to the description of field [46:01]). This field contains 0 (zero) for processes initiated by the ??RUN (Run Code File) primitive system command, by system software, or by a device on a remote BNA host system.

An MCS can read from or write to this field. Other processes can read from this field, but cannot write to it.

For details about how to access these fields, refer to "Accessing Task Attributes at the Bit Level" in Section 1, "Accessing Task Attributes."

> *Note:* *The LSN associated with any particular station can change over time. The SOURCESTATION value is not updated to reflect such changes. An alternative to SOURCESTATION is the SOURCENAME task attribute. SOURCENAME stores the originating station name, which is less volatile than the LSN.*

Only an MCS can make assignments to this task attribute. The MCS can assign this attribute to a process only before the process is initiated.

Though the SOURCESTATION value is divided into fields, the first two fields are not readable. Therefore, a process can read SOURCESTATION in the same way as it reads a simple real-valued task attribute, without attempting to read the individual fields.

A process can only indirectly determine whether the SOURCESTATION value is an LSN or a physical unit number. One method of determining this is for the process to read the SOURCEKIND task attribute value. If SOURCEKIND = 3, then SOURCESTATION is an LSN. Refer to the SOURCEKIND description in this section for details.

Alternatively, the process can read the ORGUNIT value. Field [14:15] of the ORGUNIT value is identical to field [14:15] of the SOURCESTATION value. However, one difference between ORGUNIT and SOURCESTATION is that ORGUNIT has an extra

field, [15:01], that indicates whether the originating unit was a remote terminal. Refer
to the ORGUNIT description in this section for details.

One use for the SOURCESTATION task attribute is to enable tasks of WFL jobs
to open remote files. For an example, refer to the description of the STATION task
attribute in this section.

Default

Before a process is initiated, the default SOURCESTATION value is 0. At initiation
time, the system assigns the appropriate SOURCESTATION value to the process.
Processes initiated from a MARC or CANDE session receive a SOURCESTATION value
that records the LSN associated with the session.

Inheritance

A process inherits the SOURCESTATION value of its parent.

If a WFL job is initiated from a CANDE or MARC session or from a task descended from
such a session, the WFL job inherits the SOURCESTATION of the session.

Run-Time Errors

SOURCESTATION ATTRIBUTE IS READ ONLY ON ACTIVE TASK

An MCS attempted to change the SOURCESTATION value of an in-use process. This
error is not fatal, but the requested change is not made.

SOURCESTATION ATTRIBUTE MAY ONLY BE SET BY AN MCS

A process that was not an MCS attempted to assign a value to SOURCESTATION.
The process, if nonprivileged, is discontinued with HISTORYCAUSE = 2
(PROGRAMCAUSEV) and HISTORYREASON = 52 (ONLYMCSMAYSETV).

STACKHISTORY

Type	String
Units	Not applicable
Range	See "Explanation" below
Default	Null string
Read Time	Anytime
Write Time	Never
Inheritance	None
Overwrite Rules	None (read-only)
Host Services	Supported
Attribute Number	30
Synonym	None
Restrictions	None

Explanation

The STACKHISTORY task attribute stores information about the structure of a process that terminated abnormally. The STACKHISTORY value is stored regardless of whether the termination was caused by a fault or by an operator command.

If the process terminates normally, the STACKHISTORY value is a null string. If the process terminates abnormally, STACKHISTORY stores the address of the statement that was being executed when the process terminated. STACKHISTORY also indicates which procedures had been entered, but had not yet been exited, by storing the addresses of the statements that invoked these procedures.

The STACKHISTORY value has the following format if the program was compiled with the compiler control option LINEINFO set:

```
#SSS:AAAA:Y#(DDDDDDDD),....#SSS:AAAA:Y#(DDDDDDDD).
```

The value has the following format if LINEINFO was not set:

```
#SSS:AAAA:Y,#SSS:AAAA:Y,....#SSS:AAAA:Y.
```

The elements shown in the preceding examples have the following meanings:

Element	Meaning
SSS	The code segment number in hexadecimal form. This field is expanded to four characters, SSSS, for segment numbers greater than 4095.
Colon (:)	A colon, which appears between the code segment number and the code word address, and between the code word address and the code syllable number.
AAAA	The address, in hexadecimal, of the code word within that code segment.

continued

continued

Element	Meaning
Y	The number, in hexadecimal, of the syllable within that code word.
Comma (,)	A comma, which appears after each code address except the rightmost address.
Number sign (#)	A blank space, in most cases. However, the blank space is replaced by an asterisk (*) if the code address refers to MCP code.
Ellipsis (....)	Signifies that the same format is repeated for each address.
DDDDDDDD	The line number where the statement occurs in the source file. This number appears only if the LINEINFO control option was set when the program was compiled.
Period (.)	A period terminates the last address in the STACKHISTORY value.

The addresses are listed in reverse historical order. The first address is of the statement that was being executed when the process terminated. The second address, if any, is of the most recent procedure invocation. Subsequent addresses are of previous procedure invocations.

In some cases, one or more of the code addresses in the STACKHISTORY value might refer to MCP code rather than code in the application program. These references are possible because system functions and I/O operations invoked by an application process implicitly result in calls on MCP procedures, which are executed on the application process stack.

For compilations initiated from CANDE, the LINEINFO option is set by default. The DDDDDDDD parts of the STACKHISTORY value can be compared with the sequence numbers in the source program to determine what statements are referred to.

For compilations initiated from WFL, LINEINFO is reset by default, but the LIST option is set. The source code printout that results includes addresses of the form SSS:AAAA:Y after each statement. The STACKHISTORY value can be compared with these addresses to determine what statements are referred to. Alternatively, the programmer could explicitly set LINEINFO and use the sequence numbers instead.

No value is stored for STACKHISTORY if the process incurs a fault but continues executing normally. For an example of using STACKHISTORY, refer to the *A Series Task Management Programming Guide*.

The length of the STACKHISTORY value varies greatly, depending on the number of procedure invocations that were active when the process terminated and whether the program was compiled with the LINEINFO option set. To avoid task attribute errors, you must be careful to read the STACKHISTORY value into an array large enough to hold that value. The largest possible STACKHISTORY value is 400 words (that is, 2400 characters) long.

Examples

In ALGOL, the following declaration creates an EBCDIC array large enough to hold any STACKHISTORY value:

```
EBCDIC ARRAY STACKH[0:2399];
```

The following ALGOL statement reads the STACKHISTORY value into the array:

```
REPLACE STACKH BY T.STACKHISTORY;
```

Run-Time Error

TASK ATTRIBUTE ACCESS FAULT

An attempt was made to read the STACKHISTORY value into an array that is too short. The reading process is discontinued with HISTORYCAUSE = 2 (PROGRAMCAUSEV) and HISTORYREASON = 39 (INFANTICIDEV).

STACKLIMIT

Type	Integer
Units	Words
Range	0 to 64133
Default	6000
Read Time	Anytime; actual after initiation
Write Time	Anytime; effective before termination
Inheritance	None
Overwrite Rules	Standard
Host Services	Supported
Attribute Number	50
Synonym	None
Restrictions	None

Explanation

The STACKLIMIT task attribute specifies the maximum size to which the process stack can grow. If the process stack exceeds this limit, the system discontinues the process. The system checks the STACKLIMIT value only when performing stack stretches; the system does not consider STACKLIMIT when initiating a process.

For further information about STACKLIMIT, refer to the *A Series Task Management Programming Guide*.

Run-Time Errors

ILLEGAL ATTRIBUTE VALUE - TOO LARGE

An attempt was made to assign STACKLIMIT a value larger than 64133. The process, if nonprivileged, is discontinued with HISTORYCAUSE = 2 (PROGRAMCAUSEV) and HISTORYREASON = 135 (VALUETOOLARGEV).

STACK OVERFLOW

The process stack exceeded the size specified by the STACKLIMIT task attribute. The process is discontinued with HISTORYCAUSE = 3 (RESOURCECAUSEV) and HISTORYREASON = 2 (STACKEXCEEDEDV).

STACKSIZE

Type	Integer
Units	Words
Range	See below
Default	See below
Read Time	Anytime
Write Time	Before initiation
Inheritance	See below
Overwrite Rules	Standard
Host Services	Supported
Attribute Number	7
Synonym	STACK
Restrictions	None

Explanation

The STACKSIZE task attribute provides an estimate of the amount of memory that is required for the process stack. The system inspects this value at initiation to determine the amount of memory to be allocated for the process stack at initiation.

Note that STACKSIZE is not intended to return the current process stack size. This task attribute returns only the stack estimate that was used when the process was initiated.

The programmer can affect process scheduling, or prevent stack stretches, by modifying the STACKSIZE before initiating a process. For more information, refer to the *A Series Task Management Programming Guide*.

Range

STACKSIZE accepts values in the range 0 to 16384. If a higher value is assigned, no error results, but the value is converted to 16384.

Default

STACKSIZE defaults to the value of the revised stack estimate, if there is one. If not, STACKSIZE defaults to the value of the compiler stack estimate. For details, refer to the *A Series Task Management Programming Guide*.

Inheritance

An internal process inherits the STACKSIZE value of its parent. Other processes do not inherit the parent's STACKSIZE.

Run-Time Error

STACKSIZE ATTRIBUTE IS READONLY ON ACTIVE TASK

An attempt was made to assign the STACKSIZE task attribute of an in-use process. The assigning process, if nonprivileged, is discontinued with HISTORYCAUSE = 2 (PROGRAMCAUSEV) and HISTORYREASON = 33 (READONLYONACTIVEV).

STARTTIME

Type	String
Units	Not applicable
Range	<starttime specification>
Default	Null string
Read Time	Never
Write Time	Before initiation
Inheritance	None
Overwrite Rules	See below
Host Services	Not supported
Attribute Number	79
Synonym	None
Restrictions	Available only in WFL

Range

<starttime specification>

```
  ┌─<hour>── : ─<minute>─────────────────────────────────────────→
  └ + ─<hours>── : ─<minutes>─┘

→┬────────────────────────────────────────────────────────────┬
 └ ON ─┬─<month>── / ─<day>── / ─<year>─┤
       ├─<julian date>─────────────────┤
       └ + ─<days>─────────────────────┘
```

The <day>, <days>, <hour>, <hours>, and <month> values are each a 1-digit
or 2-digit number. The <minute>, <minutes>, and <year> values must be 2-digit
numbers. The <julian date> value is a 5-digit number.

Explanation

The STARTTIME task attribute delays initiation of a WFL job until the specified time
and date. The job is compiled immediately, but remains in the job queue until the
specified start time. The job is then eligible for initiation the next time the system
selects a job from that job queue. The STARTTIME task attribute can be assigned only
to WFL jobs.

A relative start time can be specified by preceding the time or date with a plus sign (+).
Thus, a start time of + 2:00 means that the job should be initiated in two hours.

If no date or relative date is included in the STARTTIME value, today's date is assumed.

If a Julian date is used as the STARTTIME value, the first two digits signify the year and
the last three digits signify the day of the year. Thus, 87293 means day 293 of 1987.

Overwrite Rules

STARTTIME can be assigned only in the following ways, which are listed in order from most dominant to least dominant:

1. The STARTTIME of a job in a queue can be assigned or changed by the STARTTIME system command or the CANDE *?STARTTIME* command.

2. A STARTTIME assignment can be appended to the MARC, CANDE, or WFL *START* statement that submits a WFL job.

3. A STARTTIME specification can be included in the job attribute list of a WFL job.

Examples

The following example shows a STARTTIME assignment in a WFL job:

```
?BEGIN JOB;
    STARTTIME = 11:ØØ;
    RUN PROG;
?END JOB
```

The following is an example of a STARTTIME assignment appended to a CANDE, MARC, or WFL *START* statement:

```
START WFL/TEST;STARTTIME = 19:ØØ
```

The following is an example of a *STARTTIME* system command:

```
4698 STARTTIME = 21:ØØ
```

The following examples show some of the possible formats for the STARTTIME value:

```
STARTTIME = 14:33;
STARTTIME = + 2:3Ø;
STARTTIME = 23:15 ON + 1;
STARTTIME = 1Ø:ØØ ON 87Ø14;
```

STATION

Type	Integer
Units	Not applicable
Range	0 to 65535
Default	0
Read Time	Anytime
Write Time	Anytime
Inheritance	See below
Overwrite Rules	Standard
Host Services	Supported
Attribute Number	20
Synonym	INITIATOR
Restrictions	None

Explanation

The STATION task attribute stores the logical station number (LSN) of the station to be assigned any remote files used by this process. If STATION has a nonzero value, then the TITLE file attribute does not affect selection of a station for remote files. If the STATION value is zero, then the TITLE file attribute determines the station that is assigned a remote file.

Note: *The LSN associated with any particular station can change over*
time. The STATION value is not updated to reflect such changes.
An alternative to STATION is the SOURCENAME task attribute.
SOURCENAME stores the originating station name, which is less
volatile than the LSN.

A process can change the STATION value after initiation. Only remote files opened after the change to the STATION value are affected by the new value.

If the STATION value specifies an LSN that does not exist, no error occurs until an attempt is made to open a remote file.

For further information about remote file assignment, refer to the *A Series I/O Subsystem Programming Guide*.

Inheritance

A process inherits the STATION value of its parent.

The STATION attribute of a task initiated from a MARC or CANDE session inherits the negative of the LSN associated with the session. The fact that the value is negative is not a problem; when the process opens the remote file, it is opened at the originating session.

On the other hand, the STATION attribute of a WFL job submitted from a MARC or CANDE session does not inherit the LSN associated with the session. If tasks of the WFL job open remote files, and those tasks do not use the FILENAME file attribute to specify a station, then the STATION task attribute should first be explicitly assigned. The simplest way to do this is to assign STATION the value of the SOURCESTATION task attribute. SOURCESTATION is a read-only task attribute that stores the originating station number. Refer to the SOURCESTATION description in this section for details.

Example

The following task equation can be used to allow a task initiated from a WFL job to open a remote file:

```
RUN TASK/READIT;
  STATION = MYSELF(SOURCESTATION);
```

For a related example, refer to the description of the SOURCENAME task attribute.

Run-Time Error

<file name>: UNKNOWN FILE/STATION
REMOTE BACKUP DISK ERROR:<file name>

These two error messages result if a process attempts to open a remote file, the STATION task attribute is 0, and the FILENAME file attribute does not specify a valid station. The "REMOTE BACKUP DISK ERROR" message also occurs if a process attempts to open a remote file and the STATION task attribute stores a nonzero value that is not a valid LSN. The process is discontinued with HISTORYCAUSE = 9 (NEWIOERRCAUSEV) and HISTORYREASON = 37 (REMOTEBUDISK_EV) or 38 (UNKNOWNSTA_EV).

STATUS

Type	Mnemonic
Units	Not applicable
Range	See "Explanation" below
Default	NEVERUSED
Read Time	Anytime
Write Time	Anytime
Inheritance	None
Overwrite Rules	See below
Host Services	Supported
Attribute Number	12
Synonym	None
Restrictions	None

Explanation

A process can use the STATUS task attribute to read or assign the process state of another process or of itself. Some of the STATUS values can be assigned only at certain times; these limitations are noted in the following table:

Mnemonic Value	Integer Value	Meaning
BADINITIATE	–2	Initiation of the process failed. Setting this attribute to BADINITIATE has no effect. If the current value is BADINITIATE, the only assignment that has an effect is an assignment of NEVERUSED.
TERMINATED	–1	The process completed execution normally or was discontinued. Setting the attribute to TERMINATED discontinues the process with HISTORYCAUSE = 2 (PROGRAMCAUSEV) and HISTORYREASON = 39 (INFANTICIDEV). If the current value is TERMINATED, the only assignment that has an effect is an assignment of NEVERUSED.
NEVERUSED	0	No attempt has yet been made to initiate the process. Assigning this value reinitializes the task variable, restoring all task attribute values to their defaults. Assigning this attribute also releases the save memory for the task variable (100 to 200 words). This value can be set only when the current value is NEVERUSED, TERMINATED, or BADINITIATE. If the current value is NEVERUSED, assignments of any other STATUS values have no effect.
SCHEDULED	1	The process is scheduled. Setting this value has no effect.

continued

continued

Mnemonic Value	Integer Value	Meaning
ACTIVE	2	The process is active. Setting the attribute to ACTIVE has no effect unless the process is suspended. If the process is suspended, then assigning a value of ACTIVE resumes execution of the process. If the process is a frozen permanent library, then assigning a value of ACTIVE thaws the library.
SUSPENDED	3	The process is suspended. Setting the attribute to SUSPENDED suspends the process.
FROZEN	5	The process is a frozen library. Setting this value has no effect.
GOINGAWAY	6	The process is a library that is thawing. Assigning this value to a frozen permanent library causes the library to thaw and prevents any new user processes from linking to the library.

Note that assignments to the STATUS task attribute might not affect the process immediately. For example, a critical block exit error can occur if the STATUS attribute is used to terminate a task and the critical block is exited before that STATUS change has taken effect. The only assignment that has immediate effect is an assignment of NEVERUSED.

Whenever the value of the STATUS task attribute changes, the system causes the EXCEPTIONEVENT of the EXCEPTIONTASK of the process. For further information, refer to the descriptions of these attributes in this section.

Overwrite Rules

The STATUS task attribute is not inherited, and syntax errors result from any attempt to assign the STATUS task attribute to an object code file or to assign STATUS in a task equation. The STATUS task attribute of a task variable that is not in use is either NEVERUSED, TERMINATED, or BADINITIATE. When the system initiates a process, the system automatically overwrites the previous STATUS value and assigns a value that reflects the success or failure of the initiation: ACTIVE, SCHEDULED, or BADINITIATE.

Run-Time Error

INITIATE ACTIVE TASK

An attempt was made to set the STATUS attribute of an in-use process to NEVERUSED. The assigning process, if nonprivileged, is discontinued with HISTORYCAUSE = 2 (PROGRAMCAUSEV) and HISTORYREASON = 3 (INITACTIVETASKV).

STOPPOINT

Type	Real
Units	Not applicable
Range	See "Explanation" below
Default	0
Read Time	Anytime
Write Time	Never
Inheritance	None
Overwrite Rules	None (read-only)
Host Services	Supported
Attribute Number	18
Synonym	None
Restrictions	None

Explanation

The STOPPOINT task attribute reports the point at which the process terminated abnormally. If the abnormal termination was due to a fault, the type of fault is also reported. If the process terminates normally, the STOPPOINT value is zero.

The STOPPOINT value is divided into the following fields:

Field	Meaning
[47:08]	If the process encountered a fault, this field stores the type of fault. The value is the same as that of the HISTORYREASON task attribute when the value of the HISTORYCAUSE attribute is 4 (FAULTCAUSEV). Refer to the description of the HISTORYREASON attribute in this section for a list of these values.
[35:03]	This field stores the index of the syllable in the code word where process execution terminated.
[32:13]	This field stores the index of the word in the code segment where process execution terminated.
[13:01]	If reset, this bit indicates that the process was executing MCP code when it terminated. If set, the process was not executing MCP code at termination.
[12:13]	This field stores the index of the code segment where process execution ended.

For details about how to access these fields, refer to "Accessing Task Attributes at the Bit Level" in Section 1, "Accessing Task Attributes."

The code segment, code word, and code syllable indexes appear in source program listings created by the $SET LIST compiler option. For an example of such a listing, refer to the discussion of process history in the *A Series Task Management Programming Guide.*

Other information about a process that terminates abnormally is recorded in the
STACKHISTORY task attribute, which is described in this section.

SUBSPACES

> *Note:* The SUBSPACES *task attribute has no meaning on systems running the Mark 3.9 system software release or a later release. The system displays a deimplementation warning message when a process attempts to use this attribute.*

SUBSYSTEM

Note: The *SUBSYSTEM task attribute has no meaning on systems running the Mark 3.9 system software release or a later release. The system displays a deimplementation warning message when a process attempts to use this attribute.*

SUPPRESSWARNING

Type	String
Units	Not applicable
Range	<suppresswarning list>
Default	Null string
Read Time	Anytime
Write Time	Anytime
Inheritance	None
Overwrite Rules	See below
Host Services	Not supported
Attribute Number	110
Synonym	None
Restrictions	None

Range

<suppresswarning list>

<warning number>
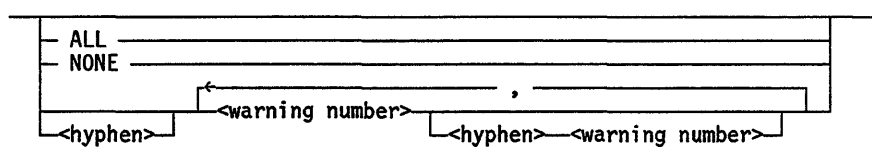
An unsigned integer in the range 1 through 29999.

Explanation

The SUPPRESSWARNING task attribute can be used to suppress run-time warning
messages for a process. Most of these messages are warnings that the process has
just used a feature that is scheduled for deimplementation on a future release. These
messages might not be of interest to a typical user or system operator, and it might be
desirable to suppress their display. A suppressed warning does not appear at the ODT or
in CANDE or MARC sessions. However, it does appear in the system log.

The programmer can suppress particular types of run-time warning messages by
assigning SUPPRESSWARNING a set of warning numbers or warning number ranges.
Each warning number corresponds to a particular run-time warning message. The
warning number for each warning message is included in the text of that message. Thus,
the following message corresponds to warning number 13:

```
WARNING 13: DISK FILE HEADER CHANGES. SEE 3.7 MCP D-NOTE 6638
```

For a list of warning messages and the warning numbers corresponding to them, refer to the *A Series System Messages Support Reference Manual*.

Warning messages can also be suppressed by the system warning suppression value. An operator can use the SUPPRESSWARNING (Suppress Warning) system command to define this value, which affects all processes on the system. A particular warning is suppressed for a process if either the system warning suppression value or the SUPPRESSWARNING task attribute indicates that the warning should be suppressed. However, the system warning suppression value and the SUPPRESSWARNING task attribute value are maintained independently and can be completely different.

Note: *The SUPPRESSWARNING option of the CO (Controller Options) system command affects messages warning of system command deimplementations. This option does not affect run-time warning messages for processes and is not related to the SUPPRESSWARNING task attribute.*

SUPPRESSWARNING can be assigned a list of numbers or number ranges. A number range consists of two numbers separated by a hyphen. For example, assigning a value of *1,3-5* causes warning messages 1, 3, 4, and 5 to be suppressed for the process. If a SUPPRESSWARNING assignment begins with a hyphen, it is interpreted as a minus sign and deletes warning types from the SUPPRESSWARNING list. Thus, if SUPPRESSWARNING has a value of *1,4,8-10*, then an assignment of *-1,9* results in a SUPPRESSWARNING value of *4,8,10*.

The programmer can suppress the display of all run-time warning messages for a process by assigning SUPPRESSWARNING a value of *ALL*. The system translates this into the value *1-29999*, which is returned if a statement reads the attribute thereafter.

The programmer can clear the SUPPRESSWARNING value by assigning a value of *NONE*. If a process reads SUPPRESSWARNING after this assignment, SUPPRESSWARNING returns a null string. In this case, the only warnings suppressed are those specified by the system warning suppression value.

The SUPPRESSWARNING value does not prevent warnings from being recorded by the TASKWARNINGS task attribute. For details, refer to the TASKWARNINGS description in this section.

The SUPPRESSWARNING task attribute value of a library process also affects any user processes while they are executing procedures from that library.

Overwrite Rules

When an ALGOL or COBOL74 program assigns a set of warning numbers to SUPPRESSWARNING, the numbers are added to the current SUPPRESSWARNING value. The system incorporates the warning numbers in ascending order and combines them into ranges where possible. For example, suppose SUPPRESSWARNING has a value of *1,4,8-10*. If a statement assigns a new value of *7,3*, the resulting value is *1,3-4,7-10*. A program can remove warning numbers by assigning SUPPRESSWARNING a value that begins with a hyphen or a value of *NONE*.

However, when SUPPRESSWARNING is assigned from CANDE, MARC, or WFL, the current SUPPRESSWARNING value is discarded and changed to exactly the value assigned. This is true regardless of whether the assignment is made by a task equation, a task attribute assignment to a task variable, or a MODIFY statement that assigns task attributes to an object code file.

Run-Time Error

SUPPRESSWARNING ATTRIBUTE INCORRECT SYNTAX

A process attempted to assign SUPPRESSWARNING a value that did not follow the suppresswarning list syntax. The assigning process, if nonprivileged, is discontinued with HISTORYCAUSE = 2 (PROGRAMCAUSEV) and HISTORYREASON = 131 (INCORRECTSYNTAXV).

SW1 through SW8

Type	Boolean
Units	Not applicable
Range	TRUE, FALSE
Default	FALSE
Read Time	Anytime
Write Time	Anytime
Inheritance	From parent
Overwrite Rules	Standard
Host Services	Supported
Attribute Numbers	84 through 91
Synonym	None
Restrictions	None

Explanation

The eight task attributes named SW1, SW2, SW3, SW4, SW5, SW6, SW7, and SW8 can each be used to store a Boolean value. The values of these task attributes have no effect on the process. They simply serve as holders for any Boolean values the user wishes to store.

You can design processes to communicate with each other by setting and reading these task attributes. For an overview of the use of task attributes in interprocess communication, refer to the *A Series Task Management Programming Guide*.

COBOL74 provides a special syntax for accessing these task attributes. A statement in the SPECIAL-NAMES paragraph can assign special condition names, which can be used later to access the task attribute value.

In RPG, the SW1 through SW8 task attributes can be accessed by way of the external indicators U1 through U8. These external indicators can be used to condition various operations so that they are only performed when the corresponding task attribute is TRUE. For details, refer to the *A Series Report Program Generator (RPG) Programming Reference Manual, Volume 1: Basic Implementation*.

Example

The following COBOL74 program accesses the SW1 task attribute in two ways:

```
100 IDENTIFICATION DIVISION.
110 ENVIRONMENT DIVISION.
120 CONFIGURATION SECTION.
130 SPECIAL-NAMES.
140     SW1 ON STATUS IS SWITCH-ONE-ON,
150         OFF STATUS IS SWITCH-ONE-OFF.
160 DATA DIVISION.
170 WORKING-STORAGE SECTION.
180 PROCEDURE DIVISION.
190 START-HERE SECTION.
200 P1.
210     IF SWITCH-ONE-OFF DISPLAY "SWITCH ONE IS OFF".
220     IF ATTRIBUTE SW1 OF MYSELF = VALUE FALSE
230         DISPLAY "SWITCH ONE IS OFF."
240     STOP RUN.
```

First, SW1 is assigned condition names at lines 140 and 150. The statement at line 210 interrogates the SW1 value by condition name. The statement at line 220 interrogates SW1 by way of the normal task attribute syntax.

TADS

Type	Boolean
Units	Not applicable
Range	TRUE, FALSE
Default	FALSE
Read Time	Anytime
Write Time	Before initiation
Inheritance	See below
Overwrite Rules	Standard
Host Services	Not supported
Attribute Number	94
Synonym	None
Restrictions	None

Explanation

The TADS task attribute invokes the Test and Debug System (TADS) to cause a program to run in test mode. The program must be written in ALGOL, C, COBOL74, COBOL85, or FORTRAN77. The TADS task attribute is ignored unless the program was compiled with the TADS compiler control option set to TRUE.

For directions about how to use TADS, refer to the following manuals:

- *A Series ALGOL Test and Debug System (TADS) Programming Guide*
- *A Series C Test and Debug System (TADS) Programming Reference Manual*
- *A Series COBOL ANSI-74 Test and Debug System (TADS) Programming Guide*
- *A Series COBOL ANSI-85 Test and Debug System (TADS) Programming Reference Manual*
- *A Series FORTRAN77 Test and Debug System (TADS) Programming Guide*
- *A Series NEWP Programming Reference Manual*

Inheritance

An internal process inherits the TADS value of its parent. This inheritance overrides any TADS value explicitly assigned to the internal process. An external process does not inherit the TADS value of its parent.

Run-Time Error

TADS ATTRIBUTE IS READONLY ON ACTIVE TASK

An attempt was made to assign the TADS attribute of an in-use process. The assigning process, if nonprivileged, is discontinued with HISTORYCAUSE = 2 (PROGRAMCAUSEV) and HISTORYREASON = 33 (READONLYONACTIVEV).

TANKING

Type	Mnemonic
Units	Not applicable
Range	See "Explanation" below
Default	UNSPECIFIED
Read Time	Anytime
Write Time	Anytime
Inheritance	None
Overwrite Rules	Object code file dominant
Host Services	Not supported
Attribute Number	60
Synonym	None
Restrictions	None

Explanation

The TANKING task attribute specifies the default tanking mode for remote files used by the process. The system uses this default tanking mode for any remote files whose TANKING file attribute has a value of UNSPECIFIED.

The TANKING task attribute values are as follows:

Mnemonic Value	Integer Value	Meaning
UNSPECIFIED	0	The remote file is not tanked unless the MCS overrides this value when assigning the file.
NONE	1	The remote file is not tanked. The MCS cannot override this value.
SYNC	2	The remote file is tanked. When the remote file is closed, the process does not continue until all tanked output has been completed. The MCS cannot override the SYNC value.
ASYNC	3	The remote file is tanked. The process can continue past the file close, and even terminate, without waiting for the tanked output to be completed. The system continues to transfer messages from the tank file to the output queue until the tank file is empty. The MCS cannot override the ASYNC value.

For further information about tanking, refer to the *A Series Task Management Programming Guide*.

Run-Time Error

TANKING ATTRIBUTE INCORRECT SYNTAX

An attempt was made to assign TANKING a value less than 0 or greater than 3.
The assigning process, if nonprivileged, is discontinued with HISTORYCAUSE = 2
(PROGRAMCAUSEV) and HISTORYREASON = 131 (INCORRECTSYNTAXV).

TARGET

Type	Integer
Units	Not applicable
Range	0 through 1319412
Default	0
Read Time	Anytime
Write Time	Anytime
Inheritance	None
Overwrite Rules	Standard
Host Services	Not supported
Attribute Number	6
Synonym	TARGETTIME
Restrictions	None

Explanation

The TARGET task attribute stores any integer value that is assigned to it by a user. The value of this attribute has no effect on the process, nor does it report any information about the process. Rather, it is provided for use in communicating information between processes. For an overview of the use of task attributes in interprocess communication, refer to the *A Series Task Management Programming Guide*.

The value of TARGET formerly had some effect on process scheduling. The operating system no longer uses this attribute for that purpose.

Run-Time Error

TASK ATTRIBUTE ACCESS FAULT

An attempt was made to assign TARGET a value greater than its maximum. The assigning process, if nonprivileged, is discontinued with HISTORYCAUSE = 2 (PROGRAMCAUSEV) and HISTORYREASON = 39 (INFANTICIDEV).

TASKERROR

Type	Real
Units	Not applicable
Range	See "Explanation" below
Default	0
Read Time	Any time
Write Time	Never
Inheritance	None
Overwrite Rules	None (read-only)
Host Services	Not supported
Attribute Number	117
Synonym	None
Restrictions	None

Explanation

The TASKERROR task attribute indicates whether an error resulted from the most recent attempt to access a task attribute of this process. If an error did result, the TASKERROR value also indicates which task attribute was being accessed, and the type of error that occurred.

The TASKERROR task attribute serves a purpose similar to the ERROR task attribute. However, TASKERROR has the advantage of providing more information about the error that occurred. Another advantage of using TASKERROR is that the value can be read repeatedly, whereas the ERROR value is erased each time it is read.

Further, a program can use the TASKERROR value as input to the ATTRIBUTEMESSAGE procedure, which translates the value into a textual error message. For information about ATTRIBUTEMESSAGE, refer to "Using WFLSUPPORT to Access Task Attributes" in Section 1, "Accessing Task Attributes."

The TASKERROR value is divided into the following fields:

Field	Value	Meaning
[47:08]	0	If field [00:01] is 1, then an error occurred in accessing a task attribute other than FILECARDS or LIBRARY. Field [39:16] stores the number of the task attribute that was used incorrectly.
		If field [00:01] is 0 (zero), then there was no task attribute error.
	1	An error occurred in accessing the FILECARDS task attribute. Field [39:16] stores the number of the file attribute that was assigned incorrectly in FILECARDS. For a list of file attributes in numerical order, refer to the description of the ATTYPE file attribute in the *A Series File Attributes Programming Reference Manual*.
	6	An error occurred in assigning the LIBRARY task attribute. Field [39:16] stores the number of the library attribute that was assigned incorrectly in the LIBRARY task attribute. For a list of library attributes in numerical order, refer to Table 2-5, "Library Attributes by Number."
[39:16]	(Various)	The number of the task attribute, file attribute, or library attribute that was assigned incorrectly. For details about the meaning of this field, refer to the discussion of field [47:08].
[23:16]	(Various)	The error code. If the value is in the range 1 through 999, it corresponds to the HISTORYREASON task attribute value. For an explanation of values in this range, refer to the HISTORYREASON task attribute description in this section.
		If the value is 1000 or greater, it corresponds to the HANDLEATTRIBUTES error number. For an explanation of values in this range, refer to Table 1-2, "HANDLEATTRIBUTES Error Numbers."
[07:05]	0	An unused field. Its value is always 0 (zero).
[02:01]	0	The validity bit. This value indicates that fields [47:08] and [39:16] do not contain valid values.
	1	Fields [47:08] and [39:16] contain valid values.
[01:01]	0	An unused field. Its value is always 0 (zero).
[00:01]	0	The exception bit. This value indicates that no error or warning occurred, and none of the previously defined fields are used.
	1	The exception bit. If 1, an error or warning occurred, and the fields defined previously are used. If 0 (zero), no error or warning occurred, and none of the previously defined fields are used.

For details about how to access these fields, refer to "Accessing Task Attributes at the Bit Level" in Section 1, "Accessing Task Attributes."

Table 2–5. Library Attributes by Number

Number	Name
0	INTNAME
1	TITLE
2	LIBPARAMETER
3	FUNCTIONNAME
4	LIBACCESS
6	FUNCTIONDEFINED
9	LIBERROR
10	LIBVALUE
11	INHERITNWSTATUS

Examples

The following are examples of ALGOL statements that read the values of individual fields of the TASKERROR task attribute. The assignments are all made to real variables (named BUF, GENERAL_TYPE, ATTRIBUTE_NUMBER, and so on).

```
BUF := T.TASKERROR;
GENERAL_TYPE := BUF.[47:8];
ATTRIBUTE_NUMBER := BUF.[39:16];
ERROR_NUMBER := BUF.[23:16];
VALIDITY_BIT := BUF.[2:1];
WARNING_BIT := BUF.[1:1];
EXCEPTION_BIT := BUF.[0:1];
```

The following are examples of COBOL74 statements that read the values of individual fields of the TASKERROR task attribute. The assignments are all made to 77-level variables of type REAL (named BUF, GENERAL-TYPE, ATTRIBUTE-NUMBER, and so on).

```
MOVE ATTRIBUTE TASKERROR OF TASK-VAR-1 TO BUF.
MOVE BUF TO GENERAL-TYPE      [ 47:07:08 ].
MOVE BUF TO ATTRIBUTE-NUMBER  [ 39:15:16 ].
MOVE BUF TO ERROR-NUMBER      [ 23:15:16 ].
MOVE BUF TO VALIDITY-BIT      [ 02:00:01 ].
MOVE BUF TO WARNING-BIT       [ 01:00:01 ].
MOVE BUF TO EXCEPTION-BIT     [ 00:00:01 ].
```

WFL allows TASKERROR to be read as a real value, but does not provide any syntax for reading the individual fields within the TASKERROR value.

Run-Time Errors

TASKERROR ATTRIBUTE IS READONLY

An attempt was made to assign a value to the TASKERROR attribute. The assigning process is discontinued with HISTORYCAUSE = 2 (PROGRAMCAUSEV) and HISTORYREASON = 9 (ATTREADONLYV).

TASKFILE

Type	File
Units	Not applicable
Range	See "Explanation" below
Default	See below
Read Time	Before termination; accurate only while process is in use
Write Time	Never
Inheritance	None
Overwrite Rules	WFL file equations only
Host Services	Not supported
Attribute Number	32
Synonym	None
Restrictions	Not available in APLB

Explanation

The TASKFILE task attribute is used to access the task file associated with a process. The task file is a printer backup file that stores any program dumps generated by that process. For more information about program dumps, refer to the *A Series Task Management Programming Guide*.

The main use of the TASKFILE task attribute is to allow a process to write comments to the task file before the process generates a dump. TASKFILE can also be used in statements that close the task file or interrogate the file attributes of the task file. TASKFILE cannot be used to assign file attributes to the task file, although the FILECARDS task attribute can be used for this purpose. For information about assigning file attributes to the task file, refer to the discussion of process history in the *A Series Task Management Programming Guide*.

A process can access its own task file or the task file of any of its ancestors. For example, a task can access its job's task file by way of the MYJOB task variable and the TASKFILE task attribute.

A process cannot access the task file of any descendant, sibling, or cousin process or of any process outside its own process family.

Default

By default, the TASKFILE attribute defines a file with the following attributes:

```
BACKUPKIND = DISK
BUFFERS = 1
INTMODE = EBCDIC
INTNAME = TASKFILE
KIND = PRINTER
LABELTYPE = OMITTED
MAXRECSIZE = 22
MYUSE = OUT
```

When the task file is opened, the system titles it according to the standard printer backup file titling convention discussed in the *A Series Task Management Programming Guide*.

Examples

The following ALGOL statements cause two program dumps and write a different comment to each program dump. The CLOSE statement causes the program dumps to be stored in two separate backup files:

```
WRITE (MYSELF.TASKFILE,//,"HI THERE, DUMP 1");
PROGRAMDUMP;
CLOSE (MYSELF.TASKFILE);
WRITE (MYSELF.TASKFILE,//,"HI THERE, DUMP 2");
PROGRAMDUMP;
```

The following ALGOL statements interrogate file attributes of the task file:

```
R := MYSELF.TASKFILE.KIND;
IF MYSELF.TASKFILE.OPEN THEN ...
```

Run-Time Errors

The following errors are always fatal, even if the accessing process is privileged, an MCS, or BNA Host Services.

NON ANCESTRAL TASKFILE

A process attempted to access the task file of another process that is not an ancestor of the accessing process. The accessing process is discontinued with HISTORYCAUSE = 2 (PROGRAMCAUSEV) and HISTORYREASON = 12 (NONANCESTRALTASKFILEV).

TASKFILE ATTRIBUTE IS READONLY

An attempt was made to assign a value to TASKFILE. The assigning process is discontinued with HISTORYCAUSE = 2 (PROGRAMCAUSEV) and HISTORYREASON = 9 (ATTREADONLYV).

TASKLIMIT

Type	Integer
Units	Descendant tasks
Range	0 to 31
Default	0 (unlimited)
Read Time	Anytime
Write Time	Anytime
Inheritance	See below
Overwrite Rules	Standard
Host Services	Not supported
Attribute Number	58
Synonym	None
Restrictions	None

Explanation

The TASKLIMIT task attribute limits the number of descendants a job can have. The value of the job's TASKLIMIT is automatically decremented by 1 each time a descendant task is initiated. When TASKLIMIT has been decremented to 0, the initiation of any further descendants causes the initiating process to be discontinued.

The job's TASKLIMIT is also decremented by 1 when an independent process is initiated by the job or one of the job's descendants. However, descendants of the independent process do not affect the original job's TASKLIMIT.

The limit applied by TASKLIMIT is cumulative. That is, it limits the total number of descendants a job can have during its history, not only the number of descendants a job can have at the same time.

If TASKLIMIT has not been set, there is no limit on the number of descendants a job can have, and reading TASKLIMIT returns a value of 0. However, explicitly assigning 0 to TASKLIMIT sets a limit of 0 on the number of descendants.

TASKLIMIT has no effect when assigned to a task. It does not limit the number of descendants the task can have.

Inheritance

If the TASKLIMIT attribute is set for a job queue, it is inherited by all WFL jobs run out of that job queue. This is true even if the WFL job attribute list specifies a different TASKLIMIT value. However, after initiation the WFL job can assign TASKLIMIT a different value.

Run-Time Error

TASKLIMIT EXCEEDED

The process attempted to initiate a task when the TASKLIMIT value of MYJOB was already decremented to 0. The process is discontinued with HISTORYCAUSE = 2 (PROGRAMCAUSEV) and HISTORYREASON = 58 (TASKLIMITEXCEEDEDV).

TASKSTRING

Type	String
Units	Not applicable
Range	\<taskstring specification\>
Default	Null
Read Time	Anytime
Write Time	Before initiation
Inheritance	None
Overwrite Rules	Standard
Host Services	Not supported
Attribute Number	113
Synonym	None
Restrictions	None

Range

\<taskstring specification\>

A string of up to 255 EBCDIC characters of which the last must be a null character.

Explanation

The TASKSTRING task attribute stores any string value that is assigned to it by a user. The value of this attribute has no effect on the process nor does it report any information about the process. Rather, it is provided for use in communicating information between processes.

For an overview of the use of task attributes such as TASKSTRING in interprocess communication, refer to the *A Series Task Management Programming Guide*.

Examples

In ALGOL, the following statement could be used to assign TASKSTRING a value of *$SET LIST*:

```
REPLACE T1.TASKSTRING BY "$SET LIST" 48"ØØ";
```

In COBOL74, the equivalent statement has the following form:

```
CHANGE ATTRIBUTE TASKSTRING OF T1 TO "$SET LIST".
```

In WFL, the assignment appears as follows:

```
T1(TASKSTRING = "$SET LIST");
```

Run-Time Error

TASKSTRING ATTRIBUTE INCORRECT SYNTAX

An attempt was made to assign a TASKSTRING value that was more than 255
characters long or that was not terminated by a null character. The assigning
process is discontinued with HISTORYCAUSE = 2 (PROGRAMCAUSEV) and
HISTORYREASON = 131 (INCORRECTSYNTAXV).

TASKVALUE

Type	Real
Units	Not applicable
Range	-4.31E68 to +4.31E68
Default	0
Read Time	Anytime
Write Time	Anytime
Inheritance	None
Overwrite Rules	Standard
Host Services	Supported
Attribute Number	9
Synonym	VALUE
Restrictions	None

Explanation

The TASKVALUE task attribute stores any real value that is assigned to it by a user. The value of this attribute has no effect on the process nor does it report any information about the process. Rather, it is provided for use in communicating information between processes.

An operator can change the TASKVALUE of an in-use process with the *<mix number> HI <integer>* form of the HI (Cause EXCEPTIONEVENT) system command.

For an overview of the use of task attributes in interprocess communication, refer to the *A Series Task Management Programming Guide*.

TASKWARNINGS

Type	String
Units	Not applicable
Range	<task warnings list>
Default	Null string
Read Time	Only while in-use
Write Time	Never
Inheritance	None
Overwrite Rules	None (read-only)
Host Services	Not supported
Attribute Number	109
Synonym	None
Restrictions	None

Range

<task warnings list>
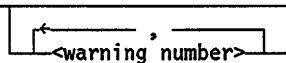
Explanation

The TASKWARNINGS task attribute records what run-time warning messages have been issued for the object code file used by the process. Most run-time warning messages notify the programmer that the process uses a feature that has been scheduled for deimplementation on a future release. A programmer can use these messages to determine what changes need to be made to a program so it can be run on a new release.

The TASKWARNINGS value is the same as the value of the WARNINGS file attribute of the object code file. The value consists of either a null string or a series of warning numbers. Each warning number represents a particular run-time warning message. The warning number for each warning message is included in the text of that message. Thus, the following message corresponds to warning number 13:

```
WARNING 13: DISK FILE HEADER CHANGES. SEE 3.7 MCP D-NOTE 6638
```

For a list of warning messages and the warning numbers corresponding to them, refer to the *A Series System Messages Support Reference Manual.*

In the TASKWARNINGS value, warning numbers are separated by commas and listed in ascending order.

The TASKWARNINGS value includes warnings that were issued for other processes that were instances of this same object code file, or that were executing a

procedure from this object code file when the warning occurred. For example, the TASKWARNINGS attribute of a library process reflects any warnings that were issued for user processes while they were executing procedures exported by the library.

If the TASKWARNINGS attribute of the MYSELF task variable is read within a library program, it returns either the warnings stored in the library object code file or those stored in the user process object code file, depending on the context. If TASKWARNINGS is read in an exported library procedure, it returns warnings stored in the user process object code file. If TASKWARNINGS is read elsewhere in the library, such as by a statement executed before the library freezes, then TASKWARNINGS returns warnings stored in the library object code file.

The TASKWARNINGS value includes all warnings that were issued for this object code file, including any that were suppressed by the SUPPRESSWARNING task attribute or the system warning suppression value. For details, refer to the description of SUPPRESSWARNING in this section.

Run-Time Error

CODE FILE MUST BE ACTIVE

An attempt was made to read the TASKWARNINGS task attribute of a process that is not in use. The accessing process, if nonprivileged, is discontinued with HISTORYCAUSE = 2 (PROGRAMCAUSEV) and HISTORYREASON = 116 (CODEFILENOTACTIVEV).

TEMPFILELIMIT

Type	Real
Units	Disk Megabytes
Range	–1 to about 4.31E68
Default	–1 (Unlimited)
Read Time	Anytime
Write Time	Anytime
Inheritance	From parent
Overwrite Rules	See below
Host Services	Supported
Attribute Number	118
Synonym	None
Restrictions	None

Explanation

The TEMPFILELIMIT task attribute specifies the maximum amount of disk space that can be allocated at one time to temporary disk files owned by the process. For an introduction to temporary files, refer to the *A Series Task Management Programming Guide.*

The amount of disk space used for temporary files increases when a process creates a new temporary disk file or increases the size of an existing temporary disk file. If one of these operations causes the process to exceed the TEMPFILELIMIT value, the system issues an I/O error for the process, and the I/O operation is not performed.

The TEMPFILELIMIT value is enforced only when the disk resource control system is active, and is never enforced for library maintenance processes.

TEMPFILELIMIT returns a value of -1 if it is read and no value was previously assigned to it. A value of -1 means that there is no limit on temporary file usage.

For more information about the disk resource control system, refer to the *A Series Disk Subsystem Administration and Operations Guide.* A related task attribute, TEMPFILEMBYTES, is discussed later in this section.

Overwrite Rules

When a process is initiated, the system assigns a TEMPFILELIMIT value that is the minimum of the following values:

- The value of the TEMPFILELIMIT usercode attribute, if this attribute has been defined for the usercode of this process

- The TEMPFILELIMIT value inherited from the parent, as long as it is not unlimited (–1)

- Any TEMPFILELIMIT value that would result from standard overwrite rules, for example, because of a previous TEMPFILELIMIT assignment to the task variable or the object code file

Once a process is running, the current value of TEMPFILELIMIT can never be increased. An assignment that attempts to increase the TEMPFILELIMIT value is ignored and the TEMPFILELIMIT value remains unchanged. On the other hand, TEMPFILELIMIT can be assigned a lower value at any time.

Run-Time Error

FILE <file name> I/O ERROR: ATTEMPT TO EXCEED TEMPORARY FILE LIMIT

The process requested more space for temporary disk files than was allowed by the TEMPFILELIMIT attribute. The I/O operation fails. If the process has specified error handling for the I/O statement that caused the error, then the process can proceed normally. Otherwise, the process is discontinued.

TEMPFILEMBYTES

Type	Real
Units	Disk megabytes
Range	0 to about 4.31E68
Default	0
Read Time	Anytime
Write Time	Never
Inheritance	None
Overwrite Rules	None (read-only)
Host Services	Supported
Attribute Number	119
Synonym	None
Restrictions	None

Explanation

The TEMPFILEMBYTES task attribute records the amount of disk space currently allocated to temporary files owned by the process. For an introduction to temporary files, refer to the *A Series Task Management Programming Guide*.

The information returned by TEMPFILEMBYTES is valid only when the disk resource control system is active and is continuously active during the entire life of the process. Temporary files created by library maintenance processes and certain other system functions are not included in the value returned by this attribute.

For more information about the disk resource control system, refer to the *A Series Disk Subsystem Administration and Operations Guide*.

Run-Time Error

TEMPFILEMBYTES IS READONLY

A process attempted to assign a value to the TEMPFILEMBYTES attribute. The assigning process, if nonprivileged, is discontinued with HISTORYCAUSE = 2 (PROGRAMCAUSEV) and HISTORYREASON = 9 (ATTREADONLYV).

TYPE

Type	Mnemonic
Units	Not applicable
Range	See "Explanation" below
Default	PROCESS
Read Time	Anytime; accurate while in use
Write Time	Never
Inheritance	None
Overwrite Rules	None (read-only)
Host Services	Supported
Attribute Number	11
Synonym	None
Restrictions	None

Explanation

The TYPE task attribute returns information about whether the process is synchronous or asynchronous, and dependent or independent. The following are the possible values and their meanings:

Mnemonic Value	Integer Value	Meaning
PROCESS	0	Asynchronous dependent process
CALL	1	Synchronous dependent process
RUN	2	Non-WFL independent process
JOBSTACK	3	WFL job

Read Time

The TYPE attribute can be read at any time. However, the value is reset to PROCESS when the process terminates, regardless of what the value was when the process was in use.

USERCODE

Type	String
Units	Not applicable
Range	<usercode assignment>
Default	Null string
Read Time	Anytime
Write Time	Anytime
Inheritance	See below
Overwrite Rules	Standard
Host Services	Supported
Attribute Number	8
Synonym	None
Restrictions	None

Range

<usercode assignment>

```
—<usercode>┬─────────────────────────────────────────────┤
           └ / —<password>─┘
```

Explanation

The USERCODE task attribute specifies the usercode under which the process is run. The usercode is a major factor in determining the privilege status of the process and what files can be accessed by the process. For information about usercodes and privilege, refer to the *A Series Task Management Programming Guide*.

A USERCODE assignment must include a password unless the usercode does not have a password defined in the USERDATAFILE. However, when a process reads the USERCODE task attribute, the password is omitted from the value returned.

When a process is initiated, the system performs validation to determine whether the USERCODE value for a process is compatible with the ACCESSCODE task attribute value and the CHARGE task attribute value. The following is an outline of this validation:

1. The system checks to see that a USER entry exists for the usercode in the USERDATA file and that the password included in the USERCODE assignment is valid. Otherwise, the system discontinues the process.

2. The system performs the accesscode validation that is explained in the ACCESSCODE task attribute description in this section.

3. The system performs the charge code validation that is explained in the CHARGE task attribute description in this section.

4. For a WFL job, the WFL compiler performs both the accesscode and charge code validation at compile time. The WFL compiler issues a syntax error if either of these validations fail. (A WFL job can receive an ACCESSCODE or CHARGE value at compile time either through inheritance or through an assignment in the job attribute list.)

If the USERCODE value of an in-use process is changed, the system performs only the first of the above types of validation. The process retains its current CHARGE value, even if it is one that would not normally be permitted for the new usercode. If the ACCESSCODE value is not allowed for the new usercode, the system changes the ACCESSCODE value to a null string. In either case, the process continues running normally.

If USERCODE is assigned a null string before initiation, the null value is overridden by inheritance from the parent at initiation time. If the USERCODE of an in-use process is assigned a null string, the process becomes a nonusercoded process and the value of the ACCESSCODE task attribute is also cleared. A nonusercoded process receives a special security status, as described in the *A Series Task Management Programming Guide*.

Only a privileged process, an MCS, or a compiler can assign a null string to the USERCODE of an in-use process. If a nonprivileged process attempts to assign a null string to the usercode of an in-use process, the nonprivileged process is discontinued with a security violation.

For processes initiated from a session, changing the usercode has the side effect of preventing process messages from being displayed at the originating terminal. The messages resume if the original usercode is restored.

Inheritance

A process inherits the usercode of its parent. Processes initiated from CANDE or MARC sessions inherit the usercode of the session.

Examples

This ALGOL statement assigns a usercode:

```
REPLACE TVAR.USERCODE BY "SMITH/DAVID.";
```

This ALGOL statement assigns a null usercode:

```
REPLACE TSK.USERCODE BY ".";
```

Run-Time Errors

When an error occurs in assigning the USERCODE task attribute, field [27:20] of the ERROR task attribute of the receiving process stores the USERDATA error code. In addition, any of the following messages can be displayed.

SECURITY VIOLATION

An attempt was made to assign an illegal usercode value. The assigning
process is discontinued with HISTORYCAUSE = 2 (PROGRAMCAUSEV) and
HISTORYREASON = 29 (SECURITYERRORV). The system adds one of the following
explanatory messages:

- If an attempt was made to assign a USERCODE value that failed security validation,
  the following additional message appears.

      INVALID TASK ATTRIBUTE: USERCODE

- If a nonprivileged process attempted to set the USERCODE of an in-use process to a
  null value, the following additional message appears:

      INVALID TASK ATTRIBUTE: USERCODE IS A DOT

- If an attempt was made to assign USERCODE to an object code file at compile time,
  the following additional message appears:

      INVALID USERCODE WHEN INITIATING A TASK

USERCODE ATTRIBUTE INCORRECT SYNTAX

An attempt was made to assign USERCODE a value that did not follow the usercode
assignment syntax. The assigning process, if nonprivileged, is discontinued with
HISTORYCAUSE = 2 (PROGRAMCAUSEV) and HISTORYREASON = 131
(INCORRECTSYNTAXV).

VALIDITYBITS

Note: The VALIDITYBITS task attribute is intended for use by the
system software only. The meanings of the various fields in the
VALIDITYBITS value are subject to change without notice. It is
therefore not possible for application programs to receive reliable
information from VALIDITYBITS. For this reason, application
programs should not make use of this attribute.

VISIBILITY

Note: *The VISIBILITY task attribute has no meaning on systems running the Mark 3.9 system software release or a later release. The system displays a deimplementation warning message when a process attempts to use this attribute.*

WAITLIMIT

Type	Real
Units	Seconds
Range	0 to about 4.31E68
Default	0 (unlimited)
Read Time	Anytime
Write Time	Anytime
Inheritance	See below
Overwrite Rules	Standard
Host Services	Supported
Attribute Number	56
Synonym	None
Restrictions	None

Explanation

The WAITLIMIT task attribute specifies the number of seconds that the process is allowed to spend in a user-requested wait state. If the process waits longer than its WAITLIMIT, it is discontinued.

The WAITLIMIT task attribute applies only to program statements that explicitly wait on one or more events. In particular, it does not apply to suspended processes that issue RSVP messages, such as processes suspended with a NO FILE condition. This attribute is intended to catch otherwise undetected application program errors.

WAITLIMIT is not cumulative; it applies to each WAIT statement separately.

The default value of 0 does not impose any wait limit. Assigning 0 to this attribute removes any previously assigned wait limit.

Some forms of the WAIT statement are not affected by WAITLIMIT. These forms can cause the process to wait any amount of time without being discontinued. The following are WAIT statement forms unaffected by WAITLIMIT:

WFL	ALGOL	COBOL74
WAIT	WAIT	WAIT UNTIL INTERRUPT
WAIT (OK)		
WAIT (<real expression>)		

In the last of the three WFL statements shown, the real expression follows the syntax described in the *A Series Work Flow Language (WFL) Programming Reference Manual*.

Inheritance

A task inherits the WAITLIMIT value of its job.

If a default value is assigned for the WAITLIMIT attribute of a job queue, that value is inherited by WFL jobs run from that job queue. However, a WFL job can change its WAITLIMIT value after initiation or assign a different WAITLIMIT value to a task.

If a limit value is set for the WAITLIMIT attribute of a job queue, then WFL jobs that specify a higher WAITLIMIT value in the job attribute list cannot be accepted into that job queue. However, after initiation a WFL job can assign a WAITLIMIT value higher than the job queue WAITLIMIT.

Run-Time Error

WAIT TIME LIMIT EXCEEDED

The process remained in a user-requested wait state for longer than the time specified by WAITLIMIT. The process is discontinued with HISTORYCAUSE = 3 (RESOURCECAUSEV) and HISTORYREASON = 9 (WAITEXCEEDEDV).

Appendix A
Understanding Railroad Diagrams

What Are Railroad Diagrams?

Railroad diagrams are diagrams that show you the rules for putting words and symbols together into commands and statements that the computer can understand. These diagrams consist of a series of paths that show the allowable structure, constants, and variables for a command or a statement. Paths show the order in which the command or statement is constructed. Paths are represented by horizontal and vertical lines. Many railroad diagrams have a number of different paths you can take to get to the end of the diagram. For example:

```
— REMOVE ┬──────────────────────────────────────┤
         ├─ SOURCE ─┤
         └─ OBJECT ─┘
```

If you follow this railroad diagram from left to right, you will discover three acceptable commands. These commands are

- REMOVE

- REMOVE SOURCE

- REMOVE OBJECT

If all railroad diagrams were this simple, this explanation could end here. However, because the allowed ways of communicating with the computer can be complex, railroad diagrams sometimes must also be complex.

Regardless of the level of complexity, all railroad diagrams are visual representations of commands and statements. Railroad diagrams are intended to

- Show the mandatory items.

- Show the user-selected items.

- Present the order in which the items must appear.

- Show the number of times an item can be repeated.

- Show the necessary punctuation.

To familiarize you with railroad diagrams, this explanation describes the elements of the diagrams and provides examples.

Some of the actual railroad diagrams you will encounter might be more complex. However, all railroad diagrams, simple or complex, follow the same basic rules. They

all consist of paths that represent the allowable structure, constants, and variables for commands and statements.

By following railroad diagrams, you can easily understand the correct syntax for commands and statements. Once you become proficient in the use of railroad notation, the diagrams serve as quick references to the commands and statements.

Constants and Variables

A constant is an item that cannot be altered. You must enter the constant as it appears in the diagram, either in full or as an allowable abbreviation. If a constant is partially underlined, you can abbreviate the constant by entering only the underlined letters. In addition to the underlined letters, any of the remaining letters can be entered. If no part of the constant is underlined, the constant cannot be abbreviated. Constants can be recognized by the fact that they are never enclosed in angle brackets (< >) and are in uppercase letters.

A variable is an item that represents data. You can replace the variable with data that meets the requirements of the particular command or statement. When replacing a variable with data, you must follow the rules defined for the particular command or statement. Variables appear in railroad diagrams enclosed in angle brackets.

In the following example, BEGIN and END are constants while < statement list> is a variable. The constant BEGIN can be abbreviated since it is partially underlined. Valid abbreviations for BEGIN are BE, BEG, and BEGI.

```
— BEGIN —<statement list>— END ————————————————————|
```

Constraints

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

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

A description of each item follows.

Vertical Bar

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

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

Percent Sign

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

```
— STOP ————————————————————————————————%
```

Right Arrow

The right arrow symbol (>) is used when the railroad diagram is too long to fit on one line and must continue on the next. A right arrow appears at the end of the first line, and another right arrow appears at the beginning of the next line.

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

Required Items

A required item can be either a constant, a variable, or punctuation. A required item appears as a single entry, by itself or with other items, on a horizontal line. Required items can also exist on horizontal lines within alternate paths or nested (lower-level) diagrams. If the path you are following contains a required item, you must enter the item in the command or statement; the required item cannot be omitted.

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

```
— EVENT —<identifier>————————————————————|
```

User-Selected Items

User-selected items appear one below the other in a vertical list. You can choose any one of the items from the list. If the list also contains an empty path (solid line), none of the choices are required. A user-selected item can be either a constant, a variable, or punctuation. In the following railroad diagram, either the plus sign (+) or 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 group of items that you can repeat. A loop can span all or part of a railroad diagram. It always consists of at least two horizontal lines, one below the other, connected on both sides by vertical lines. The top line is a right-to-left path that contains information about repeating the loop.

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

```
  ┌←──────  ;  ──────┐
──┴─<field value>─────┴──────────────────────────────────────┤
```
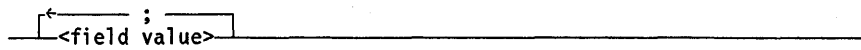
Bridge .

Sometimes a loop also includes a bridge, which is used to show the maximum number of times the loop can be repeated. The bridge can precede the contents of the loop, or it can precede the return character (if any) on the upper line of the loop.

The bridge determines the number of times you can cross that point in the diagram. The bridge is an integer enclosed in sloping lines (/ \). Not all loops have bridges. Those that do not can be repeated any number of times until all valid entries have been used.

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

```
  ┌←────── , ──────┐
──┴─/2\─┬─ LINKAGE ─┬─┴──────────────────────────────────────┤
        └─ RUNTIME ─┘
```

```
  ┌←─/2\──────────┐
──┴──┬─ LINKAGE ─┬─┴──────────────────────────────────────────┤
     └─ RUNTIME ─┘
```
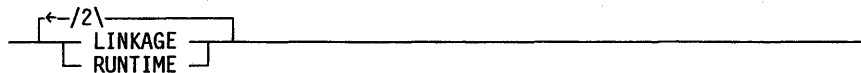
In some bridges an asterisk (*) follows the number. The asterisk means that you must cross that point in the diagram at least once. The maximum number of times that you can cross that point is indicated by the number in the bridge.

```
  ┌←────── , ──────┐
──┴─┬─/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.

The following figure shows the types of constraints used in railroad diagrams.
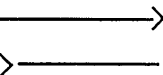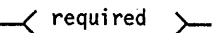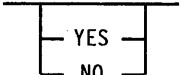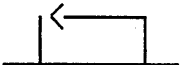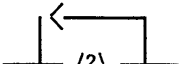
SYMBOL/PATH	EXPLANATION
——————⊣	Vertical bar. Indicates that the command or statement can be followed by another command or statement.
——————— °/o	Percent sign. Indicates that the command or statement must be on a line by itself.
—————⟶ ⟩—————	Right arrow. Indicates that the diagram occupies more than one line.
—⟨ required ⟩—	Required items. Indicates the constants, variables, and punctuation that must be entered in a command or statement.
⌐ YES ⌐ └ NO ┘	User-selected items. Indicates the items that appear one below the other in a vertical list. You select which item or items to include.
⌐⟵⌐	A loop. Indicates an item or group of items that can be repeated.
⌐⟵⌐ — /2\ —	A bridge. Indicates the maximum number of times a loop can be repeated.

Figure A–1. Railroad Constraints

Following the Paths of a Railroad Diagram

The paths of a railroad diagram lead you through the command or statement from beginning to end. Some railroad diagrams have only one path, while others have several alternate paths. The following railroad diagram indicates there is only one path that requires the constant LINKAGE and the variable < linkage mnemonic > :

— LINKAGE —<linkage mnemonic>——————————————————————⊣

Alternate paths provide choices in the construction of commands and statements. Alternate paths are provided by loops, user-selected items, or a combination of both. More complex railroad diagrams can consist of many alternate paths, or nested (lower-level) diagrams, that show a further level of detail.

For example, the following railroad diagram consists of a top path and two alternate paths. The top path includes an ampersand (&) and the constants (that are

user-selected items) in the vertical list. These constants are within a loop that can be repeated any number of times until all options have been selected. The first alternate path requires the ampersand and the required constant ADDRESS. The second alternate path requires the ampersand followed by the required constant ALTER and the required variable <new value>.

```
        ┌←─────── , ──────┐
 ─ & ──┬─┤┌─ TYPE ───────┤├────────────────────────────┤
       │ ├─ ASCII ──────┤
       │ ├─ BCL ────────┤
       │ ├─ DECIMAL ────┤
       │ ├─ EBCDIC ─────┤
       │ ├─ HEX ────────┤
       │ └─ OCTAL ──────┘
       ├─ ADDRESS ─────────────┤
       └─ ALTER ──<new value>──┘
```
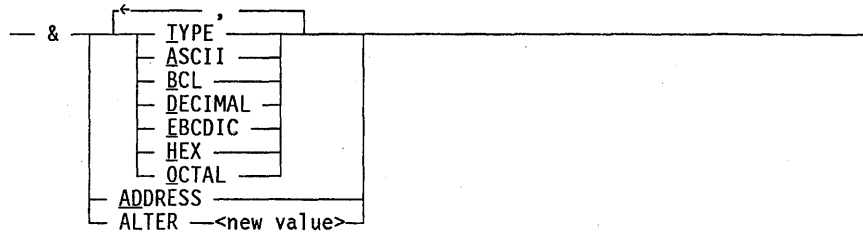
Railroad Diagram Examples with Sample Input

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 underlined, the entire word must be entered.
	The parentheses are required punctuation, and FILE4 is a sample file identifier.

Example 2

<open statement>

```
─ OPEN ──┬──────────┬──<database name>──────────────┤
         ├─ INQUIRY ─┤
         └─ UPDATE ──┘
```
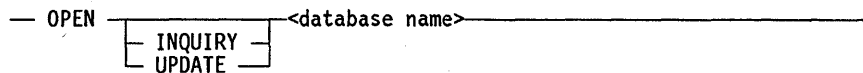
Sample Input	Explanation
OPEN DATABASE1	The constant OPEN is followed by the variable DATABASE1, which is a database name.
	The railroad diagram shows two user-selected items, INQUIRY and UPDATE. However, because there is an empty path (solid line), these entries are not required.

continued

continued

Sample Input	Explanation
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 ───┤
                                     ├─ + ────┤
                                     └─ - ────┘
```
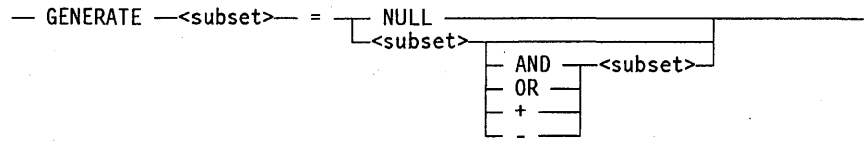
Sample Input	Explanation
GENERATE Z = NULL	The GENERATE constant is followed by the variable Z, an equal sign (=), and the user-selected constant NULL.
GENERATE Z = X	The GENERATE constant is followed by the variable Z, an equal sign, and the user-selected variable X.
GENERATE Z = X AND B	The GENERATE 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>─ ) ─┴──────────┤
```
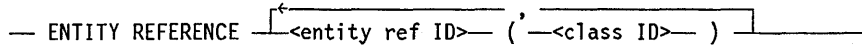
Sample Input	Explanation
ENTITY REFERENCE ADVISOR1 (INSTRUCTOR)	The required item ENTITY REFERENCE is followed by the variable ADVISOR1 and the variable INSTRUCTOR. The parentheses are required.
ENTITY REFERENCE ADVISOR1 (INSTRUCTOR), ADVISOR2 (ASST_INSTRUCTOR)	Because the diagram contains a loop, the 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>─┘
          └ - ─┘
```
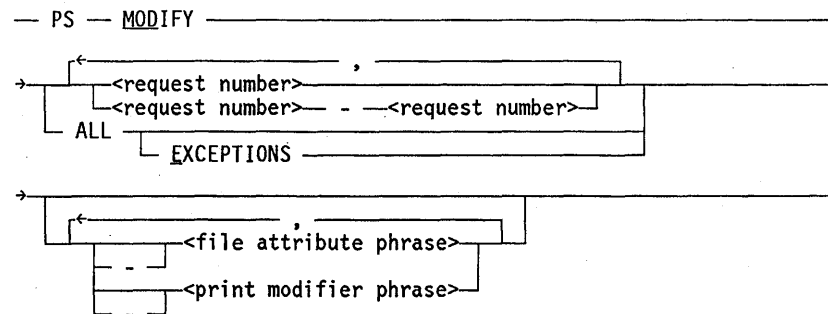
Sample Input	Explanation
PS MODIFY 11159	The constants PS and MODIFY are followed by the variable 11159, which is a request number.
PS MODIFY 11159,11160,11163	Because the diagram contains a loop, the variable 11159 can be followed by a comma, the variable 11160, another comma, and the final variable 11163.
PS MOD 11159–11161 DESTINATION = "LP7"	The constants PS and MODIFY are followed by the user-selected variables 11159–11161, which are request numbers, and the user-selected variable DESTINATION = "LP7", which is a file attribute phrase. Note that the constant MODIFY has been abbreviated to its minimum allowable form.
PS MOD ALL EXCEPTIONS	The constants PS and MODIFY are followed by the user-selected constants ALL and EXCEPTIONS.

Glossary

A

abnormal termination
> The type of termination that results when a process encounters a run-time error, or is discontinued by an operator command or a statement in another process.

access
> To perform an action on an object. Possible actions depend on the type of object; for example, interrogating or assigning a value to a variable, reading from or writing to a file, or invoking a procedure.

access mask register (AMR)
> In intersystem control (ISC), a 48-bit register attached to a host control (HC) unit and visible to all other HC units that specify the intended data flow of the HC. The ISC hardware uses the AMR mode information to determine what operations are allowable on an HC.

active
> Pertaining to the state of a process that is executing normally, and is neither scheduled nor suspended.

ALGOL
> Algorithmic language. A structured, high-level programming language that provides the basis for the stack architecture of the Unisys A Series systems. ALGOL was the first block-structured language developed in the 1960s and served as a basis for such languages as Pascal and Ada. It is still used extensively on A Series systems, primarily for systems programming.

AMR
> *See* access mask register.

ancestor
> The parent of a particular task, or the parent of any ancestor of the task.

APL
> A Programming Language. A procedure-oriented language that can produce very short but powerful programs.

APLB
> A Programming Language B. A second-generation extended version of A Programming Language (APL).

asynchronous process
> A process that executes in parallel with its initiator.

B

BDMSALGOL

A Unisys language based on Extended ALGOL that contains extensions for accessing Data Management System II (DMSII) databases.

beginning of job (BOJ)

The start of processing of a job.

beginning of task (BOT)

The start of processing of a task.

block

A program, or a part of a program, that is treated by the processor as a discrete unit. Examples are a procedure in ALGOL, a procedure or function in Pascal, a subroutine or function in FORTRAN, or a complete COBOL program.

BNA

The network architecture used on A Series, B 1000, and V Series systems as well as CP9500 and CP 2000 communications processors to connect multiple, independent, compatible computer systems into a network for distributed processing and resource sharing.

BOJ

See beginning of job.

BOT

See beginning of task.

C

calling process

A process that is linked to a library process and can import objects from that library process. *See also* user process.

CANDE

See Command and Edit.

client process

A process that is linked to a library process and can import objects from that library process. *See also* user process.

COBOL

Common Business-Oriented Language. A widely used, procedure-oriented language intended for use in solving problems in business data processing. The main characteristics of COBOL are the easy readability of programs and a considerable degree of machine independence. COBOL is the most widely used procedure-oriented language.

COBOL(68)

A version of the COBOL language that is compatible with the American National Standard X3.23-1968.

COBOL74
> A version of the COBOL language that is compatible with the American National Standard X3.23-1974.

code segment dictionary
> A memory structure that is associated with a process and that indexes the memory addresses of the various segments of program code used by that process. The same code segment dictionary can be shared by more than one process, provided that each process is an instance of the same procedure. A code segment dictionary is also referred to as a D1 stack.

Command and Edit (CANDE)
> A time-sharing message control system (MCS) that enables a user to create and edit files, and to develop, test, and execute programs, interactively.

Communications Management System (COMS)
> A general message control system (MCS) that controls online environments on A Series systems. COMS can support the processing of multiprogram transactions, single-station remote files, and multistation remote files.

compiler
> A computer program that translates instructions written in a source language, such as COBOL or ALGOL, into machine-executable object code.

COMS
> *See* Communications Management System.

constant
> An object whose value is assigned during program compilation and cannot be changed during program execution.

coroutine
> One of a group of processes that exist simultaneously, but take turns executing, so that only one of the processes is executing at any given time. The coroutine that is currently executing is called the active coroutine, and the others are called continuable coroutines.

cousin
> A process that has an ancestor in common with some other process, but does not have the same parent as the other process.

critical block
> For a dependent process, the block of the highest lexical level that includes the declaration of any critical objects used by the dependent process. The process that is executing the critical block is called the parent of the dependent process. If the parent exits the critical block while the dependent process is in use, the parent is discontinued and the dependent process is also discontinued.

critical object
> A type of object that is used by a process, but was originally declared by another process. Critical objects include the task variable for the process, the procedure declaration for the process, and any objects passed as actual parameters to the process by name or by reference.

D

data comm

See data communications.

data communications (data comm)

The transfer of data between a data source and a data sink (two computers, or a computer and a terminal) by way of one or more data links, according to appropriate protocols.

Data Communications ALGOL (DCALGOL)

A Unisys language based on ALGOL that contains extensions for writing message control system (MCS) programs and other specialized system programs.

Data Management ALGOL (DMALGOL)

A Unisys language based on ALGOL that contains extensions for writing Data Management System II (DMSII) software and other specialized system programs.

Data Management System II (DMSII)

A specialized system software package used to describe a database and maintain the relationships among the data elements in the database.

data specification

A section of a Work Flow Language (WFL) source program containing data that can be read by tasks of the WFL job. A data specification is also referred to as a data deck.

database (DB)

An integrated, centralized system of data files and program utilities designed to support an application. The data sets and associated index structures are defined by a single description. Ideally, all the permanent data pertinent to a particular application resides in a single database. The database is considered a global entity that several applications can access and update concurrently.

DB

See database.

DCALGOL

See Data Communications ALGOL.

declaration

A programming language construct used to identify an object, such as a type or variable, to the compiler. A declaration can be used to associate a data type with the object so that the object can be used in a program.

declared external procedure

A dummy procedure declaration used in ALGOL or COBOL74 to enable a program to initiate a separate program.

dependent process

A process that depends on the continued existence of another process called the parent process. *See also* task.

descendant

An offspring of a particular process, or an offspring of a descendant of that process.

direct addressing environment

The set of objects that can be accessed by statements in a particular procedure, but that are not passed as parameters to that procedure.

directory

(1) A table of contents listing the files contained on a device. The device is usually a disk or a tape. (2) A list of file names organized into a hierarchy according to similarities in their names. File names are grouped in a directory if their first name constants (and associated usercodes) are identical. These groups are divided into subdirectories consisting of those file names whose first two name constants are identical, and so on.

discontinue

To cause a process to terminate abnormally. A process can be discontinued by operator commands, by statements in related processes, or by the system software.

distributed systems service (DSS)

One of a collection of services provided on Unisys hosts to support communications across multihost networks. DSSs can be services such as file handling, station transfer, and mail transfer.

DMALGOL

See Data Management ALGOL.

DMSII

See Data Management System II.

DSS

See distributed systems service.

E

EBCDIC

Extended Binary Coded Decimal Interchange Code. An 8-bit code representing 256 graphic and control characters that are the native character set of most mainframe systems.

EFS

See extended file system.

element

A component of an array.

end of file (EOF)

A code at the end of a data file that signals that the last record in the file has been processed.

EOF

See end of file.

exception task

A process that has a special relationship with another process, such that the following are true: the exception task's EXCEPTIONEVENT task attribute is caused whenever the status of the related process changes; and the related process can use the EXCEPTIONTASK task attribute to access the task attributes of the exception task.

execution

The act of processing statements in a program.

exit

To end the processing of an entered block. Exiting the block eliminates the activation record.

expression

A combination of operands and operators that results in the generation of one or more values.

extended addressing environment

The set of objects that can be accessed by statements in a particular procedure, including any objects that were passed as parameters to that procedure.

extended file system (EFS)

A file system controlled by KEYEDIOII. This file system supports sequential, direct, and indexed organization; alternate index files; record-level locking; and delete-capability for RPG files. *Contrast with* basic file system.

external procedure

A procedure whose procedure body is contained in an object code file different from the statement that invokes the procedure. External procedures are of three kinds: declared external procedures, passed external procedures, and library procedures.

external process

A process created by initiating an external procedure.

F

family

(1) One or more disks logically grouped and treated as a single entity by the system. Each family has a name, and all disks in the family must have been entered into the family with the RC (Reconfigure Disk) system command. (2) *See also* process family.

fatal

Referring to something capable of causing a process to be discontinued. An error that causes a process to be discontinued is called a fatal error.

FETCH specification

A statement in a Work Flow Language (WFL) job that provides a message an operator can display with a PF (Print Fetch) system command. Resetting the NOFETCH system option delays initiation of jobs with FETCH specifications until the operator enters an OK command for each job.

FIB

See file information block.

file attribute

An element that describes a characteristic of a file and provides information the system needs to handle the file. Examples of file attributes are the file title, record size, number of areas, and date of creation. For disk files, permanent file attribute values are stored in the disk file header.

file information block (FIB)

A data structure in an object code file that contains information describing a file.

formal parameter

An object that is declared in a procedure heading and that receives its value from an actual parameter when the procedure is invoked.

FORTRAN

Formula Translation. A high-level, structured programming language intended primarily for scientific use.

FORTRAN77

A version of the FORTRAN language that is compatible with the ANSI X3.9-1978 standard.

frozen

The state of a process whose STATUS task attribute has the value FROZEN. This STATUS value indicates that the process is a library process and provides objects that can be imported by user processes.

G

guard file

A disk file created by the GUARDFILE utility program that describes the access rights of various users and programs to a program, data file, or database.

H

halt/load

A system-initialization procedure that temporarily halts the system and loads the master control program (MCP) from a disk to main memory.

HC unit

See host control (HC) unit.

host control (HC) unit

A specialized data link processor (DLP) that enables host systems to communicate through an intersystem control (ISC) hub on a channel-to-channel communications interface between A Series and CP9500 systems.

I

I/O

Input/output. An operation in which the system reads data from or writes data to a file on a peripheral device such as a disk drive.

import object

The declaration of a library object in a user program.

in-use process

A process that has been submitted for initiation and has not yet terminated. The state of an in-use process can be scheduled, active, or suspended.

independent process

A process that does not depend on the continued existence of a parent process. An independent process is the head of any process family it is part of. *See also* job.

independent runner (IR)

A master control program (MCP) procedure that is initiated as an independent process. The procedure is executed in its own process stack rather than in the stack of a user process. An IR can be either visible or invisible. If the IR is visible, its status can be interrogated. If the IR is invisible, it does not appear in mix displays.

index

A value used to specify a particular element of an array variable.

InfoGuard

The Unisys security-enhancement software for A Series systems. InfoGuard provides such features as password management, selective logging and auditing, tape volume security, and simplified system-security configuration.

inheritance

The automatic transfer of particular task attribute values from a process to a descendant process. More broadly, inheritance also refers to the automatic transfer of values from job queue attributes or session attributes to the equivalent task attributes of a descendant process.

initiation

A type of procedure invocation that causes the creation of a new process, with its own process stack and process information block (PIB). Additionally, a new code segment dictionary is created if a code segment dictionary for that procedure is not already available.

initiator

The process that initiates a particular process. The initiator can be a different process from the parent process.

instance

A process that is an execution of a particular procedure and that has its own process stack. Multiple instances of a procedure can exist at the same time; a new instance is created each time the procedure is initiated.

interactive process
> A process that reads input from a terminal or operator display terminal (ODT), and whose actions are largely determined by the input received. A data entry process, such as the Editor, is an example of an interactive process.

internal procedure
> A procedure whose procedure body is contained in the same object code file as the statement that invokes the procedure.

internal process
> The execution of an internal procedure that has been initiated.

invocation
> The act that transfers control to the start of a specified procedure, initializes any parameters, and begins the execution of the statements of the procedure. Invocations are of two kinds: entrances and initiations.

J

job
> An independent process. The job of a particular task is the independent process that is the eldest ancestor of that task.

job log
> A log that is stored in a job file and contains log entries for a particular job and its descendant tasks. When the job terminates, the job log is processed to produce the job summary.

job queue
> A structure in the system software that stores a list of jobs that have been compiled and are waiting to be initiated.

job summary
> A file, produced after a job completes execution, that lists information such as the tasks initiated by the job, the beginning and ending times for each task, and the termination information for each task.

L

library
> A program that exports objects for use by user programs.

library process
> An instance of the execution of a library. The sharing option of a library determines whether multiple user programs use the same instance of the library.

local object
> An object that is declared within a particular block.

logical file

A file variable declared in a program, which represents the file and its structure to the program. A logical file has no properties of its own until it is described by file attributes or associated with a physical file.

logical station number (LSN)

(1) In the Network Definition Language II (NDLII), a unique number assigned to each station in a network. Each station has an LSN assigned according to the order in which the stations are defined in NDLII. The first defined station is 1. (2) In the Interactive Datacomm Configurator (IDC), a unique number assigned to each station structure. When IDC creates the DATACOMINFO file from the network information file II (NIFII), it assigns an LSN to each structure sequentially, beginning with the number 2. The numbers allocated by IDC are the same as those used by the operating system to identify a station.

LSN

See logical station number.

M

MAKEUSER

A utility used to define, modify, or display information about the usercodes that are available on the system. The usercode information is stored in a file called the USERDATAFILE.

MARC

See Menu-Assisted Resource Control.

master control program (MCP)

The central program of the A Series operating system. The term applies to any master control program that Unisys may release for A Series systems.

MCP

See master control program.

MCS

See message control system.

Menu-Assisted Resource Control (MARC)

A menu-driven interface to A Series systems that also enables direct entry of commands.

message control system (MCS)

A program that controls the flow of messages between terminals, application programs, and the operating system. MCS functions can include message routing, access control, audit and recovery, system management, and message formatting.

microsecond

One-millionth of a second (.000001).

mix

> The set of processes that currently exist on a particular computer. The mix can include active, scheduled, and suspended processes.

mix number

> A 4-digit number that identifies a process while it is executing. This number is stored in the MIXNUMBER task attribute.

MYJOB

> A predeclared task variable that a process can use to access the task attributes of its job.

MYSELF

> A predeclared task variable that a process can use to access its own task attributes.

N

normal termination

> The termination of a process that has executed successfully, without any errors and without being terminated prematurely by an operator command or another process.

O

object

> Any item declared in a program. Arrays, files, procedures, tasks, and variables are all examples of objects.

object code file

> A file produced by a compiler when a program is compiled successfully. The file contains instructions in machine-executable object code.

ODT

> *See* operator display terminal.

offspring

> The dependent process whose critical block is owned by a particular parent process.

operator display terminal (ODT)

> A terminal or other device that is connected to the system in such a way that it can communicate directly with the operating system. The ODT allows operations personnel to accomplish system operations functions through either of two operating modes: system command mode or data comm mode.

outer block

> The portion of a program that has the lowest lexical level.

overlay

> To load code or data into a memory area that was previously allocated to other code or data, and to write any data that previously occupied the area to a disk file if necessary.

P

parameter

An identifier associated in a special way with a procedure. A parameter is declared in the procedure heading and is automatically assigned a value when the procedure is invoked.

parameter passing

The act of passing an object or a value from an actual parameter to a formal parameter.

parent

A process that owns the critical block of a dependent process. If the parent exits the critical block before the dependent process terminates, the dependent process is discontinued.

partner process

The process that is specified by the PARTNER task attribute of another process. A process can transfer control to its partner process by executing a general continue statement.

performance

(1) A measurement of how efficiently a process uses resources such as processor time, I/O time, or elapsed time. (2) A measure of the amount of work a computer system is able to do in a given period of time.

physical file

A file as it is stored on a particular recording medium such as a disk or a tape.

PIB

See process information block.

PL/I

Programming Language I. A high-level, structured programming language designed primarily for scientific and commercial use.

private process

A process whose task attributes cannot be accessed by other processes. Assigning the *private process* option to the OPTION task attribute causes a process to become a private process.

privilege

The ability to invoke actions that are not ordinarily allowed, such as accessing private files stored under other usercodes or invoking privileged functions such as SETSTATUS. The concept of privilege applies to usercodes, programs, and processes.

procedure

A block that can be invoked by statements elsewhere in the same program or, in some cases, by statements in another program. In most instances, a procedure has a procedure heading and a procedure body. Examples are a procedure in ALGOL, a procedure or function in Pascal, a subroutine or function in FORTRAN, or a complete COBOL program.

procedure initiation
> *See* initiation.

process
> The execution of a program or of a procedure that was initiated. The process has its own process stack and process information block (PIB). It also has a code segment dictionary, which can be shared with other processes that are executions of the same program or procedure.

process family
> A group of processes consisting of a single job and any tasks that are descendants of that job.

process information block (PIB)
> A memory structure that is associated with each process stack, and which stores the values of the task attributes of that process.

process stack
> A memory structure that stores information about the current state of the execution of a procedure. The process stack includes activation records for all blocks that the process has entered and not yet exited.

process state
> The current status of a process. The three process states are scheduled, active, or suspended.

processor
> A hardware component that executes programs and procedures.

program
> (1) A specification of the sequence of computational steps that solve a computational problem. The steps are written (coded) in a particular programming language. (2) An object code file.

pseudostation
> A station created by the operating system that can be attached to, and controlled by, a message control system (MCS) like a *real* station. Unlike a real station, however, a pseudostation is not declared in the SOURCENDLII file or the DATACOMINFO file, has no line assigned, and does not need a corresponding physical terminal on the local host.

R

remote file
> A file with the KIND attribute specified as REMOTE. A remote file enables object programs to communicate interactively with a terminal.

remote host system
> Any computer system that is linked to the local host by means of a telecommunications network.

remote job entry (RJE)
A Unisys message control system (MCS) that allows jobs, data, and control commands to be sent to a central system from a remote card reader; RJE also allows output of data from the central system to be sent to remote peripherals.

remote process
A process initiated by a process that was running on another host system.

resuming
The act of changing a library process into a nonlibrary process. For example, a temporary library process resumes execution as a nonlibrary process when the last user process delinks from the library. *Contrast with* thawing.

RJE
See remote job entry.

RPG
Report Program Generator. A high-level, commercially oriented programming language used most frequently to produce reports based on information derived from data files.

RSVP message
A message the system displays for a suspended process that states the reason the process was suspended. RSVP messages ask for a reply such as *OK* or *DS*.

run-time error
An error occurring during the execution of a program, which causes the system software to terminate execution of that program abnormally.

S

scheduled process
A process whose initiation is delayed, either because the operator has entered an HS (Hold Schedule) system command or because the operating system estimates the process is likely to need more memory than is currently available.

session
The interactions between a user and a message control system (MCS) during a particular period of time that is assigned an identifying session number. Logging on initiates a new session; logging off terminates a session. Each Menu-Assisted Resource Control (MARC) or Command and Edit (CANDE) dialogue at a terminal accesses a different session.

sibling
A task that has the same parent as another task.

SORT facility
An operating system procedure that sorts a file or a set of records. SORT can be activated through ALGOL, COBOL(68), COBOL74, PL/I, or the SORT compiler.

source file
A file in which a source program is stored.

source program

A program coded in a language that must be translated into machine language before execution. The translator program is usually a compiler.

stack

(1) A region of memory used to store data items in a particular order on a last-in, first-out basis. (2) A nonpreferred synonym for process stack.

station

The outer end of a communication line. A station can correspond to a single terminal connected on a single line, or several stations can be connected on a line.

support library

A library that is associated with a function name. User programs can access a support library by way of its function name instead of its object code file title. The operator uses the SL (Support Library) system command to link function names with libraries.

suspended process

A process that has temporarily stopped executing and cannot continue until appropriate operator or programmatic action is taken. A process can be suspended deliberately by an operator command or a statement in a program. In addition, the operating system can suspend a process automatically, for example, if the process has requested a file that is missing.

synchronous process

A process whose initiator waits after initiating the process. When the process terminates, the initiator resumes execution.

system command

Any of a set of commands used to communicate with the operating system. System commands can be entered at an operator display terminal (ODT), in a Menu-Assisted Resource Control (MARC) session, or by way of the DCKEYIN function in a privileged Data Communications ALGOL (DCALGOL) program.

system software

The master control program (MCP) and all other object code files necessary for system operation.

T

TADS

See Test and Debug System.

task

A dependent process.

task attribute

Any of a number of items that describe and control various aspects of process execution such as the usercode, priority, and the default family specification. Task attributes can be assigned interactively through task equations, or programmatically through statements that use task variables.

task file

A printer backup file that is associated with each process, and that stores any program dumps generated by the process while the TOPRINTER program dump option is enabled. Processes can also write comments to the task file by way of the TASKFILE task attribute.

task variable

An object that is used to interrogate or modify the task attributes of a particular process.

tasking

The act of initiating, monitoring, or controlling processes. The processes can be either jobs or tasks. Operators and users can enter tasking commands from an operator display terminal (ODT), a Command and Edit (CANDE) session, or a Menu-Assisted Resource Control (MARC) session. Programs can initiate processes with such statements as CALL, PROCESS, or RUN. Programs can monitor and control processes by reading and assigning the values of various task attributes.

termination

The act of permanently ceasing execution of a process. The process stack and process information block (PIB) are removed. The code segment dictionary can also be removed.

Test and Debug System (TADS)

A Unisys interactive tool for testing and debugging programs and libraries. TADS enables the programmer to monitor and control the execution of the software under testing and examine the data at any given point during program execution.

thawing

The act of changing a permanent or control library into a temporary library. *Contrast with* resuming.

U

user process

(1) A process that is not an invisible independent runner, a message control system (MCS), or a system library. (2) A process that is linked to a library process and can import objects from that library process. *Synonym for* calling process, client process.

user program

(1) A program that is not part of the system software. (2) A program that uses objects imported from a library program.

usercode

An identification code used to establish user identity and control security, and to provide for segregation of files. Usercodes can be applied to every task, job, session, and file on the system. A valid usercode is identified by an entry in the USERDATAFILE.

usercode attribute

A characteristic that can be associated with a usercode in the USERDATAFILE. A set of standard usercode attributes, such as PU, MAXPW, IDENTITY, and PASSWORD, are supplied as part of the description of the USERDATAFILE structure. The system

usercode

An identification code used to establish user identity and control security, and to provide for segregation of files. Usercodes can be applied to every task, job, session, and file on the system. A valid usercode is identified by an entry in the USERDATAFILE.

usercode attribute

A characteristic that can be associated with a usercode in the USERDATAFILE. A set of standard usercode attributes, such as PU, MAXPW, IDENTITY, and PASSWORD, are supplied as part of the description of the USERDATAFILE structure. The system administrator or security administrator can define additional usercode attributes to meet the specific needs of an installation.

USERDATAFILE

A system database that defines valid usercodes and contains various data about each user (such as accesscodes, passwords, and chargecodes) and the population of users for a particular installation.

V

variable

An object in a program whose value can be changed during program execution.

W

WFL

See Work Flow Language.

WFL job

A Work Flow Language (WFL) program, or the execution of such a program.

window

In the Communications Management System (COMS) architecture, the concept that enables a number of program environments to be operated independently and simultaneously at one station. One of the program environments can be viewed while the others continue to operate.

Work Flow Language (WFL)

A Unisys language used for constructing jobs that compile or run programs on A Series systems. WFL includes variables, expressions, and flow-of-control statements that offer the programmer a wide range of capabilities with regard to task control.

WORM

Write-once, read-many. *See also* WORM device, WORM medium.

WORM device

A random-access optical disk drive that can write to the WORM media only once, but can access the files on the WORM media many times.

WORM medium
    A high-density storage medium that can be written to only once and can be read from many times.

Bibliography

A Series ALGOL Programming Reference Manual, Volume 1: Basic Implementation
(8600 0098) Unisys Corporation.

A Series ALGOL Test and Debug System (TADS) Programming Guide (1169539).
Unisys Corporation.

A Series C Test and Debug System (TADS) Programming Reference Manual
(8600 1591). Unisys Corporation.

A Series CANDE Operations Reference Manual (8600 1500). Unisys Corporation.

A Series COBOL ANSI-68 Programming Reference Manual (8600 0320). Unisys
Corporation.

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

A Series COBOL ANSI-74 Test and Debug System (TADS) Programming Guide
(1169901). Unisys Corporation.

*A Series COBOL ANSI-85 Test and Debug System (TADS) Programming Reference
Manual* (8600 0957). Unisys Corporation.

A Series DCALGOL Programming Reference Manual (8600 0841). Unisys
Corporation.

A Series Disk Subsystem Administration and Operations Guide (8600 0668). Unisys
Corporation.

A Series DMSII Application Program Interfaces Programming Guide (5044225).
Unisys Corporation.

A Series File Attributes Programming Reference Manual (8600 0064). Unisys
Corporation.

A Series FORTRAN Programming Reference Manual (1222691). Unisys
Corporation.

A Series FORTRAN77 Programming Reference Manual (3957 6053). Unisys
Corporation.

A Series FORTRAN77 Test and Debug System (TADS) Programming Guide (1222667).
Unisys Corporation.

A Series I/O Subsystem Programming Guide (8600 0056). Unisys Corporation.

A Series Menu-Assisted Resource Control (MARC) Operations Guide (8600 0403).
    Unisys Corporation.

*A Series MultiLingual System (MLS) Administration, Operations, and Programming
    Guide* (8600 0288). Unisys Corporation.

A Series NEWP Programming Reference Manual (5044233). Unisys Corporation.

A Series PL/I Reference Manual (1169620). Unisys Corporation.

*A Series Print System (PrintS/ReprintS) Administration, Operations, and
    Programming Guide* (8600 1039). Unisys Corporation.

*A Series Report Program Generator (RPG) Programming Reference Manual, Volume 1:
    Basic Implementation* (8600 0544). Unisys Corporation.

A Series Security Administration Guide (8600 0973). Unisys Corporation.

A Series Security Features Operations and Programming Guide (8600 0528). Unisys
    Corporation.

A Series System Administration Guide (8600 0437). Unisys Corporation.

A Series System Commands Operations Reference Manual (8600 0395). Unisys
    Corporation.

A Series System Messages Support Reference Manual (8600 0429). Unisys
    Corporation.

A Series System Operations Guide (8600 0387). Unisys Corporation.

A Series System Software Support Reference Manual (8600 0478). Unisys
    Corporation.

A Series System Software Utilities Operations Reference Manual (8600 0460). Unisys
    Corporation.

A Series Systems Functional Overview (8600 0353). Unisys Corporation.

A Series Task Management Programming Guide (8600 0494). Unisys Corporation.

A Series Work Flow Language (WFL) Programming Reference Manual (8600 1047).
    Unisys Corporation.

Index

A

accept event, 2–14
ACCEPTEVENT ATTRIBUTE IS
       READONLY error message, 2–15
ACCEPTEVENT task attribute, 2–14
< accesscode assignment>, 2–16
ACCESSCODE ATTRIBUTE INCORRECT
       SYNTAX error message, 2–17
< accesscode password>, 2–16
  in < accesscode assignment>, 2–16
ACCESSCODE task attribute, 2–16
< accesscode>, 2–16
  in < accesscode assignment>, 2–16
accessing process
  and task attribute errors, 1–26
ACCUMIOTIME task attribute, 2–19
ACCUMPROCTIME task attribute, 2–20
ALGOL
  bit-level task attribute access, 1–10
  Boolean task attribute syntax, 1–6
  event task attribute syntax, 1–7
  integer and real task attribute syntax, 1–7
  mnemonic task attribute syntax, 1–8
  string task attribute syntax, 1–9
  task attribute access in, 1–2
  task-valued task attribute syntax, 1–9
< alternate family>, 2–69
  in < family specification>, 2–69
APLB
  task attribute access in, 1–2
APPLYLIST task attribute, 2–21
ARRAYS option, of OPTION task attribute,
    2–144
asynchronous processes
  LOCKED task attribute, 2–125
ATTABLEGEN, 2–1
ATTEMPT TO EXCEED TEMPORARY
       FILE LIMIT error message, 2–214
< attribute value>, 2–120
ATTRIBUTE/INTERPRETER/INTERFACE
  and ATTRIBUTEMESSAGE calls, 1–18
  and HANDLEATTRIBUTES calls, 1–11

ATTRIBUTEMESSAGE procedure of
       WFLSUPPORT library, 1–18
AUTORESTORE task attribute, 2–22
AUTORM option, of OPTION task attribute,
    2–144
AUTOSWITCHTOMARC task attribute,
    2–24
AX (Accept) system command
  and the ACCEPTEVENT task attribute,
    2–14

B

BACKUP option of OPTION task attribute,
    2–144
< backup prefix>, 2–28
BACKUPDESTINATION ATTRIBUTE
      INCORRECT SYNTAX error
      message, 2–51
BACKUPDESTINATION, synonym for
      DESTNAME task attribute, 2–50
BACKUPFAMILY ATTRIBUTE MAY ONLY
      BE SET BY AN MCS... message,
    2–27
BACKUPFAMILY task attribute, 2–25
BACKUPKIND file attribute
  and BACKUPFAMILY task attribute,
    2–25
BACKUPPREFIX, synonym for BDNAME
    task attribute, 2–28
BASE option, of OPTION task attribute,
    2–144
BDBASE option
  effect on BDNAME task attribute, 2–29
BDBASE option, of OPTION task attribute,
    2–145
BDNAME ATTRIBUTE INCORRECT
      SYNTAX error message, 2–29

O

object code files
    task attribute assignments
        using COMPILE and MODIFY, 1-5
        using HANDLEATTRIBUTES, 1-21
OP (Options) system command
    LPBDONLY operating system option,
        2-144
    NOFETCH operating system option, 2-72
    PDTODISK operating system option,
        2-146
    RESOURCECHECK operating system
        option, 2-162
operating system options
    LPBDONLY, 2-144
    NOFETCH, 2-72
    PDTODISK, 2-146
    RESOURCECHECK, 2-162
OPTION task attribute, 2-144
options, (See operating system options,
    OPTION task attribute)
OPTIONS, synonym for OPTION task
    attribute, 2-144
ORGHOSTNAME task attribute, 2-12
ORGUNIT task attribute, 2-149
    interrogating at the bit level, 1-9
OTHERPBITCOUNT task attribute, 2-152
OTHERPBITTIME task attribute, 2-153
overwrite rules, 1-24

P

partner processes
    PARTNER task attribute description,
        2-154
PARTNER task attribute, 2-154
PARTNEREXISTS task attribute, 2-156
<password>, 2-8
PDTODISK operating system option, 2-146
PL/I
    task attribute access in, 1-2
PRESENTARRAYS option, of OPTION task
    attribute, 2-145
<primary family>, 2-69
    in <family specification>, 2-69
<print attribute phrase>
    in <print specification>, 2-157
PRINT LIMIT EXCEEDED error message
    in task attribute discussion, 2-131

<print modifier phrase>
    in <print specification>, 2-157
<print specification>, 2-157
print system
    PRINTDEFAULTS task attribute, 2-157
PRINTDEFAULTS ATTRIBUTE
        INCORRECT SYNTAX error
        message, 2-158
PRINTDEFAULTS task attribute, 2-157
PRINTLIMIT, synonym for MAXLINES task
    attribute, 2-130
PRIORITY task attribute, 2-159
    and PRIORITY usercode attribute, 2-160
private processes
    and OPTION task attribute, 2-146
PRIVATELIBRARIES option, of OPTION
    task attribute, 2-145
PRIVILEGED REQUIRED TO SET
        FILEACCESSRULE = ACTOR
        message, 2-75
processes
    accessing, and task attribute errors, 1-26
    private processes
        and OPTION task attribute, 2-146
    receiving, and task attribute errors, 1-26
PROCESSIOTIME, synonym for
        ACCUMIOTIME task attribute, 2-19
processor time
    interrogating programmatically, 2-20
PROCESSTIME, synonym for
        ACCUMPROCTIME task attribute,
        2-20
PUNCH LIMIT EXCEEDED error message,
    2-127
PUNCHLIMIT, synonym for MAXCARDS
    task attribute, 2-126

Q

QUEUE, synonym for CLASS task attribute,
    2-37

R

railroad diagrams, explanation of, A-1
read-only task attributes, 2-10
real task attributes, syntax for using, 1-7
receiving process
    and task attribute errors, 1-26

T

86000502-000