

ClearPath Enterprise Servers

Work Flow Language (WFL) Programming Reference Manual

ClearPath MCP 19.0

NO WARRANTIES OF ANY NATURE ARE EXTENDED BY THIS DOCUMENT. Any product or related information described herein is only furnished pursuant and subject to the terms and conditions of a duly executed agreement to purchase or lease equipment or to license software. The only warranties made by Unisys, if any, with respect to the products described in this document are set forth in such agreement. Unisys cannot accept any financial or other responsibility that may be the result of your use of the information in this document or software material, including direct, special, or consequential damages.

You should be very careful to ensure that the use of this information and/or software material complies with the laws, rules, and regulations of the jurisdictions with respect to which it is used.

The information contained herein is subject to change without notice. Revisions may be issued to advise of such changes and/or additions.

Notice to U.S. Government End Users: This software and any accompanying documentation are commercial items which have been developed entirely at private expense. They are delivered and licensed as commercial computer software and commercial computer software documentation within the meaning of the applicable acquisition regulations. Use, reproduction, or disclosure by the Government is subject to the terms of Unisys' standard commercial license for the products, and where applicable, the restricted/limited rights provisions of the contract data rights clauses.

Contents

Section 1. WFL Capabilities

Documentation Updates	1-2
What's New?	1-2
Overview of WFL	1-3
Task Initiation	1-3
Other Task Initiation Statements	1-4
Task Specifications	1-5
Data Specifications	1-6
Flow of Control Statements	1-6
Processing Data	1-7
Subroutine Control	1-9
Task Control	1-11
File Handling	1-13
File Management	1-14
Communication	1-15
Job Format	1-16

Section 2. Job Initiation

Overview	2-1
Sources for Job Initiation	2-1
START and WFL Commands from CANDE Sessions	2-1
START Statements from Running WFL Jobs	2-4
Operator Display Terminals (ODTs)	2-4
Menu-Assisted Resource Control (MARC)	2-5
Distributed Systems Services	2-6
User Programs in Other Languages	2-7
Magnetic Tapes	2-8
Job Continuation After a Task Fails	2-9
Job Restart After a Halt/Load	2-9
ON RESTART Statement	2-12
Dummy Files	2-12

Section 3. Job Structure

Overview	3-1
Job Syntax	3-1

Job Structure	3-1
Job Contents	3-2
Job Format	3-2
AT Host Name	3-2
Job Title	3-3
Job Parameter List	3-4
Job Disposition	3-7
Job Attribute List	3-7
Resource-Limiting Attributes	3-8
CLASS Specification	3-10
FETCH Specification	3-10
STARTTIME Specification	3-11
Declaration List	3-14
Statement List	3-15
WFL Job Example	3-17

Section 4. Declarations

Overview	4-1
Declaration Syntax	4-1
Scope of Declarations	4-1
Variable Initialization	4-2
Constant Identifiers	4-2
Boolean Variables	4-3
Integer Variables	4-4
Real Variables	4-4
String Variables	4-4
File Variables	4-5
Task Variables	4-6
Subroutines	4-7
Subroutine Parameters	4-8
Global Data Specifications	4-9

Section 5. Task Initiation

Overview	5-1
Task Initiation Statements	5-1
Task Equation	5-3
Task Attributes	5-3
Task Attribute Assignment	5-8
Complex Task Attribute Assignments	5-11
ACCESSCODE Assignment	5-11
CORE Assignment	5-12
CURRENTDIRECTORY Assignment	5-13

DATAPATH Assignment	5-14
EXECUTEPATH Assignment	5-15
FAMILY Assignment	5-16
OPTION Assignment.	5-18
PRINTDEFAULTS Assignment	5-19
RESOURCE Assignment	5-21
SUPPRESSWARNING Assignment	5-21
USERCODE Assignment	5-22
Using Task Variables	5-23
Assigning Task Attributes	5-24
Reusing Task Variables	5-25
Interrogating Task Attributes	5-27
Interrogating Task Status.	5-28
File Attribute Inquiry	5-28
Interrogating Complex Task Attributes	5-29
MYJOB and MYSELF Predeclared Task Variables	5-29
File Equations	5-30
Causing the Task to Use a Different Input or Output File.	5-31
Changing the Attributes of Files Used by the Task	5-31
Causing the Task to Read from a Data Specification	5-32
How the Task Can Override WFL File Equations	5-32
Resolving Repeated File Equations to the Same File	5-32
Global File Assignment	5-33
Using Remote Files.	5-34
File Attribute Assignment	5-35
Device Kind Assignment	5-35
Serial Number Assignment	5-36
Using File Attributes	5-37
Assigning File Attributes	5-39
Interrogating File Attributes	5-40
Nonresident Files	5-42
Library Equation	5-42
Overriding WFL Library Equations	5-43
Resolving Repeated Library Equations to the Same Library	5-43
Database Equation	5-44
Local Data Specifications.	5-44

Section 6. Statements

Overview	6-1
WFL Statement Groupings	6-1
ABORT Statement	6-4
ACCESS Statement	6-5

ADD Statement	6-6
ALTER Statement	6-7
Archive Subsystem	6-16
ARCHIVE Backup Statement	6-18
ARCHIVE Statement Options	6-21
ARCHIVE Disk Volume	6-22
ARCHIVE Disk Volume Attribute List	6-23
ARCHIVE Tape Volume	6-23
ARCHIVE Tape Volume Attribute List	6-24
ARCHIVE CD Volume	6-32
ARCHIVE CD Volume Attribute List	6-33
ARCHIVE Task Equation List	6-36
ARCHIVE MERGE Statement	6-37
ARCHIVE PURGE Statement	6-38
ARCHIVE RELEASE Statement	6-38
ARCHIVE RESTORE Statement	6-39
ARCHIVE ROLLOUT Statement	6-41
ARCHIVE VOLUME Statement	6-43
Assignment Statements	6-45
BIND Statement	6-47
CASE Statement	6-48
CATALOG Statement	6-49
CHANGE Statement	6-50
COMPILE or BIND Statement	6-53
Naming the Object Code File	6-54
Choosing a Compiler	6-54
Binding	6-55
Object Code File Disposition	6-55
Task Variables	6-56
Compiler Task Equation List	6-57
File, Library, and Database Equations and Task Attributes	6-57
Local Data Specifications	6-58
Compound Statement	6-59
COPY or ADD Statement	6-60
Copying Files	6-63
Library Maintenance	6-63
COPY Options	6-65
COPY Request	6-72
Copying Files from Tape or CD-ROM	6-95
COPY and ADD Statement Examples	6-103
COPY File Transfer Services	6-106
CREATE LIBMAINTDIR Statement	6-116
CRUNCH Statement	6-119

DISPLAY Statement	6-119
DO Statement	6-120
GO Statement	6-120
IF Statement.	6-121
INITIALIZE Statement	6-122
INSTRUCTION Statement	6-123
LOCK Statement.	6-124
LOG Statement.	6-125
MKDIR Statement.	6-125
MODIFY Statement	6-126
MOVE Statement	6-128
Null Statement	6-131
ON Statement	6-132
OPEN Statement	6-135
PASSWORD Statement.	6-135
PB Statement	6-136
PRINT Statement	6-136
Printing Portions of a File.	6-139
PROCESS Statement	6-141
PURGE Statement	6-142
RELEASE Statement.	6-143
REMOVE Statement	6-143
REPLACE Statement.	6-146
RERUN Statement	6-147
RESTORE Statement	6-148
Library Maintenance	6-150
RESTORE Statement Options	6-150
RESTORE Tape and CD-ROM Attributes.	6-151
RETURN Statement	6-154
REWIND Statement	6-154
RUN Statement	6-155
SECURITY Statement	6-159
START Statement	6-162
START AND WAIT Statement	6-165
STOP Statement.	6-166
Subroutine Invocation Statement.	6-167
UNWRAP Statement.	6-169
USER Statement.	6-173
VOLUME Statement	6-174
Tape Volume Security	6-180
VOLUME ADD Statement with Tape Security Subsystem.	6-180
WAIT Statement	6-184
WHILE Statement.	6-187

WRAP Statement	6-188
--------------------------	-------

Section 7. Expressions

Overview	7-1
Boolean Expressions.	7-1
Boolean Primary	7-2
Integer Expressions	7-7
Integer Primary.	7-8
Real Expressions	7-10
Real Primary.	7-11
String Expressions	7-13
String Primary.	7-14
Mnemonic Primaries.	7-23
Constant Expressions	7-24
Boolean Constant Expression	7-25
Integer Constant Expression	7-25
Real Constant Expression	7-26
String Constant Expression	7-27

Section 8. Basic Constructs

Overview	8-1
Invalid and Valid Characters	8-1
Valid Character Elements.	8-2
Identifiers.	8-2
Constants.	8-3
Names.	8-4
File Names, Titles, and Directories.	8-7
Using String Primaries.	8-11
Restrictions on the Use of String Primaries.	8-12
Passing Parameters to a Task	8-12
Copying Multiple Files.	8-13

Section 9. WFL Control Options

Overview	9-1
CODE Option	9-1
ERRORLIMIT Option.	9-2
INCLUDE Option	9-2
LIST Option	9-4
LIST1 Option	9-5
NEWSEGMENT Option.	9-5
WARNSUPPRESS Option	9-6

XREF Option	9-7
XREFFILES Option	9-8

Appendix A. Sample WFL Jobs

Overview	A-1
Compiling a Program	A-1
Initiating Other Jobs	A-4
Updating Files	A-5

Appendix B. Reserved Words, Predefined Words, and Keywords

Overview	B-1
Reserved Words	B-1
Predefined Words	B-1
Keywords	B-3

Appendix C. Understanding Railroad Diagrams

Railroad Diagram Concepts	C-1
Paths	C-1
Constants and Variables	C-2
Constraints	C-3
Following the Paths of a Railroad Diagram	C-5
Railroad Diagram Examples with Sample Input	C-6

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

Tables

- 5-1. Task Attribute Groupings 5-4
- 5-2. Expressions for Task Attribute Inquiry 5-27
- 5-3. File Attribute Types 5-37
- 5-4. Expressions for File Attribute Inquiry 5-40

- C-1. Elements of a Railroad Diagram C-2

Section 1

WFL Capabilities

Work Flow Language (WFL) is used for constructing jobs that compile or run programs. WFL includes variables, expressions, and flow-of-control statements that offer the programmer a wide range of capabilities with regard to task control.

This manual is a complete language reference for WFL users, whether they are beginning users just learning WFL, or experienced users who need to check on the details of a particular construct.

Purpose and Scope

Although this manual sometimes mentions specific task attributes or file attributes, it does not describe these attributes in great detail. For detailed information regarding task attributes, refer to the *Task Attributes Programming Reference Manual*. For detailed information about file attributes, refer to the *File Attributes Programming Reference Manual*.

This manual also mentions some specific system commands that are entered at an operator display terminal (ODT). For detailed information about these commands, refer to the *System Commands Reference*.

Audience and Prerequisites

The audience for this manual consists of programmers and operators who need to write jobs that initiate and control tasks.

The user is assumed to have had prior experience with programming in a block-structured language (for example, ALGOL, COBOL, RPG, or some other block-structured language). However, there is no single programming language the user needs to know to understand this manual. Additionally, the user is assumed to be familiar with ClearPath MCP systems, at least to the extent of knowing how to create and edit files by using CANDE or the Editor.

If you are not familiar with ClearPath MCP systems, you should first read the *System Administration Guide* and the *System Operations Guide*. If you are not familiar with the concepts of process initiation and process control, you should first read the *Task Attributes Programming Reference Manual*. If you prefer to learn by example, consider reading *WFL Made Simple*.

How to Use This Manual

This manual is intended primarily for reference, although a new user can learn a great deal about WFL by reading the first five sections. [Section 6, Statements](#), begins with a functional grouping of the various WFL statements, and then presents descriptions of the statements in alphabetical order for quick reference.

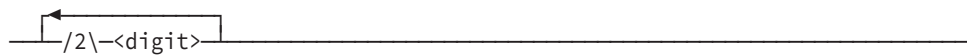
Railroad diagrams are used to illustrate WFL syntax. If you are not familiar with this type of syntax notation, you should read [Appendix C, Understanding Railroad Diagrams](#).

Some WFL constructs share a common syntax. In certain instances, these constructs share the same railroad diagram. For example:

<day interval>

<mm>

<dd>



The syntax for many WFL constructs is rather complex. To show an appropriate level of detail, most railroad diagrams use basic constructs that are explained elsewhere in the manual. You can quickly locate the more detailed syntax for these subordinate constructs using the index.

Documentation Updates

This document contains all the information that was available at the time of publication. Changes identified after the release of this document are included in problem list entry (PLE) 19242366. To obtain a copy of the PLE, contact your Unisys service representative or access the current PLE from the Unisys Product Support website:

<http://www.support.unisys.com/all/ple/19242366>

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

What's New?

New or Revised Information	Location
Added TOTALMEMORYLIMIT to Resource Usage Limits description in Table 5-1.	Section 5, Task Initiation

New or Revised Information	Location
Updated SETGROUPCODE and SETUSERCODE attribute descriptions.	ALTER Statement
Added the TRANSLATE Functions.	Section 7, Expressions

Overview of WFL

WFL is used for constructing jobs that compile or run programs. WFL is a true programming language with its own compiler that either produces an object code file used in running a job or executes a job interpretively. A WFL job is always recompiled each time it is run.

WFL can initiate programs that are written in any of the languages accepted on ClearPath MCP systems. It is a high-level language with flow-of-control structures and subroutines analogous to those found in ALGOL. However, WFL includes features related to task control that would be difficult or impossible to duplicate in any of the other available languages.

One advantage to initiating tasks through WFL instead of through CANDE or some other means is that a WFL job is automatically restarted if it is interrupted by a halt/load. This capability is discussed in [Job Restart After a Halt/Load](#).

WFL accepts jobs from a large variety of sources. With a few exceptions, identical jobs can be presented to WFL from all of the available sources. [Section 2, Job Initiation](#), describes the different sources from which WFL will accept jobs.

WFL supports the MultiLingual System (MLS). All error or warning messages given by the WFL compiler can be translated into another language with the Message Translation Utility. For more information, refer to the *MultiLingual System Administration, Operations, and Programming Guide* and the *Message Translation Utility Operations Guide*.

WFL and the MCP must be the same release level to function properly. If the release levels are not the same, WFL waits on an AX condition before freezing. It is possible to proceed with a mismatched WFL by entering AX OK, but unexpected results might occur. To declare a WFLSupport library that is the same release level as the MCP, use the System Library (SL) command. To terminate the waiting WFLSupport library, enter AX DS or use the Unlock (LP-) command and the Discontinue (DS) command.

The following pages outline the various capabilities of WFL and present example WFL jobs. While most of the WFL statements will be mentioned here, you should refer to [Section 6, Statements](#), for a more complete description of a particular statement.

Task Initiation

The two basic capabilities of WFL are compiling and running programs. The two WFL statements that correspond to these capabilities are called:

- COMPILE
- RUN

Other Task Initiation Statements

The other WFL statements that initiate tasks include:

- ARCHIVE for archiving files (using the DIFFERENTIAL, FULL, INCREMENTAL, MERGE, RESTORE, RESTOREADD, and ROLLOUT options)
- BIND for invoking the Binder
- COPY, ADD, and REPLACE for copying files
- LOG for running the LOGANALYZER utility
- MOVE for copying disk files from one disk family to another, and removing the original file
- PB for running the SYSTEM/BACKUP utility
- RESTORE, RESTOREADD, and RESTOREREPLACE for restoring files
- START for initiating other WFL jobs from the current job
- START AND WAIT for conditionally initiating a task
- WRAP and UNWRAP for wrapping and unwrapping files

Tasks initiated by any of these statements are executed serially; that is, the job waits for the task to complete before continuing on to the next statement. However, any task can be made to run asynchronously by preceding the task initiation statement with the word PROCESS. Asynchronous tasks run in parallel with the job.

The START statement normally causes the job to be compiled synchronously and executed asynchronously. A PROCESS START causes both the compile and execution to occur asynchronously with the job.

An asynchronous task can also be initiated by a subroutine invocation statement that occurs within a PROCESS statement. A subroutine invocation statement normally does not initiate a task.

Example

The following example shows a simple form of the COMPILE and RUN statements:

```
?BEGIN JOB RUNPROG;  
  COMPILE (JONES)OBJECT/ALGOL/TEST WITH ALGOL LIBRARY;  
  COMPILER FILE CARD(TITLE=ALGOL/TEST ON MYPACK);  
  RUN (JONES)OBJECT/ALGOL/TEST;  
?END JOB.
```


The COMPILE statement in this example specifies that the object code file titled (JONES)OBJECT/ALGOL/TEST is to be saved. (Other variations in the COMPILE statement can be used to specify whether the object code file is to be run immediately, saved and run immediately, or whether the source file is to be compiled for syntax only.)

The line beginning COMPILER FILE CARD is a file equation that tells the ALGOL compiler the name of the source file to use as input. The RUN statement causes the program to execute. The file title specified in the RUN statement must be the title of the object code file (not the source file) of the program.

Task Specifications

The following types of specifications can be included in a task initiation statement:

- Task attribute assignments
- Library equations
- File equations
- Database equations
- Local data specifications

Many task attributes with very diverse functions are available through WFL. Some of them can be used to access information about the task execution, such as accumulated processor and I/O time. Other functions alter the way the program is run, such as specifying the priority to be assigned to a task or the usercode a task is to run under.

The task attributes available through WFL are listed under [Task Attributes](#). For a more detailed description, refer to the task attribute descriptions in the *Task Attributes Programming Reference Manual*.

File equations can be used to modify attributes of the files used by a program, or can cause the program to use different files than it normally would have. For example, file equations can cause the program to read input from a different source than is specified in the program, and write output to a different destination than is specified in the program. This feature of WFL eliminates the need for many time-consuming alterations and recompilations of existing programs.

For more information, see [File Equations](#) and [Task Equation](#) for further information. Individual file attributes are described in the *File Attributes Programming Reference Manual*.

Examples

In the following example, the task attribute PRIORITY specifies the priority to be assigned to the task:

```
?BEGIN JOB COMP/DATA;  
  RUN (RAJA)OBJECT/COMP/DATA; % Run the desired program with  
    PRIORITY=50; % a priority of 50  
?END JOB.
```

The following example uses a file equation:

```
?BEGIN JOB WFLTEST;
  RUN (WALLY)OBJECT/ALGOL/TEST;
    FILE TERMIN(TITLE=WFLIN,KIND=DISK); % Causes the program to
                                         % read from the disk file
                                         % WFLIN, instead of the
                                         % file TERMIN
?END JOB.
```

Data Specifications

Data specifications supply input to a program that expects input from a card reader file during execution; that is, a file declared in the program as being of kind `READER`.

A data specification can also be used in place of any input file the program reads by fileequating the title of the input file to the title of the data specification in the WFL job, and by declaring the type of input file to `READER`.

Two kinds of data specifications are available:

- Local data specifications
- Global data specifications

A global data specification can be declared at the start of a job, and can be used by more than one task.

Example

The following example illustrates a job that uses a global data specification:

```
?BEGIN JOB INTREF;
DATA INPUT          % Begin global data specification
    7
    11
?                    % End global data specification
  RUN (WENDY)OBJECT/INT/REF;
    FILE TERMIN(TITLE=INPUT,KIND=READER); % Causes the program to
                                         % read from the global
                                         % data specification INPUT
                                         % instead of the file
                                         % TERMIN.
?END JOB.
```

Flow of Control Statements

The WFL jobs in the examples presented thus far have been set up to execute statements in sequential order. However, WFL also enables you to execute statements conditionally or repeatedly by using flow-of-control statements. The flow-of-control statements include:

- IF
- DO

- WHILE
- CASE
- GO

The IF statement executes a certain statement only if the value of a particular Boolean expression is TRUE.

The DO statement and the WHILE statement are both used to cause repeated execution of a statement or set of statements. Both statements evaluate a Boolean expression before each repetition is initiated, to decide whether to do it again. The DO statement differs from the WHILE statement in that it checks the Boolean expression at the end of the loop instead of at the start; the statements included in the loop are therefore guaranteed to execute at least once.

The CASE statement chooses one of several possible alternative actions, depending on the value of the variable or case expression that follows the word CASE.

Whenever the syntax of a flow-of-control statement calls for a statement to be included, that statement can be another flow-of-control statement. Therefore, flow-of-control statements can be nested to form complex flow-of-control structures.

Processing Data

The use of conditional statements implies that the job is going to be processing some data that can vary at run time. The following kinds of data can vary at run time:

- WFL jobs that are stored on disk and initiated by a START statement can have parameter values passed to them.
- During the course of a WFL job, the job can inspect the values of task attributes associated with tasks that were initiated in the job. Certain task attributes record task history. The WFL job can then make decisions about what to do next based on the history of the task that was initiated.
- The job can also inspect the values of file attributes or test whether a file with a given title is resident, and then make decisions accordingly.
- The job can include ACCEPT functions that will prompt you to supply input at run time and will cause the job to wait for your reply.

Examples

The following WFL job illustrates a use of the IF statement:

```
?BEGIN JOB WFLTEST;  
  TASK COMPOK;  
  DISPLAY "COMPILING NOW";  
  COMPILE (SFL)OBJECT/SORT/PROC WITH ALGOL [COMPOK] LIBRARY;  
  COMPILER FILE CARD(TITLE=SORT/PROC ON MYPACK);  
  IF COMPOK IS COMPILEDOK THEN  
    BEGIN  
      DISPLAY "COMPILED SUCCESSFULLY";
```

```
        DISPLAY "RUNNING SORT/PROC";
        RUN (SFL)OBJECT/SORT/PROC;
    END
ELSE
    DISPLAY "COMPILE NOT SUCCESSFUL --- WILL NOT RUN";
?END JOB.
```

This example first compiles a program and then inspects a task variable named COMPOK to see whether the compile was successful. If the compile was successful, the Boolean expression COMPOK IS COMPILEDOK evaluates to TRUE. In this case, the program that was compiled will be run. If the compile was not successful, then no attempt is made to run the program.

The DISPLAY statements in this example display messages at the ODT (and also at the CANDE terminal where the job was initiated).

The following example illustrates one use of the CASE statement:

```
?BEGIN JOB WFLTEST(String COMPTYPE);
  CASE COMPTYPE OF
    BEGIN
      ("SYNTAX"):      COMPILE (MAINT)OBJECT/ORD WITH COBOL74 SYNTAX;
                       COMPILER FILE CARD(TITLE=ORD,KIND=DISK);
      ("GO"):          COMPILE (MAINT)OBJECT/ORD WITH COBOL74 GO;
                       COMPILER FILE CARD(TITLE=ORD,KIND=DISK);
      ("LIBRARY"):     COMPILE (MAINT)OBJECT/
ORD WITH COBOL74 LIBRARY;
                       COMPILER FILE CARD(TITLE=ORD,KIND=DISK);
      ("LIBRARY-GO"):  COMPILE (MAINT)OBJECT/
ORD WITH COBOL74 LIBRARY GO ;
                       COMPILER FILE CARD(TITLE=ORD,KIND=DISK);
      ELSE: DISPLAY "INCORRECT COMPILE TYPE ENTERED";
    END;
?END JOB.
```

In this example, a WFL job has been constructed that enables a user to compile a particular file in any of four possible ways according to the string parameter specified in the START statement.

The following table lists four START statements, each with a different string parameter specified and the results of each statement.

Statement	Result
START WFLTEST("SYNTAX")	Compiles the file ORD to check for syntax errors, but does not save the object code file.
START WFLTEST("GO")	Compiles and runs the file ORD, but does not save the object code file.
START WFLTEST("LIBRARY")	Compiles the file ORD, and saves the object code file as (MAINT)OBJECT/ORD.

Statement	Result
START WFLTEST("LIBRARY-GO")	Compiles the file ORD and runs the object code file. The object code file is saved as (MAINT)OBJECT/ORD.

The following is an example of the DO statement:

```
?BEGIN JOB WFLTEST( INTEGER REPS );
  INTEGER INTREPS := 0;
  DO BEGIN
    RUN (WALLY)OBJECT/STOCK/PLAN;
    INTREPS := INTREPS + 1;
  END
  UNTIL INTREPS = REPS;
?END JOB.
```

This job runs the program (WALLY)OBJECT/STOCK/PLAN the number of times specified in the parameter of the START statement that initiated this job. For instance, START WFLTEST(7) runs the program (WALLY)OBJECT/STOCK/PLAN seven times.

Subroutine Control

A WFL job can include any number of subroutines, and each subroutine can contain further subroutine declarations within itself. In addition, a subroutine can contain statements invoking itself, or other subroutines in the WFL job. (The rules concerning which subroutines can be invoked from a given subroutine are discussed under [Scope of Declarations](#)).

WFL enables an identifier to be associated with a statement or group of statements. This can be accomplished by using the subroutine declaration. Once a subroutine has been declared, it can be invoked in the WFL job by using the identifier associated with it. The subroutine is a convenient shorthand for referring to a piece of code that the WFL job is going to invoke repeatedly.

Examples

The WFL subroutines can be invoked with parameters that pass values to the subroutines. Consider the following example:

```
?BEGIN JOB WFLCHART(REAL VALUE1, REAL VALUE2, REAL VALUE3);
  SUBROUTINE RUNCHART(REAL RVAL VALUE); % Beginning of subroutine
  BEGIN % declaration
    RUN (PORT)CHART/COL/NAV(RVAL);
  END RUNCHART; % End of subroutine
  % declaration
  % WFL job statements follow
  DISPLAY "RUNNING WITH FIRST VALUE SUPPLIED";
  RUNCHART(VALUE1);
  DISPLAY "RUNNING WITH SECOND VALUE SUPPLIED";
  RUNCHART(VALUE2);
```

```
    DISPLAY "RUNNING WITH THIRD VALUE SUPPLIED";
    RUNCHART(VALUE3);
?END JOB.
```

This job is initiated with a START statement that has three real numbers specified as parameters. The job then calls on the subroutine RUNCHART three times, each time supplying a different parameter value to the subroutine. This method saves code repetition when a lengthy subroutine is called.

The following example is a subroutine that invokes itself:

```
?BEGIN JOB WFLINV(INTEGER VALUE1, INTEGER VALUE2);
    SUBROUTINE DISPLAYONE;
        DISPLAY "IT WORKED ALL RIGHT";
    SUBROUTINE RUNINV(INTEGER IVAL1);
    BEGIN
        RUN (PARTS)OBJECT/OLD/INV(IVAL1);
        STATION=MYSELF(SOURCESTATION);
        DISPLAYONE;
        IVAL1 := IVAL1 + 1;
        IF IVAL1 LEQ VALUE2 THEN
            RUNINV(IVAL1);
        END;
    RUNINV(VALUE1);
    DISPLAYONE;
?END JOB.
```

This job accepts two integer values as parameters. The job uses the first job parameter as a parameter in the RUN statement for the program OBJECT/OLD/INV. This program is run several times, with an increase in the value of the parameter each time, until the parameter equals the value of the second parameter that was supplied to the job.

This example shows several kinds of subroutine invocations that are available. The outer block of the WFL job calls on the subroutines RUNINV and DISPLAYONE. The subroutine RUNINV also calls on the subroutine DISPLAYONE, which was previously defined. Further, the subroutine RUNINV calls on itself, so that it will continue repeating as long as the expression in the IF statement evaluates to TRUE. The following example illustrates the use of the RETURN statement as a means of exiting a subroutine early:

```
?BEGIN JOB WFLTEST(INTEGER VALUE1);
    TASK T;
    SUBROUTINE RUNTEST(INTEGER IVAL1);
    BEGIN
        RUN (WALLY)OBJECT/TEST(IVAL1) [T];
        IF T ISNT COMPLETEDOK THEN
            RETURN;
        IVAL1 := IVAL1 + 1;
        IF IVAL1 LEQ 7 THEN
            RUNTEST(IVAL1);
        END;
    RUNTEST(VALUE1);
?END JOB.
```

This example runs the program (WALLY)OBJECT/TEST repeatedly. However, if the task variable T indicates the program did not run successfully, the RETURN statement is invoked to cause the subroutine to be exited. The remaining statements in the subroutine will then be bypassed, and control will pass back to the parent of the subroutine (in this case, the outer block of the WFL job).

Task Control

Several WFL statements are available to control the execution of a task. These statements determine

- When a task is run
- When a task is terminated
- How a task responds to error conditions
- What usercode a task runs under

These statements are referred to broadly as task control statements.

Task Control Statements

Task control statements include the following:

- ABORT
- STOP
- WAIT
- ON
- USER

The ABORT and STOP statements are available for terminating a job or task prematurely. The difference between the two statements is that the ABORT statement displays messages indicating that the job or task was terminated abnormally, while the STOP statement indicates a normal termination. Generally these statements are used as part of an IF statement or CASE statement, so that the job or task is terminated only if the specified conditions are met.

The WAIT statement suspends job execution until some specified condition is met. The uses of the WAIT statement include the following:

- To cause the job to wait for an asynchronous task to complete, or to wait until the asynchronous task achieves a specified state
- To cause the job to wait until a file becomes resident
- To cause the job to wait for an OK message from the user

Examples

This example illustrates a use of the ABORT statement:

```
?BEGIN JOB WFLTEST;
  TASK COMPOK;
  COMPILE (WALLY)OBJECT/MENU/PLAN WITH PASCAL [COMPOK] LIBRARY;
  COMPILER FILE CARD(TITLE=MENU/PLAN ON MUNCHPACK);
  IF COMPOK IS COMPILEDOK THEN
    RUN (WALLY)OBJECT/MENU/PLAN
  ELSE
    ABORT "UNSUCCESSFUL COMPILE";
?END JOB.
```

This job first attempts to compile the source file MENU/PLAN. If the compile was successful, the job will run the object code file that was the result of the compilation. If the compile was not successful, the job is terminated abnormally by the ABORT statement and the message "UNSUCCESSFUL COMPILE" is displayed.

The following example illustrates a use of the WAIT statement:

```
?BEGIN JOB WFLTEST;
  TASK TEST1, TEST2;
  PROCESS RUN (WALLY)OBJECT/SORTIT [TEST1];
  PROCESS RUN (WALLY)OBJECT/SORT2 [TEST2];
  DO
    WAIT
  UNTIL TEST1 IS COMPLETED AND TEST2 IS COMPLETED;
  RUN (WALLY)OBJECT/COLLATE;
?END JOB.
```

In this example, two asynchronous tasks are started by PROCESS statements. A DOUNTIL loop follows, instructing the job to wait until both asynchronous tasks are completed before continuing.

The USER statement provides another form of task control. This statement changes the usercode the WFL job is running under to the specified usercode. The tasks initiated by the WFL job will then inherit the new usercode, and its associated privileges.

The following is an example of the USER statement in a job:

```
?BEGIN JOB RECEIVABLE;
  RUN OBJECT/INTEREST;

  USER=SCROOGE/CALC;
  RUN OBJECT/PENALTIES;
  RUN (CRATCHIT)OBJECT/CREDIT;
?END JOB.
```

In this example, the first RUN statement runs the program OBJECT/INTEREST under the job's original usercode, which was determined at the time of job initiation. The USER statement then changes the job's usercode to usercode SCROOGE, which has CALC as its password. The remaining two programs that the job initiates are then run under usercode SCROOGE, even though the object code file for the last one resides under usercode CRATCHIT.

The usercode of the job can also be set by specifying the USERCODE task attribute as a job attribute, and the usercode for an individual task can be set by specifying the USERCODE for that task.

File Handling

Although a WFL job cannot read from or write to files itself, it provides considerable control over the files that programs initiated through WFL can read from or write to. The main tool that provides this control is file equation, which is discussed under [Task Initiation](#) earlier in this section. In addition, there are several statements available in WFL that provide even more flexibility in file handling.

File Handling Statements

Files can be opened in WFL by using the OPEN statement. The file specified in this statement is opened with the I/O capabilities specified in the NEWFILE attribute, FILEUSE attribute, or other attributes of the file. If the NEWFILE attribute or the FILEUSE attribute is not assigned, an attempt is made to open the specified file for both input and output. Refer to the *File Attributes Programming Reference Manual* for more information about file attributes.

The following table lists five statements that close files in different ways.

Statement	Result
CRUNCH	Closes and releases a file, and returns the unused portion of the last row of the file to the system. Once a file is crunched, it takes up less space, but can no longer be expanded. This statement is used for disk files only.
LOCK	Closes, releases, and locks a file. If the file is a disk file, it is kept as a permanent file on disk. If the file is a tape file, the tape is rewound and unloaded, making the unit inaccessible to the system until it is readied again manually.
PURGE	Removes a file and frees the area it was occupying for reuse.
RELEASE	Closes and releases a file.
REWIND	Closes a file and rewinds it. For a tape file, this means that the tape is rewound. For a disk file, the record pointer is reset to the first record of the file.

A WFL job can interrogate the attributes of a file. To do this, the file must first be declared in the job with certain minimal attributes specified. In addition, the file must be opened.

Examples

The following job examines a file to determine whether it is a COBOL74 or COBOL85 file, and chooses the corresponding compiler:

```
?BEGIN JOB COBOLCOMP (STRING SYMBOL) ;
  FILE SYMBOLF ;
  STRING COMPNAME ;
  SYMBOLF (TITLE=#SYMBOL,KIND=DISK,NEWFILE=FALSE,
          DEPENDENTSPECS=TRUE) ;
  OPEN (SYMBOLF) ;
  IF SYMBOLF (FILEKIND) = "COBOL74SYMBOL" THEN
    COMPNAME := "COBOL74"
  ELSE
    COMPNAME := "COBOL85" ;
  COMPILE OBJECT/#SYMBOL WITH #COMPNAME LIBRARY GO ;
  COMPILER FILE CARD (TITLE=#SYMBOL,KIND=DISK) ;
?END JOB.
```

The following subroutine uses the file opening and closing capabilities of WFL to create dummy files (empty files that can be used to indicate that some particular event has occurred in the job). Because subroutines cannot contain file declarations, file F is declared globally.

```
SUBROUTINE MAKE (STRING FILENAME) ;
BEGIN
  F (TITLE = #FILENAME, KIND = DISK, NEWFILE = TRUE) ;
  OPEN (F) ;
  LOCK (F) ;
END MAKE ;
```

Refer to [Job Restart After a Halt/Load](#) for an example of why you might want to use a job to create dummy files.

File Management

WFL provides several statements for managing files. These include the following:

- COPY statement
- ADD statement
- CHANGE statement
- REMOVE statement
- SECURITY statement

File Management Statements

The COPY statement copies files on disk or tape. The new copy of a file can be created with a different usercode or file name, and the file can be copied to other disk or tape units using library maintenance or to other hosts using distributed systems services (DSS).

The ADD statement also copies files. However, the ADD statement will not copy a file if a file with the same title already resides at the destination. Also, the destination specified must be a disk. The ADD statement is useful for adding a directory of files to a disk where some of the files are already resident and are to be preserved.

The CHANGE statement changes the names of files on a disk. The REMOVE statement removes files from disk.

The SECURITY statement changes the security specifications of files on a disk. The various security settings affect whether the file can be read or changed by tasks running under other usercodes.

Examples

The following example copies a file from a disk to a tape, specifying a different usercode and file name for the new copy:

```
COPY (ID)FILEA AS (JONES)NEWF FROM ORDS (DISK) TO SAVEA (TAPE);
```

The following example copies a directory of files to a disk on another host:

```
ADD (ID)F/= FROM ORDS (DISK) TO SAVEA (DISK, HOSTNAME=MLM);
```

The following example changes the name of a directory of files residing on a pack named STORPACK:

```
CHANGE COMPJOBS/= ON STORPACK TO QCOMPS/=;
```

The following example removes files from two different packs:

```
REMOVE MAXERRS FROM PARTS, BADERRS FROM STORPACK;
```

The following example changes the security of a file to PUBLIC (thus enabling any user free access to the file):

```
SECURITY NOVA/DATA/NINER PUBLIC;
```

Communication

WFL includes several statements that help inform you about what a job is doing. These statements include the following:

- DISPLAY statement
- ACCEPT function
- INSTRUCTION statement
- FETCH specification

In addition, these statements enable input to modify the activity of a job.

Communication Statements

The DISPLAY statement can be used to display a message at a specific point during job execution. The DISPLAY message appears at the ODT, in the job log, and at the CANDE or MARC station that initiated the job (if it was initiated through CANDE or MARC).

The ACCEPT function will ask you to supply input to the job while it is running. The ACCEPT function will display the specified message and will wait for you to return a value by way of the AX (Accept) system command.

The INSTRUCTION statement stores information about the job in a form that you can display at any time during job execution by using the IB (Instruction Block) system command.

The FETCH specification stores a message that you can display by using a PF (Print Fetch) system command, and suspends initiation of the job until the message has been displayed. The FETCH specification is intended to be used for informing you about what resources are required by the job.

Examples

The following example includes DISPLAY statements:

```
?BEGIN JOB WFLTEST (STRING DAY);  
  DISPLAY "CALL WILLIS AT EX. 3574 IF RUN-TIME ERRORS OCCUR";  
  RUN (ODDCOM)OBJECT/COPY/FACTORS(DAY);  
  DISPLAY "GET INPUT FROM ARCH. TAPE 143 IF FILE REQUIRED";  
  RUN (ODDCOM)OBJECT/FT/VERIFY(DAY);  
?END JOB.
```

This example runs two programs that are each preceded by a display message informing the user what to do if certain conditions occur.

The following example shows a use of the ACCEPT function:

```
?BEGIN JOB;  
  STRING FN, PK;  
  FN := ACCEPT("ENTER NAME OF FILE DESIRED");  
  PK := ACCEPT("ENTER NAME OF PACK DESIRED");  
  RUN (OPR)DAILY/UPDATE;  
  FILE IN (TITLE = #FN ON #PK);  
?END JOB.
```

In this example, the values submitted by you are used in a file equation for a RUN statement. The job could have been written to receive the same values from you by way of a start parameter list. However, in that case you would have to know in advance what parameters to supply. The ACCEPT function, on the other hand, displays a message informing you what type of information to enter next.

Job Format

The following features are available to increase the readability of a WFL code:

- Free formatting
- Case-insensitivity
- Comments
- Subroutine ending identifiers

Free formatting enables WFL constructs to be entered in any column of a line and continue over any number of lines. You can use varying amounts of indentation to indicate the nesting of constructs. However, the invalid character must appear in the first column.

WFL is case-insensitive. Keywords and variables can be in uppercase, lowercase, or mixed case. However, if the CASESENSITIVEPW security option is set, passwords are treated in a case-sensitive manner; lowercase characters are not converted to uppercase, and some special characters can be included without enclosing the token in double quotes. The following special characters are accepted in such a password.

&	-	{	}	\	~
'	[]		:	\$
<	*	@	(_	+
>	=	!	^	?	#

The pound sign (#) cannot be the first character of the password, and the ampersand (&) cannot be used when the password is used in a WRAP or UNWRAP statement unless the password is enclosed in double quotes.

A percent sign (%) can appear in any column of a line, and will cause the WFL compiler to ignore the remainder of the line, which can be used for comments.

The identifier that identifies a subroutine can be repeated after the END statement for that subroutine. In cases where subroutines are nested, this feature helps eliminate confusion about which subroutine is being ended.

Example

The following example illustrates the use of a nested subroutine ending identifier:

```

SUBROUTINE SUBOUTER;      % Beginning of outer subroutine
BEGIN
  SUBROUTINE SUBNESTED; % Beginning of nested subroutine
  BEGIN
    .
    .
    END SUBNESTED;        % End of nested subroutine
  .
  .
  END SUBOUTER;           % End of outer subroutine

```


Section 2

Job Initiation

Overview

A program written in WFL is referred to as a job. During execution, a job normally executes as a task with its own object code file. However, certain statements can be executed interpretively. This means that no object code file is produced by the WFL compiler; the statement is executed directly by the WFL compiler.

The statements ALTER, CHANGE, PRINT, REMOVE, RERUN, SECURITY, and START are executed interpretively in the following situations:

- When one of the above statements is the only statement appearing after the CANDE WFL command
- When one of the above statements (except the PRINT statement) is entered individually at an ODT, rather than as part of a job
- When a job originating from a user program by way of an array consists only of one of the above statements

These situations are described in detail in [START and WFL Commands from CANDE Sessions, Operator Display Terminals \(ODTs\)](#), and [User Programs in Other Languages](#) later in this section.

A WFL job can initiate other tasks. Examples of tasks are compilations and executions of user programs. The job task controls the execution of the tasks it initiates.

Sources for Job Initiation

The following pages describe how to initiate WFL jobs from each of the possible sources, and discuss limitations specific to each source.

START and WFL Commands from CANDE Sessions

To run WFL jobs from a CANDE session, use the WFL command or the START command.

WFL Command

The WFL command causes CANDE to pass any following text to the WFL compiler. This enables a user to type WFL, followed by a WFL statement or even a job, and transmit the entire job at once. The WFL job can extend over any number of lines.

The phrases BEGIN JOB and END JOB can be omitted from a job entered after the WFL command in CANDE. Refer to [Section 3, Job Structure](#), for the formal syntax of a job.

Note: *Jobs submitted through the CANDE WFL statement cannot include a CLASS, FETCH, or STARTTIME specification. In addition, the job cannot contain data specifications, or the INCLUDE control option. Any WFL control options must be followed by a semicolon (;) to separate them from the rest of the job.*

When WFL jobs are initiated from a CANDE session in this way, any messages generated by the job are displayed at the terminal where the job originated, as well as at the ODT. The terminal also queues CANDE commands entered while the WFL job was running and executes them after the job is completed. CANDE control commands the exception to this rule; they are executed immediately.

The statements CHANGE, PRINT, REMOVE, RERUN, SECURITY, and START are executed interpretively if one of them is the only statement following the CANDE WFL command. This means that no object code file is produced by the WFL compiler in these cases; the statement is executed directly by the WFL compiler.

Storing Jobs in Disk Files

WFL jobs can also be saved in files on disk. To do this, you should first use the CANDE MAKE command to create a file of type JOB. You can then enter the complete job into the file, either in CANDE, the Editor, or some other text entry utility. For further details, refer to the *CANDE Operations Reference Manual* and the *Editor Operations Guide*.

The question mark (?) preceding the BEGIN JOB and END JOB constructs in column 1 can cause strange results in some text entry utilities. In these cases, it might be necessary to enter the question mark in column 2, and shift the line to the left later.

More than one job can be stored in a single disk file. The jobs will be compiled and executed in the order that they appear in the file.

Note: *If a file contains more than one job, none of the jobs can include a job parameter list.*

WFL can open a PUBLIC SECURED file (or one guarded with a guardfile allowing EXECUTE access.) WFL can use such files as the job source file and as included files (files identified in a \$INCLUDE compiler option specification).

The EXECUTE attribute of the file is updated if compilation is successful and not for syntax. If the file contains more than one job, the attribute is only updated if the last job compiles successfully.

START Command

A WFL job saved in a file can be initiated using the CANDE START command, followed by the name of the file, and any parameters being passed to the WFL job. Refer to the *CANDE Operations Reference Manual* for the detailed syntax of this command.

The START command passes the contents of the file directly to the WFL compiler for processing. The contents of the file must be a complete WFL job, including the BEGIN JOB and END JOB constructs. The job can include data specifications, unlike WFL jobs submitted through the CANDE WFL command.

Any messages from the job are displayed at the originating terminal as well as at the ODT. However, the terminal remains free to accept any CANDE commands that are entered and responds immediately, even if they are not control commands.

If you use the START command in CANDE to start a JOBSYMBOL or DATA file that is secured, CANDE returns #FILE NOT AVAILABLE. Use WFL START from CANDE to start the job.

Other CANDE Commands

The following CANDE commands directly invoke WFL statements that have the same names:

- ADD
- ARCHIVE
- CATALOG
- COPY
- MODIFY
- PRINT
- START
- UNWRAP
- WRAP

Therefore, it is not necessary to precede these commands with WFL. Several other CANDE commands have the same names as WFL statements, but do **not** invoke WFL, and in some cases have different syntax or functions than their WFL counterparts. These statements include:

- | | |
|-----------|------------|
| • ACCESS | • PASSWORD |
| • BIND | • REMOVE |
| • CHANGE | • RUN |
| • COMPILE | • SECURITY |
| • DO | • STOP |
| • LOG | |

To invoke these WFL statements in CANDE, you must precede the statements with the CANDE WFL command, or include them in a job file and initiate the job with a START command.

Note: WFL jobs originated from CANDE sessions inherit the usercode of the CANDE session unless a USERCODE job attribute specification is included in the job. The job inherits the family specification associated with the usercode it is running under unless a FAMILY job attribute specification is included in the job.

START Statements from Running WFL Jobs

Running WFL jobs can initiate other WFL jobs through the WFL START statement, which is described in detail in [Section 6, Statements](#). The WFL START statement has the same effect as the CANDE START statement.

Only jobs stored on disk can be initiated by the START statement. Such a job can include any of the WFL constructs defined in this manual.

Once the job has been started, it has no further connection with its parent job. Discontinuing the parent job does not affect the job that was started from it, and termination of the started job also does not affect the parent job.

Any messages generated by the started job are directed to the same source as messages from the parent job. For example, if the parent job is initiated from CANDE, then messages from both the parent job and the job started by the parent job would appear at the originating terminal.

Note: The started job inherits the usercode and family assigned to the parent job unless the started job includes USERCODE or FAMILY job attribute specifications.

Operator Display Terminals (ODTs)

Initiate WFL jobs from an ODT using one of the following methods:

- Use the START statement to initiate a job stored in a disk file.
- Type the complete job at the ODT and transmit the job.

When a complete job is entered at the ODT, it should begin with the BEGIN JOB construct. It is not necessary to append the END JOB construct to the job, since WFL will add this automatically.

Note: A complete job entered at an ODT cannot include data specifications. Any WFL control option must be followed by a semicolon (;) to separate it from the rest of the job.

A number of WFL statements are recognized by the CONTROLLER and are passed to WFL automatically, even if they are not preceded by a BEGIN JOB heading. These statements include the following:

- | | | |
|-----------|------------|------------|
| • ABORT | • CATALOG | • REMOVE |
| • ACCESS | • CHANGE | • RERUN |
| • ADD | • COMPILE | • RUN |
| • ALTER | • COPY | • SECURITY |
| • ARCHIVE | • DISPLAY | • START |
| • BEGIN | • PASSWORD | • USER |
| • BIND | • PROCESS | • VOLUME |

A WFL job initiated at an ODT runs without a usercode unless the job that includes a USERCODE job attribute specification is run from an ODT with TERM USER specified, or the HU system command has an associated usercode. The HU system command designates a usercode for certain distributed systems services Host Services requests if they come from an ODT that has no terminal usercode assigned to it. The TERM USER system command controls the format of all displays on the ODT, and associates the usercode with certain requests initiated at the ODT where the command is entered. See the *System Commands Reference* for more details on these commands.

The statements ADD, ARCHIVE, CATALOG, COPY, MODIFY, MOVE, REPLACE, RESTORE, RESTOREADD, RESTOREREPLACE, UNWRAP, VOLUME and WRAP, when run without a usercode, are privileged and can access any file, volume family, catalog entry or archive entry.

Note: *The job, and any tasks initiated by the job, will therefore be unable to access files residing under usercodes unless their security is PUBLIC. In addition, if the job is running without a usercode, and no FAMILY specification is included in the job, the FAMILY will default to DISK = DISK ONLY.*

The statements ALTER, CHANGE, REMOVE, RERUN, SECURITY, and START are executed interpretively if entered individually at an ODT (rather than as part of a job). When this occurs, no object code file is produced by the WFL compiler; the statements are executed directly by the WFL compiler and do not enter a job queue. Thus, no queue attributes, such as FAMILY, are inherited by the job or the statement. These particular statements are treated as privileged if entered in this way, and can be used to affect files residing under usercodes.

Menu-Assisted Resource Control (MARC)

WFL jobs can be initiated through MARC from the action line of a screen that includes "COmnd" as one of the screen action hints displayed on line 3, or from the command input screen, which provides an entry field 15 lines long.

Any WFL statements or a complete WFL job can be initiated from MARC by entering the word WFL before the statements. The BEGIN JOB and END JOB constructs can be omitted. However, jobs submitted in this way cannot include data specifications or a STARTTIME specification.

Note: *If WFL control options are included, they must be followed by a semicolon (;). If any job parameters are included, the job can be compiled for syntax only.*

Three WFL statements are recognized by MARC and passed to WFL even if they are not preceded by WFL. Each statement has the same syntax when it is used in a WFL job. These statements include:

- START
- COPY
- ADD

Jobs initiated through the START statement can include all of the WFL constructs defined in this manual. Initiating a WFL job with any of the above statements causes MARC to enter tasking mode and display the task status screen. Refer to the *MARC Operations Guide* for information about how to display and control task progress from tasking mode.

Distributed Systems Services

WFL jobs can be constructed to take advantage of distributed systems services between systems in several ways. For more information about how to use distributed systems services, refer to the *Distributed Systems Services Operations Guide* and the *TCP/IP Distributed Systems Services Operations Guide*.

If a terminal is attached to CANDE, then the names of available BNA hosts can be displayed using the CANDE ?HN command. The terminal can be transferred to another of the listed BNA hosts by entering the command:

```
CONNECT TO <host specification>
```

When the connection is complete, you can log on under a usercode recognized by the remote host. While the terminal is connected to the remote host, any commands you enter are directed to the remote host and executed there. The CANDE WFL and START commands can thus be used to run jobs on the remote host. The connection is terminated when you end the session with a BYE command.

Even if a terminal is not connected to a remote host, a job initiated at that terminal can be directed to run at a remote host by beginning the job with the AT <hostname constant>.

Note: *A job including AT <hostname constant> cannot contain a job parameter list. Also, the <i> construct before the END JOB construct is required.*

Any messages generated by the job are still routed back to the original terminal, but are preceded by the name of the system the job is running on.

Examples

The following is an example of a job with an AT <hostname constant> specification. This job is stored in a file on the user's own system. However, when the job is initiated, it will run on the system named LA15D. The job will also look on LA15D for the object code files of the tasks that it runs.

```
?AT LA15D BEGIN JOB RUNNIT;
  RUN (WALLY)OBJECT/MAKEIT ON SHIPPK;
  RUN (ODCON)SNOBOL/REPL ON ORDSPK;
?END JOB.
```

Tasks can be made to run on different systems than the parent job by specifying the HOSTNAME attribute after the task initiation statement. The object code files for the tasks are also searched for on the system specified by the HOSTNAME attribute. Messages generated by the tasks are still routed back to the original terminal, but are preceded by the name of the system the job is running on.

```
?BEGIN JOB RUNNIT;
  RUN (WALLY)OBJECT/MAKEIT ON SHIPPK;
    HOSTNAME=SF15B;
  RUN (ODCON)SNOBOL/REPL ON ORDSPK;
    HOSTNAME=SD9A;
?END JOB.
```

This job runs on the user's own system. However, it initiates tasks that run on systems SF15B and SD9A, respectively. The job looks for the object code files for those tasks on the systems that the tasks are run on.

Files can be copied between MCP hosts, BNA hosts, and hosts connected to a TCP/IP network by using the WFL COPY statement, which can reference sources or destinations on remote hosts. For example:

```
?BEGIN JOB COPYDATA;
  COPY *SYSTEM/CRUNCHER AS (WALLY)CRUNCHER/TUESDAY FROM ODDPACK
    (KIND=DISK,HOSTNAME=SF15B) TO MODPACK (KIND=DISK,HOSTNAME=LA15D);
?END JOB.
```

This example copies a file from the system named SF15B to the system named LA15D. The two hosts are BNA hosts and the file is transferred using Host Services file transfer which is part of the distributed systems services.

User Programs in Other Languages

WFL jobs can be initiated from user programs written in any of several different programming languages through the use of certain statements. The following table lists the different programming languages and the statements of each that initiate WFL jobs from user programs.

Language	Language Statement	Reference Document
----------	--------------------	--------------------

ALGOL	ZIP statement	<i>ALGOL Programming Reference Manual, Volume 1: Basic Implementation</i>
COBOL74	CALL SYSTEM WFL statement	<i>COBOL ANSI-74 Programming Reference Manual, Volume 1: Basic Implementation</i>
COBOL85	CALL SYSTEM WFL statement	<i>COBOL ANSI-85 Programming Reference Manual, Volume 1: Basic Implementation</i>
DCALGOL	CONTROLCARD function	<i>DCALGOL Programming Reference Manual</i>
RPG	ZIP operation code	<i>Report Program Generator (RPG) Programming Reference Manual, Volume 1: Basic Implementation</i>

Input submitted from such programs can be in array form or in file form.

Note: *If the job submitted is from a data array contained in the program, the job cannot contain any data specifications, and any WFL control options included in the job must be followed by a semicolon (;).*

If the job is stored in a file external to the program, it **can** include any WFL construct defined in this manual. However, if the job contains a job parameter list, it can only be initiated by a START statement, which can be submitted in array form.

Jobs initiated from user programs inherit the usercode and associated privileges of the initiating program.

Jobs originating from a user program by way of an array, and that consist of a single CHANGE, PRINT, REMOVE, RERUN, SECURITY, or START statement, are executed interpretively. That is, an object code file is not generated, and the statement is executed directly by the WFL compiler. These statements are not considered privileged unless the user program is privileged.

Magnetic Tapes

The preferred method for initiating jobs stored in files on tape is to first copy the files onto disk and then start them. The files can be copied using the WFL COPY statement, either from CANDE or as part of another WFL job. Once the files are on disk, they are started using the CANDE START command, a START statement in another WFL job, or the MARC START command.

If a site is attempting to conserve disk space, the job file can be removed from disk after each use and recopied from the tape whenever it is needed. The job can even include a REMOVE statement that would remove its own job file from disk after each use.

Job Continuation After a Task Fails

A WFL job can be written to take special action in the event that a task terminates abnormally. The following features are available:

- The ON TASKFAULT statement can be used to direct the job to execute a particular statement or set of statements whenever a task of the job fails. Refer to [ON Statement](#) for details.
- The STATUS, HISTORYCAUSE, and HISTORYTYPE attributes of a task can be interrogated after a task terminates. An IF statement or CASE statement can be used to cause different actions to be taken according to the values of these attributes. Refer to [Interrogating Task Status](#) for details on STATUS. HISTORYCAUSE and HISTORYTYPE are both mnemonic task attributes. Refer to [Interrogating Complex Task Attributes](#). Task attributes are also discussed in the *Task Attributes Programming Reference Manual*.
- The task state expression can be used to determine whether a task terminated normally. An IF statement or CASE statement can be used to cause different actions to be taken according to the value of this expression. Refer to [Boolean Expressions](#) for details.

The abnormal termination of a task does not affect the job; the job continues to execute and proceeds to the statement following the one that initiated the task.

Job Restart After a Halt/Load

WFL jobs interrupted by a halt/load are automatically restarted after the halt/load. Execution of the job begins where it left off before the halt/load; if a task was in progress, the task is restarted. Keep the following considerations in mind when writing a WFL job to ensure that it will restart properly after a halt/load.

As a WFL job is executing, information about the job is saved off so the job can be restarted in the event of a halt/load. The process of saving off the job information is called job rollout. When a job rollout is done, the values of integer, real, Boolean, and string variables are saved so that these values can be restored to the correct values when the job is restarted. If a system halt/load occurs while a job is running, the job will be restarted by the MCP at the most recent successful job rollout.

In general, a job rollout is attempted before each task is run. This way, only the task that was executing when the halt/load occurred will have to be restarted. However, there are some restrictions on when a successful job rollout can be accomplished. If any asynchronous tasks are active when a job rollout is attempted, the job rollout cannot be completed and will be skipped. If a job rollout has to be skipped, the job can still be restarted after a halt/load; however, the previous successful job rollout will be retained as the current restart point.

In the following example, if a halt/load occurs when OBJECT/PROG2 is running, the job will be restarted at "job rollout point #2". Thus, the job will be restarted just prior to the initiation of OBJECT/PROG2.

```
?BEGIN JOB RESTART/EXAMPLE/1;
% job rollout point #1
RUN OBJECT/PROG1;
% job rollout point #2
RUN OBJECT/PROG2;      % Executing OBJECT/PROG2 when halt/
load occurs
?END JOB.
```

For the asynchronous task case, consider the following example:

```
?BEGIN JOB RESTART/EXAMPLE/2;
% job rollout point #1
PROCESS RUN OBJECT/PROG1;    % Note: OBJECT/PROG1 is PROCESSED
% job rollout point #2
RUN OBJECT/PROG2;           % Both OBJECT/PROG1 and OBJECT/PROG2
                             % are active when halt/load occurs
?END JOB.
```

In this example, the job rollout that is attempted at “job rollout point #2” will not be successful (because the task OBJECT/PROG1 is active). “Job rollout point #1” will still be the current restart point. If a halt/load occurs during the execution of OBJECT/PROG2, the job will be restarted at “job rollout point #1”.

A job rollout is attempted before each of the following statements:

- ALTER
- ARCHIVE
- Asynchronous subroutine invocation
- CHANGE
- COMPILE
- COPY
- LOG
- MODIFY
- OPEN
- PRINT
- REMOVE
- RUN
- START
- WAIT

If one of the following statements contains an ACCEPT function, a job rollout is attempted before the statement:

- CASE
- Assignment statement
- DO

- File attribute statement
- IF
- Task attribute statement
- WHILE

A job rollout is also attempted after the WAIT statement. The WAIT statement is the only statement where a job rollout is executed both before and after the statement. This occurs because the WAIT statement is often used to synchronize processed tasks, and attempting the job rollout both before and after the WAIT statement helps to ensure that a successful job rollout is completed.

When waiting for a long task to be completed before initiating a halt/load, it is important to consider that the job rollout is executed by the WFL job after the task is completed. Waiting for the task to go to “End of Task” is not sufficient to ensure that the job rollout is completed. For example:

```
?BEGIN JOB RESTART/EXAMPLE/3;
% job rollout point #1
PROCESS RUN OBJECT/PROG1;
% job rollout point #2 (not successful because of active task)
WAIT(OK);
% job rollout point #3
.
.
.
?END JOB.
```

If the intention is to wait until OBJECT/PROG1 is completed before halt loading (so PROG1 will not have to be restarted), it is important to allow “job rollout point #3” to be executed. The job would restart at “job rollout point #1”, which obviously would not be the desired result.

The following example illustrates how to avoid the potential problem by enabling the job rollout to be successfully completed before the possible halt/load:

```
?BEGIN JOB RESTART/EXAMPLE/4;
TASK T1,T2;
PROCESS RUN OBJECT/PROG1 [T1]
PROCESS RUN OBJECT/PROG2 [T2]
WAIT (T1 IS COMPLETED);
WAIT (T2 IS COMPLETED)
%% A successful job rollout will be executed here %%
WAIT ("Okay to halt/load now", 60);
.
.
.
?END JOB.
```

However, the contents of most file or task variables are not saved across a halt/load. They will have the following values:

- Most of the attributes of file variables are returned to what they were immediately after the original file declaration.
- Most of the attributes of task variables are set to their system default values. This has the same effect as reinitializing the task variable with the INITIALIZE statement. Any task attribute assignments that occurred prior to the halt/load are not restored, including any that were included in the task variable declaration.

The values of constant identifiers that appear in the job parameter list are saved across a halt/load.

ON RESTART Statement

The ON RESTART <statement> form of the ON statement can be used to specify actions to be taken if job execution is interrupted by a halt/load. Any statement or group of statements can be specified in the ON RESTART statement. The following kinds of actions might be desirable:

- File and task variables can be reassigned the correct values.
- Job execution can be restarted at a particular point in the job, by using a GO statement.

If a halt/load occurs in the middle of a job, job execution restarts at the most recent successful job rollout. If during the halt/load process the JOBDESC file is removed, all jobs are also removed. Therefore, no jobs are restarted. For details, refer to the ??RJ (Remove JOBDESC File) primitive command in the *System Commands Reference*.

The ON RESTART statement is processed after a halt/load only if a successful job rollout occurred after the ON RESTART statement.

Some jobs include statements that interrogate a task variable to find out whether a task completed successfully or abnormally. However, the value of the task variable is not retained across a halt/load. An alternative method for storing information about whether a task completed successfully is to declare a Boolean variable in the job expressly for this purpose, and assign a value of TRUE or FALSE to the variable after the task completes. The value of the Boolean variable is saved across a halt/load, so it can always be interrogated later.

A useful technique for keeping track of exactly how far job execution had progressed before a halt/load occurred is to declare an integer variable and increment the value of that variable at several stages during the job. The ON RESTART statement can include statements that inspect the value of this integer variable and cause different actions to occur according to the value stored.

Dummy Files

Another method for storing information about job execution is to have the job create dummy files at specified points during the job execution, or whenever certain conditions are met. The job can then use the file residence inquiry to establish whether such files have been created, and make decisions based on their presence or absence.

Refer to [File Handling](#) for an example of a subroutine that can be used to create dummy files. The file residence inquiry is described under [Interrogating File Attributes](#).

For more detailed information about restarting WFL jobs after a halt/load, see the *Task Management Programming Guide*.

Section 3

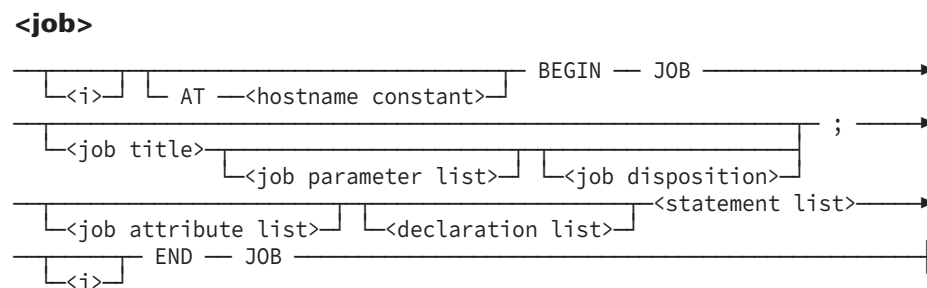
Job Structure

Overview

This section describes the overall structure of a WFL job, and discusses the job attributes and specifications that can be included at the beginning of a job.

Job Syntax

The following syntax represents a complete WFL job:



Job Structure

A job is composed of three parts:

- Job heading
- Job body
- End of job

Job Heading

The job header initializes the job by means of the following elements:

- <i> construct (optional)
- Hostname constant (optional)
- BEGIN JOB expression (required)
- Job title (optional)
- Job parameter list (optional)
- Job disposition (optional)
- Job attribute list (optional)

Job Body

The job body contains the operational part of the job and consists of

- Declaration list (optional)
- Statement list

End of Job

The end of job terminates the operation of the job and contains the following elements:

- <i> construct (optional)
- END JOB expression

Job Contents

The contents of WFL jobs are extremely flexible. The only two elements that appear in all WFL jobs are the BEGIN JOB and END JOB constructs. The WFL compiler adds these statements around single WFL statements that are entered through CANDE or at the ODT. The other elements are optional.

Job Format

The formatting of WFL jobs is also flexible. WFL constructs can begin in any column of a line, can continue over one line onto the next, and can include varying amounts of space between words. Also, multiple statements can appear on a line, as long as they are separated by semicolons (;). The only restriction is that the <i> construct must appear in the first column.

The <i> construct before the BEGIN JOB construct is optional and has no effect on the job. The <i> construct before the END JOB construct is optional normally, but is required if an AT <hostname constant> specification is included in the job. Refer to [Invalid and Valid Characters](#) for information about the <i> construct.

Comments can be entered into the job by preceding them with a percent sign (%). Any input following the percent character on a line is ignored by the WFL compiler. However, input that precedes the percent sign on a line is treated as part of the job. If the job is submitted from an ODT, or in array form from a user program, any input following the percent character up to and including the next semicolon (;) is ignored.

WFL control options can be included anywhere in the job, as long as they appear on a line with a dollar sign (\$) in the first or second column. These options are discussed in [Section 9, WFL Control Options](#).

AT Host Name

The AT <hostname constant> specification causes a job initiated on one system to be compiled and run on another system.

The WFL compiler at the receiving host system compiles the job; the sending host does not analyze the contents of the job, except for the END JOB construct. If the END JOB construct is missing, an “UNEXPECTED END OF FILE” error message is issued.

The host name specified must be an available host in a BNA or Open Systems Interconnection (OSI) network.

If the host specified is not available, the job compilation is aborted and the error message “UNKNOWN HOST SPECIFIED” is displayed. The CANDE ?HN command or the HOSTNAME (Host Name) system command can be used before the job is started to determine what BNA hosts are available. The CANDE ?AT local host NW OSIGATEWAY command or the NW OSIGATEWAY system command can be used to determine what OSI hosts are available.

A null character within a quoted string causes the transferred job to be incorrectly terminated, which causes erroneous syntax errors at the receiving host. This incorrect termination occurs because the sending host does not analyze the contents of the job; the sending host transfers data to the receiving host only until a null character is reached.

Using the START FOR SYNTAX statement is not allowed for jobs that include an AT <hostname constant> specification. Doing so generates the following syntax error:

```
START FOR SYNTAX IS ONLY VALID FOR LOCAL HOST
```

Note: A job that includes an AT <hostname constant> specification cannot have a job parameter list unless all the job parameters are optional and are not specified in the START statement.

Example

The following job will run at the system named SF6B:

```
?AT SF6B BEGIN JOB ADDIT;
.
.
.
?END JOB.
```

Job Title

<job title>

—<file title constant>—————|

Explanation

The job title specifies the name of the job. The job name appears in the:

- BOJ and EOJ messages displayed by the job
- Active entry display at the ODT
- Job summary generated by a job

- System log

The job title must be included if there is a job parameter list or job disposition in the job; otherwise, the job title is optional.

Though the syntax for a job title is the same as that for a file title constant, the job title does not have to be the name of any file. There is no connection between the job title of a job and the title of the file it is stored in. For the syntax of a file title constant, see [File Names, Titles, and Directories](#).

The job name can also be assigned by including the NAME attribute in the job attribute list for the job. Note that the NAME attribute overrides the job name in the <job title>.

If neither a job title nor a NAME job attribute is included in the job, the WFL compiler assigns a name to the job.

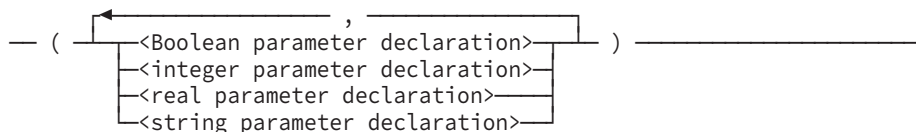
Examples

The following are examples of job headings that include valid job titles:

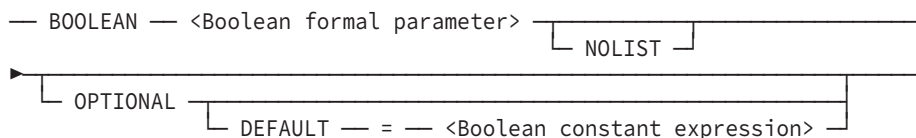
```
?BEGIN JOB RUNPROG;  
?BEGIN JOB DATA/SURVEYOR;  
?BEGIN JOB (WALLY)MENU/PLAN;  
?BEGIN JOB (LAO)OLD/DOC ON FORMSPK;
```

Job Parameter List

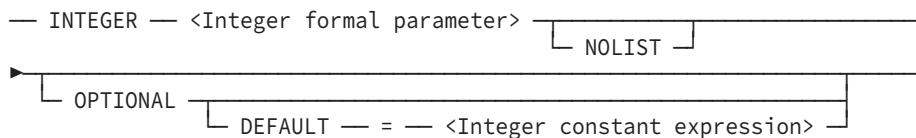
<job parameter list>

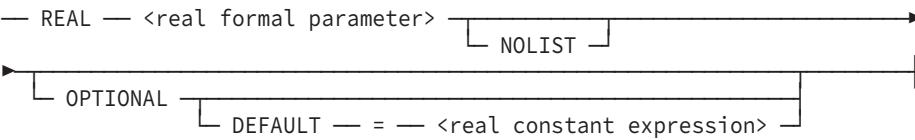


<Boolean parameter declaration>

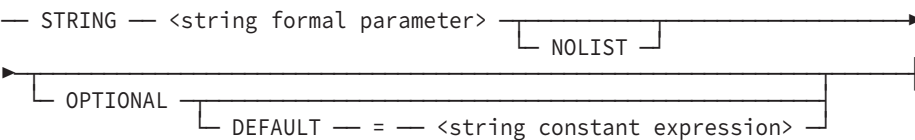


<integer parameter declaration>

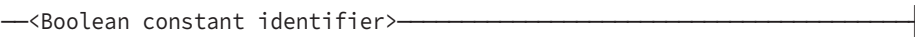


<real parameter declaration>

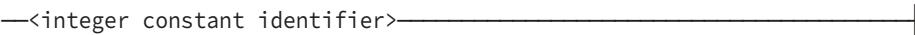
<string parameter declaration>



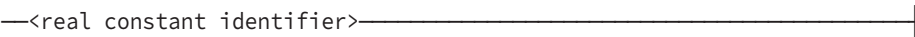
<Boolean formal parameter>



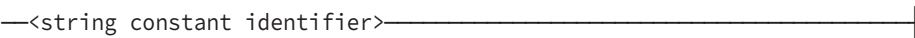
<integer formal parameter>



<real formal parameter>



<string formal parameter>



Explanation

A job parameter list declares constant identifiers and assigns them the values supplied in the START statement that initiated the job. The constant identifiers retain their original values throughout the job. They can be one of the following types:

- BOOLEAN
- INTEGER
- REAL
- STRING

These constant identifiers can be used in a WFL job in most of the places a constant or an expression of the appropriate type is permitted. (For example, a Boolean constant identifier can be used wherever a Boolean constant, Boolean constant expression or Boolean expression is permitted.)

The keyword NOLIST indicates that the parameter is not listed in the job summary, is suppressed from the list of parameters shown in response to an SQ (Show Queue) system command, and is suppressed from the analysis of a BOJ (Beginning of Job) log record.

The keyword **OPTIONAL** indicates that an actual parameter does not need to be passed for that job parameter. If an actual parameter is not passed, the default values are assigned as follows:

Type	Default Value
Boolean	FALSE
Integer	0
Real	0
String	"" (two double quotation marks)

Default values can also be specified with the **DEFAULT** clause following the keyword **OPTIONAL**. The specified default value is ignored if an actual parameter is provided.

A job parameter list can only be used in WFL jobs that are stored on disk and initiated through a **CANDE**, **MARC**, or **WFL START** statement. If a job initiated from another source includes a job parameter list, then it must be compiled for syntax only (see the description of the **SYNTAX** job disposition later in this section).

Note: *A job that includes an AT <hostname constant> specification cannot have a job parameter list unless all the job parameters are optional and are not specified in the START statement.*

When more than one job is stored in a disk file, none of the jobs can contain job parameter lists.

Example

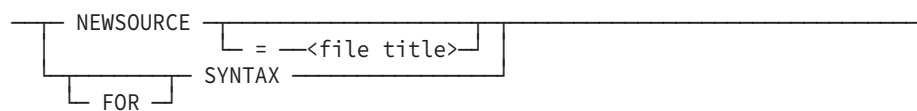
Consider the statement **START (WALLY)RUNPROG(23.8, 7)**, which initiates the following job:

```
?BEGIN JOB RUNPROG (REAL VAL1, REAL VAL2);  
  REAL INVAL;  
  INVAL := VAL2 * VAL2;  
  RUN OBJECT/WALLY/COUNTER(VAL1);  
  RUN OBJECT/WALLY/COUNTER(INVAL);  
?END JOB.
```

This job runs the program **OBJECT/WALLY/COUNTER** twice, first with a parameter value of 23.8, and then with a parameter value of 49 (because **INVAL** was assigned the square of the **VAL2** parameter).

Job Disposition

<job disposition>



Explanation

The NEWSOURCE job disposition saves a copy of the job as a JOBSYMBOL file. NEWSOURCE is ignored if the job is initiated through a START command. If NEWSOURCE is specified for a zip with array, a syntax error is generated. If the optional <file title> is omitted, the <job title> is used.

The SYNTAX job disposition compiles a job for syntax checking only. When the job is initiated, the WFL compiler simply compiles it and displays a list of any syntax errors in the job. This job disposition is beneficial when you only need to debug a job.

Example

The following is an example of a job heading that includes a NEWSOURCE job disposition with implicit naming of the file title. If the job is initiated through a ZIP WITH FILE or through CONTROLCARD using queue input, the job symbol JOB/TEST/NEWSOURCE is created:

```
?BEGIN JOB JOB/TEST/NEWSOURCE NEWSOURCE;
```

The following is an example of a job heading that includes a NEWSOURCE job disposition with explicit naming of the file title. If the job is initiated through a ZIP WITH FILE or through CONTROLCARD using queue input, the job symbol JOB/TEST/NEWSOURCE/1 is created:

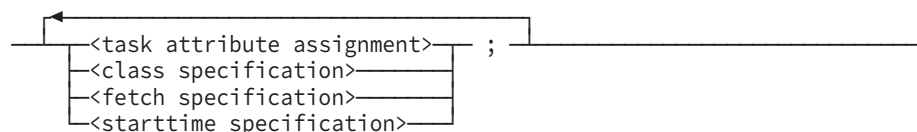
```
?BEGIN JOB TEST/NEWSOURCE NEWSOURCE = JOB/TEST/NEWSOURCE/1;
```

The following is an example of a job heading that includes a SYNTAX job disposition:

```
?BEGIN JOB GEO/ANLYS FOR SYNTAX;
```

Job Attribute List

<job attribute list>



Explanation

The job attribute list assigns task attributes and job specifications to the job. The CLASS specification, FETCH specification, and STARTTIME specification can only be applied to a job and are described under their own headings later in this section.

The syntax for task attribute assignment is described in [Section 5, Task Initiation](#). The types and the meanings of the task attributes recognized by the WFL compiler are described in the *Task Attributes Programming Reference Manual*.

Attributes not assigned to values in the job attribute list are assigned values by the system on the following basis:

- Certain attributes inherit values associated with the usercode the job is originated from, such as the CLASS, CHARGECODE, FAMILY, CONVENTION, PRIORITY, and USERCODE attributes. These values are stored in USERDATA locator nodes in the USERDATAFILE. For information about the USERDATAFILE, refer to the *Security SDK*.
- Other attributes can inherit values associated with the class (or queue) of the job. A class can have default values and/or maximum values specified for the PRIORITY, MAXPROCTIME, MAXIOTIME, or ELAPSEDLIMIT attributes. A job is discontinued if the job attribute list specifies greater resource limits than those associated with the class of the job.
- Attributes assume their default values if they are not included in the job attribute list and do not receive values from the usercode or class of the job. The attributes that limit resource usage default to their maximum possible values.

Task attributes can also be assigned to specific tasks initiated by a job. Refer to [Section 5, Task Initiation](#), for a description of how to assign task attributes to specific tasks.

Task attributes associated with a job can be changed or interrogated later in the job by using the MYJOB predeclared task variable.

Note: A string primary cannot be used in job-related task attribute assignments unless you use the MYJOB or MYSELF predeclared task variable syntax. Refer to [MYJOB and MYSELF Predeclared Task Variables](#) for details.

Resource-Limiting Attributes

Resource-limiting attributes affect the total resources available to a job and all its tasks when such attributes are included in the job attribute list. Resource-limiting attributes include the following:

- ELAPSEDLIMIT
- MAXIOTIME
- MAXLINES
- MAXPROCTIME
- MAXWAIT

- PRINTLIMIT

While these attributes can also be set for specific tasks, the accumulated resource usage of the job and all its tasks cannot exceed the values set for these attributes for the job.

Some of the job attribute values are inherited by the tasks initiated by a job. The inheritance status of each attribute is listed in the *Task Attributes Programming Reference Manual*.

The HOSTNAME task attribute has no effect when it is used in a job attribute list, but an AT <hostname constant> specification can be used instead. Refer to [AT Host Name](#) for details.

If a particular task attribute is assigned values more than once in a job attribute list, the last assignment overrides the previous ones.

The job attribute list is terminated by the appearance of any construct that is not a job attribute assignment.

USER is accepted as a synonym for the USERCODE task attribute when it is entered as part of a job attribute list. If the USER statement is the first statement of a job, it is interpreted as part of the job attribute list rather than as an executable statement. The specified USERCODE is validated during the compilation as a new usercode logging on for the duration of the job execution.

Note: When a usercode assignment occurs in the <job attribute list>, job attributes associated with the usercode in the USERDATAFILE are applied to the job and to the job compilation. A usercode assignment (either through a USER statement or a <task attribute assignment>) occurring in the <statement list> affects only the USERCODE and ACCESSCODE attributes. If the usercode of the job is changed, job messages stop appearing at the originating terminal even though the job continues to execute normally. Refer to [USER Statement](#) for details.

File equations cannot be included in the job attribute list. A statement beginning with FILE is interpreted as a file declaration and terminates the job attribute list.

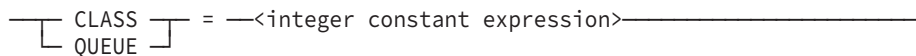
Example

The following example job includes a number of common job attribute assignments:

```
?BEGIN JOB COMPJOB;
  CLASS = 3;
  USERCODE = JOHN/JUAN;
  CHARGECODE = SHIPPING;
  MAXPROCTIME = 60;
  PRIORITY = 80;
  RUN (WALLY)POSTAGE/SUMMARY ON SHIPPK;
?END JOB.
```

CLASS Specification

<class specification>



(CLASS, QUEUE); "="; <integer constant expression> !

Explanation

The CLASS specification assigns the number of the queue desired for the job. (QUEUE is a synonym of CLASS.)

The CLASS, CLASSLIST, and ANYOTHERCLASSOK usercode attributes can be used to define the valid values for the class number.

The queue number specified by the usercode attribute CLASS is implicitly included in the list of queues allowed to be used by the user, regardless of the values of ANYOTHERCLASSOK and CLASSLIST attributes.

CLASSLIST is a list of queues that the user is allowed to use or is restricted from using. The value of ANYOTHERCLASSOK determines whether the list specified the valid queues or the invalid queues. If the attribute CLASS is nonzero, the user can use the specified queue regardless of the values of CLASSLIST and ANYOTHERCLASSOK.

For example, a user with the attributes shown below can specify any job class except 39 or 41, even though 40 appears in the CLASSLIST:

```
CLASS = 40
ANYOTHERCLASSOK
CLASSLIST = 39, 40, 41
```

For more information on managing WFL jobs in queues, refer to the *System Administration Guide*.

Examples

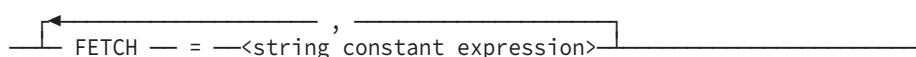
The following job extracts include CLASS specifications:

```
?BEGIN JOB COMP;
  CLASS = 5;

?BEGIN JOB CLASSVAL( INTEGER CL );
  CLASS = CL;
```

FETCH Specification

<fetch specification>



Explanation

The FETCH specification provides operators with information about the job. A maximum of 255 characters can be specified in the string constant expression.

One or more strings can be included in the FETCH specification. Initiation of a job is suspended if it includes a FETCH specification; the job appears in the waiting entries list at the ODT with a "REQUIRES FETCH" message. The FETCH messages can be displayed using the PF (Print Fetch) system command. The job can be reactivated using the CANDE OK command or the OK system command.

If the system option 19 (NOFETCH) is set, the FETCH attribute does not suspend job initiation. However, it is still possible to use the PF system command to display the fetch message associated with a job. System options are set or displayed using the OP (Options) system command.

Example

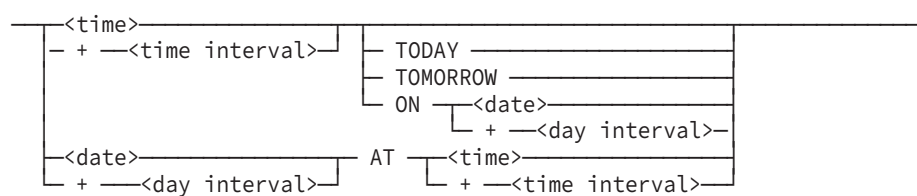
```
?BEGIN JOB UPDATE;
  FETCH = "THIS JOB NEEDS THREE TAPE DRIVES";
  RUN NIGHTLY/UPDATE;
?END JOB.
```

STARTTIME Specification

<starttime specification>

— STARTTIME — = —<starttime spec>—

<starttime spec>

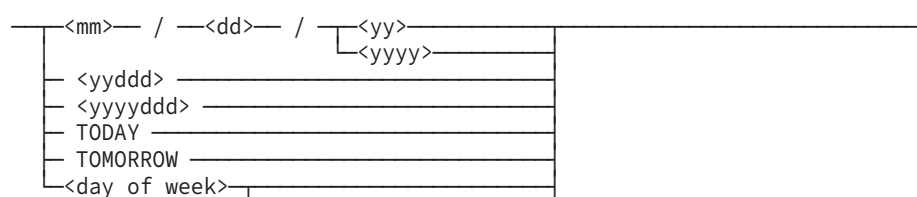


<time>

<time interval>

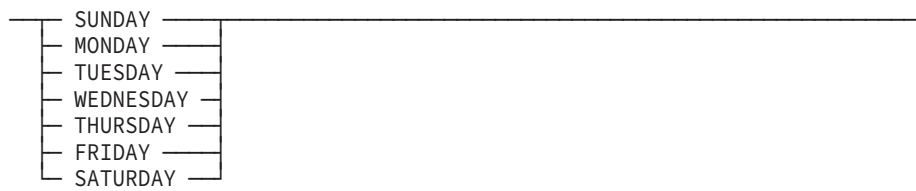
—/2\—<digit> : —/2*\—<digit>—

<date>



└─<month, day and year>─┘

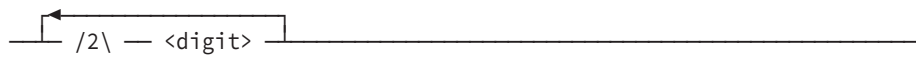
<day of week>



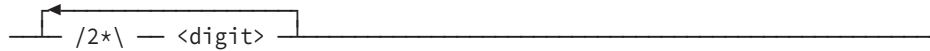
<day interval>

<mm>

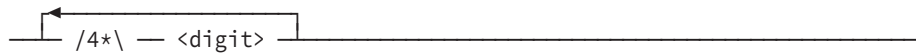
<dd>



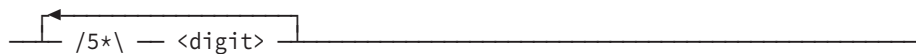
<yy>



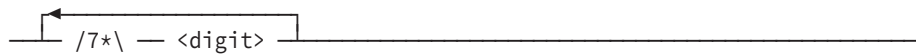
<yyyy>



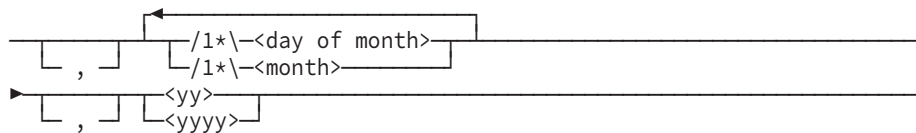
<yyddd>



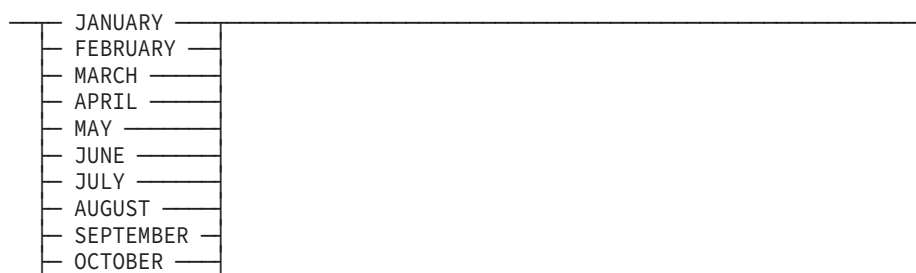
<yyyyddd>



<month, day and year>



<month>





Explanation

The STARTTIME specification delays job initiation until the specified start time.

The time construct specifies the time of day on a 24-hour clock. The time and time interval constructs are of the form HH: MM, where HH specifies the hour (or number of hours) and MM specifies the minute (or number of minutes). HH must be less than 24, and MM must be less than 60. The minute values must be 2-digit numbers. If a time interval is specified, that time interval is added to the current time.

The day, hour, and month values can either be a 1-digit or a 2-digit number.

The date construct can be input either in the Gregorian format: mm/dd/yy or mm/dd/yyyy, or the Julian format: yyddd or yyyyddd. If a 5-digit 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. If a 7-digit Julian date is used, the first four digits signify the year (including the century) and the last three digits signify the day of the year. For example, if you input either 19293 or 2019293, they will both equal day 293 of 2019.

Notes:

- *A two-digit year is interpreted as a year between 65 years ago and 34 years in the future. However, a year prior to 1970 is invalid.*
- *For the STARTTIME specification, years prior to 2000 are invalid.*

If a day interval is specified, that number of days is added to the current date.

If the time is specified without a date or day interval, the current date is used.

If TODAY is specified, the current date is used.

If TOMORROW is specified, one day is added to the current date.

If a day of the week is specified, then enough days are added to the current date to start the job on the next day of the week that was specified.

If a day of the week and a month, day, and year are specified then the system verifies that the specified date falls on the specified day of week and that date is used.

The FS (Force Schedule) system command can be used to cause job initiation to proceed immediately, without waiting for the specified start time. For a description of the FS command, refer to the *System Commands Reference*.

Examples

The following job begins execution after 10:00 p.m. on March 20, 1990:

```
?BEGIN JOB EXAMPLE1;  
    STARTTIME = 22:00 ON 03/20/90;  
    .  
    .  
    .  
?END JOB.
```

The following job begins execution a minimum of 1 hour and 30 minutes after entering the system:

```
?BEGIN JOB EXAMPLE2;  
    STARTTIME = +1:30;  
    .  
    .  
    .  
?END JOB.
```

The following job is executed on host BLUE and begins execution after 11:00 a.m. according to the system clock on host BLUE:

```
?AT BLUE BEGIN JOB;  
    STARTTIME = 11:00;  
    .  
    .  
    .  
?END JOB.
```

If the current day is Saturday, then the following job begins after 1:00 a.m. 7 days later than the current date. Otherwise, the job begins after 1:00 a.m. on the next date that falls on a Saturday.

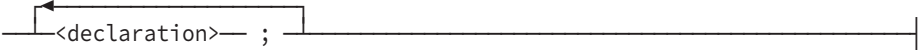
```
?BEGIN JOB EXAMPLE1;  
    STARTTIME = 1:00 ON SATURDAY;  
    .  
    .  
    .  
?END JOB.
```

The following job begins execution after 2:00 p.m. on November 24, 2008.

```
?BEGIN JOB EXAMPLE1;  
    STARTTIME = 14:00 ON MONDAY, NOVEMBER 24, 2008;  
    .  
    .  
    .  
?END JOB.
```

Declaration List

<declaration list>


—<declaration>— ; —

Explanation

Declarations define constants, variables, subroutines, and global data specifications.

Declarations of variables can also include initial value assignments. The following types of variables are available in WFL: BOOLEAN, INTEGER, REAL, STRING, FILE, and TASK. For details about the declarations available in WFL, refer to [Section 4, Declarations](#).

All declarations for the job must precede all statements in the job, and the declarations in a subroutine must precede all the statements in that subroutine.

More than one declaration can appear on a line, as long as each declaration is followed by a semicolon (;).

Example

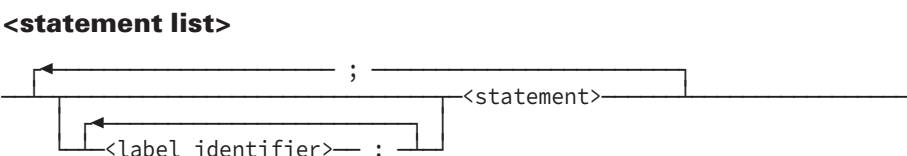
The following job segment illustrates the use of declarations:

```

CONSTANT DEBUG = FALSE;
BOOLEAN TFVAL,
        SUBTVALU;
FILE FILE1(TITLE=(ODCOM)FIXIT ON PACK);
INTEGER INTVAL := 43;
REAL RLVAL := 19.67,
      RLVAL2 := 16.8;
STRING STRNGVL := "Test Run";
TASK DOUBAT (PRIORITY=75,INFILE(TITLE=(WALLY)DATA/SAVER));
DATA INPUT/PREP
    43
    69
    128
? % End of data
SUBROUTINE PROVIT;
BEGIN
    RUN (RAJA)DATA/PREP(TFVAL,INTVAL,STRNGVL,RLVAL);
    FILE INPUT (TITLE=INPUT/PREP,KIND=READER);
END PROVIT;

```

Statement List



Explanation

The statements in a WFL job are grouped together in a statement list. More than one statement can appear on a line, while a lengthy statement can continue across several lines. Individual statements are distinguished by a statement separator.

The usual statement separator is a semicolon (;) appearing at the end of a statement. However, an invalid character at the start of a line also acts as a statement separator.

Each statement can be preceded by one or more label identifiers. These label identifiers can then be referred to by GO statements in the job. GO statements transfer control to the point in the job where the specified label identifier appears.

Refer to [Section 6, Statements](#), for descriptions of all the statements available in WFL.

Examples

In this first example, the semicolon signals the end of a statement.

```
BEGIN JOB SEPARATOR/EXAMPLE1;
IF DECIMAL( TIMEDATE(HHMMSS) ) < 120000 THEN
    DISPLAY "Good Morning, it is now: "
        & TIMEDATE(DISPLAY)
ELSE
    DISPLAY "Good Afternoon, it is now: "
        & TIMEDATE(DISPLAY);
WAIT(5); DISPLAY "Bye";
?END JOB.
```

In this next example, the invalid character—the question mark (?)—in the first column of a statement implies the end of the previous statement. A semicolon must be used to separate multiple statements appearing on a single line.

```
?BEGIN JOB SEPARATOR/EXAMPLE2
?IF DECIMAL( TIMEDATE(HHMMSS) ) < 120000 THEN
    DISPLAY "Good Morning, it is now: "
        & TIMEDATE(DISPLAY)
ELSE
    DISPLAY "Good Afternoon, it is now: "
        & TIMEDATE(DISPLAY)
?WAIT (5); DISPLAY "Bye"
?END JOB.
```

The following example shows the use of statement label identifiers:

```
?BEGIN JOB LABEL/EXAMPLE3;
STRING S;
RETRY:
    COPY X AS Y;
    IF FILE Y ISNT RESIDENT THEN
        BEGIN
            AX:
                S:= ACCEPT("FILE Y NOT COPIED.  AX: RETRY, OR AX: SKIP");
                IF S = "RETRY" THEN
                    GO TO RETRY
                ELSE IF S = "SKIP" THEN
                    ABORT
                ELSE
                    GO TO AX;          % Ask again.
            END;
        ?END JOB.
```

WFL Job Example

The following example illustrates a complete WFL job:

```
?BEGIN JOB EXAMPLE      (STRING TESTNAME,INTEGER TESTNUMBER);
    NAME=EXAMPLE/#STRING(TESTNUMBER,*);    % Job attribute
    USERCODE = WFL/MANUAL;                  % Job attribute
    CLASS = 2;                              % Job attribute
    TASK TCOMP, TRUN;                        % Variable declaration
    COMPILE #(TESTNAME & STRING(TESTNUMBER,*))[TCOMP] WITH ALGOL[TRUN] GO;
    COMPILER DATA CARD                      % The COMPILE statement is
        BEGIN                               % followed by a local data
            INTEGER I;                      % specification containing the
            DISPLAY ("P IS RUNNING");       % program being compiled.
            DISPLAY ("NOW ABORT");          % For information about local
            I=I/0;                           % data specifications, see the
        END.                               % "Task Initiation" section.
?                                           % End of data.
IF TCOMP IS COMPILEDOK THEN                % Displays message if compile
    DISPLAY "COMPILED OK"                   % is successful.
ELSE ABORT "*DID NOT COMPILE";              % Aborts job if compile
                                           % fails.
IF TRUN IS COMPLETEDOK THEN                % Displays messages if run
    DISPLAY "RAN OK"                       % is successful; otherwise,
ELSE ABORT "*** RUN ABORTED";              % aborts job.
?END JOB.
```


Section 4

Declarations

Overview

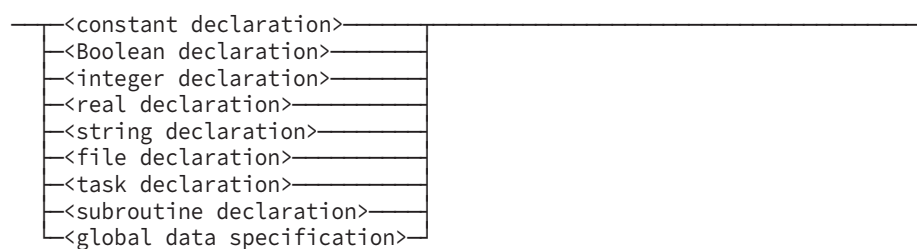
Declarations define constants, variables, subroutines, and global data specifications.

Statement label identifiers are not declared. Refer to [Statement List](#) for a discussion of label identifiers.

Declaration Syntax

The following syntax represents the declarations available in a WFL job:

<declaration>



Explanation

All declarations in a job must follow the job attribute list and must precede any statements. All declarations in a subroutine must follow the BEGIN statement for the subroutine and must precede any executable statements in the subroutine.

Declarations can occur in any order.

Scope of Declarations

Declarations can occur either at the job level or within subroutines. Declarations occurring at the job level can define:

- Global variables
- Subroutines
- Global data specifications

Declarations

Items declared at the job level can be referenced anywhere in the job, including in any of the subroutines.

Variables and subroutines declared within a subroutine are local to that subroutine. They can only be used in the subroutine they are declared in, and in subroutines nested within that subroutine. Local variables are reinitialized each time the subroutine is invoked.

Note: A variable or subroutine cannot be referenced before its declaration. For example, a subroutine cannot make use of a globally declared variable unless the declaration of that variable occurs before the declaration of the subroutine.

Special care should be taken when assigning values to a global variable in an asynchronous subroutine. Refer to [PROCESS Statement](#) for further information.

Variable Initialization


The initial value of a variable can be specified in the declaration; if it is not, the default value for that variable type is applied to the variable. The initial value of a variable is stored before the execution of the first statement in the job or subroutine.

Boolean, integer, real, and string variables retain their values across a halt/load. However, the contents of file and task variables are not saved. Refer to [Job Restart After a Halt/Load](#) for details.

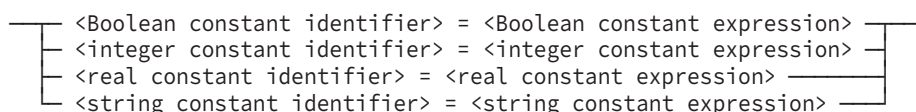
The INITIALIZE statement assigns a value of NEVERUSED to the STATUS task attribute of the specified task variable, and restores the default values to all other task attributes and file equations associated with that task variable.

Constant Identifiers

<constant declaration>

— CONSTANT — 

<constant declaration element>



Explanation

A constant declaration declares one or more constant identifiers and assigns their value. A constant identifier can be one of the following type:

- Boolean
- Integer
- Real

- The constant declaration enables constant values to be referenced by name rather than by specifying the actual values throughout the job.

The values of constant identifiers are retained across a halt/load.

The following example represents the use of constant identifiers:

Boolean Variables

```

graph LR
    subgraph BooleanRule [BOOLEAN]
        direction LR
        B1[<Boolean identifier>] --- B2[ , <Boolean constant expression> ]
    end

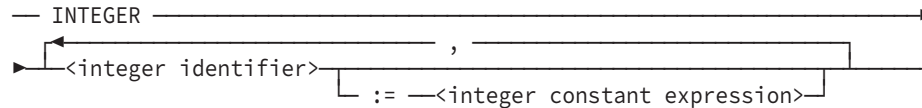
```

A Boolean declaration declares a variable of type `BOOLEAN`. The default initial value of a Boolean variable is `FALSE`.

The values of Boolean variables are saved across a halt/load and are restored when the job restarts.

Integer Variables

<integer declaration>



Explanation

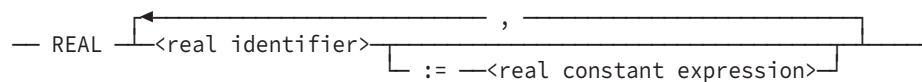
An integer declaration declares a variable of type INTEGER. The default initial value of an integer variable is 0.

If the ***:= integer constant expression*** clause is specified for an integer identifier, this integer constant expression is used as the initial value for that integer identifier.

The values of integer variables are saved across a halt/load and are restored when the job restarts.

Real Variables

<real declaration>



Explanation

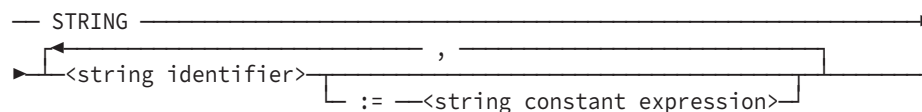
A real declaration declares a variable of type REAL. The default initial value of a real variable is 0.

If the clause ***:= real constant expression*** is specified for a real identifier, this real constant expression is used as the initial value for that real identifier.

The values of real variables are saved across a halt/load and are restored when the job restarts.

String Variables

<string declaration>



Explanation

A string declaration declares a variable of type STRING. The default initial value of a string variable is a null string ("").

The values of a string variable are saved across a halt/load.

If the **`:= string constant expression`** clause is specified for a string identifier, this string constant expression is used as the initial value for that string identifier.

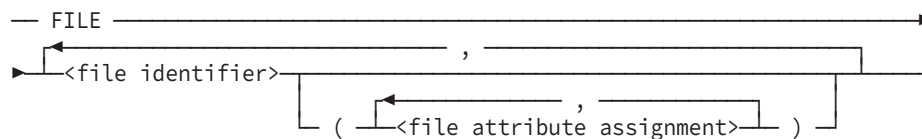
Example

The following example illustrates the declaration of string variables:

```
STRING STR1, STR2;
STRING STR3 := "STRVAL";
```

File Variables

<file declaration>



Explanation

A file declaration defines a logical file with the specified file attributes. A file declared in a WFL job can be used in either of two ways:

- To enable file attribute inquiries.

If the declared file is associated with a physical file, the file identifier can be used in expressions that return the values of attributes of the physical file. This subject is discussed in [Interrogating File Attributes](#).

- To enable a task to use a declared file.

If a file is declared in a global file assignment, the global file assignment causes a task to use the declared file in place of a file that the task would normally use. This subject is discussed in [File Equations](#).

WFL cannot directly read from or write to the declared file. However, the file equation capabilities of WFL do provide considerable control over the choice of files that will be used by a task. Refer to [File Equations](#) for further details.

Note: *File declarations cannot occur in a subroutine.*

The WFL job can also include input data for a task, in the form of data specifications. Refer to [Global Data Specifications](#) and [Local Data Specifications](#) for further details.

Declarations

File attributes included in a file declaration must be assigned constants or constant expressions. Attributes of type name, file name, or title must be assigned constants only.

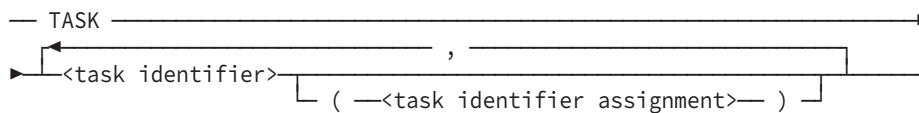
File attribute values associated with a file variable are not saved across a halt/load. Refer to [Job Restart After a Halt/Load](#) for further details.

Note: The use of the file identifier SUMMARY should be avoided for printer files. If a printer file is declared in WFL with a file identifier of SUMMARY, and a job summary is generated with the default title of SUMMARY, the printer file declared in WFL will be lost. This occurs because the job summary file will overwrite the file declared in WFL.

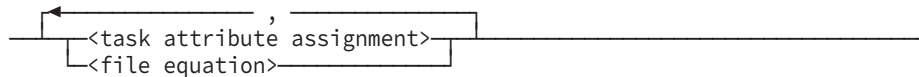
To create a printer file in WFL with the file identifier SUMMARY, either use the JOBSUMMARY task attribute to suppress the job summary, or use the JOBSUMMARYTITLE task attribute to assign a different name to the job summary file.

Task Variables

<task declaration>



<task identifier assignment>



Explanation

The task declaration declares task variables, which can be associated with particular tasks in a task initiation statement.

If a task initiation statement includes a task variable, then any task attribute assignments or file equations that have been specified for that task variable are applied to the task.

The values assigned to file and task attributes in a task declaration must be constants or constant expressions. Attributes of type name, file name, or title must be assigned constants only.

Any task attribute values or file equations associated with a task variable are not saved across a halt/load. Refer to [Job Restart After a Halt/Load](#) for further details.

Task variables can be assigned task attribute values or file equations later in the job by the task assignment statement. Refer to the assignment statement in [Section 6, Statements](#), for a description of the task assignment statement.

The task variable can also be used for inquiries about the values of any task attributes associated with the task, including those task attributes that record task status and task history. This is possible by using various WFL expressions, which are described in

[Section 7, Expressions.](#)

WFL provides two predefined task variables. The following task variables can be used to interrogate the values of attributes in the job:

- MYSELF
- MYJOB

Refer to [Using Task Variables](#) for a discussion of the uses of the task variables, including the MYSELF and MYJOB task variables.

Subroutines

<subroutine declaration>

```

— SUBROUTINE —<subroutine identifier>———→
      |
      |——<subroutine parameters>———|
      |
      |——<statement>———|
      |——<subroutine block>———|

```

<subroutine parameters>

```

— ( ———— , ———— ) ————|
      |
      |—— BOOLEAN —<Boolean identifier>———|
      |      |
      |      |—— VALUE ———|
      |
      |—— INTEGER —<integer identifier>———|
      |      |
      |      |—— VALUE ———|
      |
      |—— REAL —<real identifier>———|
      |      |
      |      |—— VALUE ———|
      |
      |—— STRING —<string identifier>———|
      |      |
      |      |—— VALUE ———|
      |
      |—— FILE —<file identifier>———|
      |
      |—— TASK —<task identifier>———|

```

<subroutine block>

```

— BEGIN ————<declaration list>———<statement list>——— END ————→
      |
      |——<subroutine identifier>———|

```

Explanation

A subroutine declaration identifies a series of statements that are executed when the subroutine is invoked by a subroutine invocation statement. The subroutine invocation statement is described in [Section 6, Statements.](#)

The simplest form of a subroutine consists of the subroutine heading, followed by a single statement. Declarations and multiple statements can be included in the subroutine, but they must be bracketed by the words BEGIN and END.

Any kind of WFL declaration can be included in a subroutine except for file declarations and global data specifications. However, subroutines can include local data specifications, and references to global data specifications and files that are declared globally. Variables

Declarations

declared in a subroutine are local to that subroutine (refer to [Scope of Declarations](#)).

The WFL statements are described in [Section 6, Statements](#).

WFL enables a subroutine identifier to be repeated after the end of the subroutine. This feature helps create WFL jobs that are easier to read, especially in cases where nested subroutines occur. If a subroutine identifier is specified after END, it must be the same subroutine identifier specified at the start of the declaration.

A subroutine declaration can include another subroutine declaration within itself. This is referred to as nesting. Subroutines can be nested to a maximum of 10 levels.

Notes:

- *WFL jobs that have nested subroutines can result in larger code files due to the new Long Name Call/LNMC, Long Value Call/LVLC code, which replaces Name Call/NAMC, Value Call/VALC and uses more bytes.*
- *If the WFL compiler issues a warning message that a code segment exceeds its capacity, then you must split the statements in that subroutine into multiple subroutines. Ignoring the warning message can result in the initiation or compilation of a WFL job being aborted.*
- *If the WFL compiler issues a warning that a code segment exceeds its capacity and that code segment is the outer block of the WFL job, then you must move code from the outer block into multiple subroutines. Ignoring the warning message can result in the initiation or compilation of a WFL job being aborted.*

Subroutine Parameters

The subroutine parameters specification can be included in the subroutine heading to declare parameters for the procedure. These parameters are assigned values taken from the statement that invoked the subroutine. The parameters are treated as local variables within the subroutine; they can be interrogated or assigned new values anywhere within the subroutine or within any subroutines nested within it.

Declarations in a subroutine cannot declare an identifier that has the same name as any of the parameters of that routine. Each parameter in the parameter list must be preceded by a keyword specifying the parameter type.

A parameter is considered to be call-by-reference unless it is followed by the word VALUE, which indicates that the parameter is call-by-value. Task and file variables can only be call-by-reference.

Note: *Any changes made to the value of a call-by-reference parameter in the course of the subroutine also change the value of the variable that was passed to that parameter. By contrast, any changes made to the value of a call-by-value parameter in the course of the subroutine do not affect the value of the variable that was passed to that parameter.*

If a constant or an expression is passed as a parameter in the subroutine invocation statement, that parameter is passed by value whether or not VALUE is specified for that parameter. When a real parameter is passed to an integer in a subroutine call, it requires integer truncation and treats the real parameter as an expression. Because an expression is passed as a parameter, that parameter is passed by value.

When a parameter is a string expression, the string expression can contain up to 1800 characters.

Example

The following example illustrates the use of subroutines:

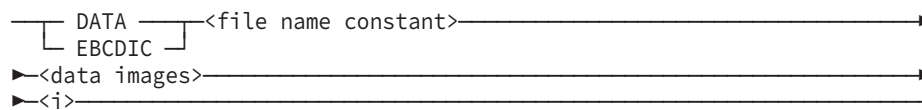
```
?BEGIN JOB SUBSHOW;
  INTEGER VAL1 := 7,          % Global variable declarations
  VAL2 := 11;
  % Begin subroutine declaration
  SUBROUTINE FIRSTSUB(INTEGER PARAM1, INTEGER PARAM2 VALUE);
  BEGIN
    PARAM1 := PARAM1 * 2;      % Changes made to the values of the
    PARAM2 := PARAM2 * 2;      % parameters
  END FIRSTSUB;
  % End subroutine declaration
  FIRSTSUB(VAL1,VAL2);        % Subroutine invocation
  RUN (RAJA)AREA/MEASURE(VAL1,VAL2); % Task initiation
?END JOB.
```

In this example, the global variables VAL1 and VAL2 are assigned values of 7 and 11, respectively. The subroutine invocation statement passes the variables VAL1 and VAL2 as values for the parameters PARAM1 and PARAM2. PARAM1 is therefore assigned a value of 7, and PARAM2 is assigned a value of 11. The subroutine contains statements that double the values of these parameters to 14 and 22, respectively.

The RUN statement that follows the subroutine invocation statement passes VAL1 and VAL2 to the program (RAJA)AREA/MEASURE. The values the program receives are 14 and 11. This occurs because the change made to the value of the call-by-reference parameter PARAM1 also affects the global variable VAL1 that was passed to it. The change made to the call-by-value variable PARAM2 does not affect VAL2.

Global Data Specifications

<global data specification>



<data images>

One or more card images or records of EBCDIC data, none of which can contain a question mark (?) in the first column.

Explanation

A global data specification contains data that can be used as input by programs initiated in the job. The global data specification is read as if it were a card reader input file, and can be closed and reopened or read by more than one task.

The file name constant identifies the global data specification. The file name constant cannot contain a usercode or an asterisk (*).

The data images must be of the type specified at the start of the global data specification. The types of global data specifications are defined as follows:

Data Type	Meaning
DATA	Allows any characters in the EBCDIC character set.
EBCDIC	A synonym for DATA.

The data images should begin on the line or card image following the global data specification heading.

Note: Any data following the heading on the same line is ignored.

Tasks are read from the global data specification as if they were an input file, and each line of data images is treated as a new record of the input file. A record in a disk file created by CANDE with a FILEKIND=JOBSYMBOL contains data in columns 1 through 80, spaces in columns 81 and 82, and the sequence number in columns 83 through 90. Since the default MAXRECSIZE that a program uses when reading these data images is 14 words (84 characters), care must be taken to ensure that unwanted characters are not included in the data.

Note: To ensure that no unwanted information is read from columns 81 through 84 of the data images, equate UNITS to CHARACTERS and MAXRECSIZE to 80.

The <i> construct that terminates the global data specification also separates the data specification from the next declaration or statement; it is not necessary to follow the data specification with a semicolon (;).

For a global data specification to be used by a task, the task initiation statement should contain a file equation equating the TITLE file attribute to the file name constant specified in the global data specification. The file equation should also equate the KIND attribute of the file to READER.

When more than one task uses the same global data specification as input, each task begins reading at the start of the deck. WFL also enables you to use local data specifications, which provide input for a single task only. Refer to [Local Data Specifications](#) for a further description of this capability.

A global data specification can appear only in jobs stored in disk files and initiated by a START statement.

Note: Global data specifications can occur only in the outer block of a job; that is, they cannot be declared in subroutines.

Example

The following job uses a global data specification:

```
?BEGIN JOB SURF/METER;
DATA WAVE/HEIGHTS          % Begin global data specification
  3.5
  1.2
  23.4
?                             % End global data specification
  RUN (WALLY)SURF/ANALYZE;
    FILE WAVEIN(TITLE=WAVE/HEIGHTS,KIND=READER);
  RUN (WALLY)SURF/REPORT;
    FILE WAVES(TITLE=WAVE/HEIGHTS,KIND=READER);
?END JOB.
```

Declarations

Section 5

Task Initiation

Overview

A task is a process that runs in its own stack. This section discusses the various task initiation statements, task attribute assignments, file equations, task variables, and the use of local data specifications to provide input to a task.

Task Initiation Statements

Explanation

The following table lists all the WFL statements that initiate tasks. For more information about a particular statement, refer to [Section 6, Statements](#).

Statement	Effect
ADD	Copies files that are not already resident on the destination.
ARCHIVE	Archives files according to the archive task selected. Include one of the following words to complete the archive statement, as in ARCHIVE FULL: <ul style="list-style-type: none">• DIFFERENTIAL• FULL• INCREMENTAL• MERGE• RESTORE• RESTOREADD• ROLLOUT
BIND	Combines object code files.
COMPILE	Compiles a program.
COPY	Copies files.

Task Initiation

Statement	Effect
LOG	Runs the LOGANALYZER utility, which reports on selected records in the system log.
PB	Runs the SYSTEM/BACKUP utility.
RUN	Executes a program.
START	Initiates a job stored in a disk file.

Each task initiation statement normally executes synchronously; that is, the job waits for completion of the task before continuing to the next statement in the job. Certain tasks can be made to run asynchronously by preceding the task initiation statement with the word `PROCESS`. An asynchronous task executes concurrently with the job. Refer to [PROCESS Statement](#) for a complete list of statements that can be processed asynchronously.

A subroutine invocation statement normally does not initiate a task. However, an asynchronous task can be initiated by a subroutine invocation statement that occurs within a `PROCESS` statement.

The `START` statement differs from the other task initiation statements in an important way. The `START` statement both compiles and executes a job. The compile is executed synchronously with the job that contains the `START` statement, but the execution occurs asynchronously and is not associated with the original job. A `PROCESS START` also causes the compilation to occur asynchronously from the originating job.

Example

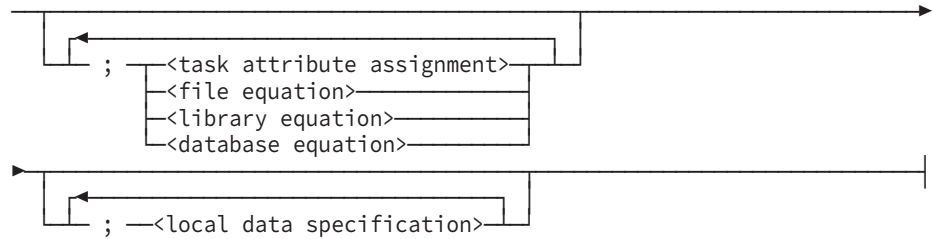
The following example illustrates the difference between synchronous and asynchronous task initiation:

```
COMPILE PROG/F1 WITH COBOL85 LIBRARY;  
  COBOL85 FILE CARD = PROG/SOURCE ON DISK;  
PROCESS COPY & COMPARE PROG/F1 TO PROGTAPE;  
RUN PROG/F1;
```

The statements in this example compile a program, and then copy it onto tape while the program is executing.

The `COMPILE` statement initiates a synchronous task of compiling the program. When the compilation is finished, the `COPY` statement starts an asynchronous process of copying the new code file to tape. As soon as the copy process starts, the `RUN` statement initiates the synchronous process of executing the new program. The second line of the `COMPILE` statement contains a file equation; however each of these task initiation statements can be followed by a task variable and by task equations.

The remainder of this section is devoted to explaining the common features that enable you to control the manner in which tasks are executed.



[Table 5–1](#) lists the task attributes that are available through WFL, with the task attributes grouped together into several functional categories. Task attributes with a type of task or event are not available through WFL, and thus do not appear in the table.

More detailed information on task attributes and their use, including descriptions of all the task attributes, can be found in the *Task Attributes Programming Reference Manual*.

Table 5–1. Task Attribute Groupings

Task	Task Attribute	WFL Type
Billing	CHARGE USERCODE	<name task attribute> <usercode assignment>
Host Services Tasking	HOSTNAME ITINERARY <i>Note: The form of the <name> construct that allows 17 EBCDIC characters other than quotation marks (") is not supported.</i>	<name task attribute> <string task attribute>
Data Comm	AUTOSWITCHTOMARC DESTSTATION DISPLAYONLYTOMCS LANGUAGE ORGUNIT SOURCEKIND SOURCESTATION STATION STATIONNAME TANKING	<Boolean task attribute> <integer task attribute> <Boolean task attribute> <name task attribute> <integer task attribute> <real task attribute> <real task attribute> <integer task attribute> <string task attribute> <mnemonic task attribute>
Debugging	OPTION TADS TASKFILE	<option assignment> <Boolean task attribute> <file name task attribute>

Table 5–1. Task Attribute Groupings (cont.)

Task	Task Attribute	WFL Type
Files	CURRENTDIRECTORY DATABASE DATAPATH EXECUTEPATH FAMILY FILE FILEACCESSRULE OPTION Notes: <ul style="list-style-type: none"> <i>FILECARDS is an acceptable synonym for FILE.</i> <i>The AUTORM option is the only option of this attribute that affects disk files.</i> 	<string task attribute> <database equation> <datapath assignment> <executepath assignment> <family assignment> <file equation> <mnemonic task attribute> <option assignment>
Identification	JOBNUMBER MIXNUMBER NAME MPID WORKLOADGROUP	<integer task attribute> <integer task attribute> <title task attribute> <string task attribute> <string task attribute>
Interprocess Communication	AX LOCKED PARTNEREXISTS STATUS SW1 through SW8 TARGET TASKLIMIT TASKSTRING TASKVALUE TYPE	<string task attribute> <Boolean task attribute> <Boolean task attribute> <mnemonic task attribute> <Boolean task attribute> integer task attribute> <integer task attribute> <string task attribute> <real task attribute> <mnemonic task attribute>

Table 5-1. Task Attribute Groupings (cont.)

Task	Task Attribute	WFL Type
Job Summaries	JOBSUMMARY JOBSUMMARYTITLE NOJOBSUMMARYIO OPTION Note: The NOSUMMARY option is the only option of this attribute that affects job summaries.	<mnemonic task attribute> <title task attribute> <Boolean task attribute> <option assignment>
Libraries	LIBRARY LIBRARYSTATE LIBRARYUSERS	<library equation> <real task attribute> <integer task attribute>
Memory Management	CORE STACKLIMIT STACKSIZE Note: STACK is an acceptable synonym for STACKSIZE.	<core assignment> <integer task attribute> <integer task attribute>
Print Output	BACKUPFAMILY BDNAME DESTSTATION OPTION PRINTDEFAULTS TASKFILE Note: The BACKUP, BDBASE, TODISK, and TOPRINTER options are the only options that affect printer output.	<name task attribute> <file name task attribute> <integer task attribute> <option assignment> <printdefaults assignment> <file name task attribute>

Table 5–1. Task Attribute Groupings (cont.)

Task	Task Attribute	WFL Type
Resource Usage Data	ACCUMIOTIME ACCUMPROCTIME ELAPSEDTIME INITPBITCOUNT INITPBITTIME OTHERPBITCOUNT OTHERPBITTIME TEMPFILEBYTES	<real task attribute> <real task attribute> <real task attribute> <real task attribute> <real task attribute> <real task attribute> <real task attribute> <real task attribute>
Resource Usage Limits	ELAPSEDLIMIT MAXIOTIME MAXLINES MAXPROCTIME MAXWAIT PRIORITY RESOURCE SAVEMEMORYLIMIT STACKLIMIT TASKLIMIT TEMPFILELIMIT TOTALMEMORYLIMIT WAITLIMIT	<real task attribute> <real task attribute> <integer task attribute> <real task attribute> <integer task attribute> <integer task attribute> <resource assignment> <real task attribute> <integer task attribute> <integer task attribute> <real task attribute> <real task attribute> <real task attribute>
Restarting Tasks	BRCLASS CHECKPOINTABLE RESTART RESTARTED	<mnemonic task attribute> <Boolean task attribute> <integer task attribute> <Boolean task attribute>
Security	ACCESSCODE FILEACCESSRULE USERCODE DECKGROUPNO	<accesscode assignment> <mnemonic task attribute> <usercode assignment> <integer task attribute>

Table 5–1. Task Attribute Groupings (cont.)

Task	Task Attribute	WFL Type
Task History	ERROR	<mnemonic task attribute>
	HISTORY	<real task attribute>
	HISTORYCAUSE	<mnemonic task attribute>
	HISTORYTYPE	<mnemonic task attribute>
	HSPARAMSIZE	<integer task attribute>
	OPTION	<option assignment>
	STACKHISTORY	<string task attribute>
	STATUS	<mnemonic task attribute>
	STOPPOINT	<real task attribute>
	SUPPRESSWARNING	<suppresswarning assignment>
	TASKWARNINGS	<taskwarnings assignment>

Task Attribute Assignment

<task attribute assignment>

<Boolean task attribute>	=	<Boolean expression>
<file name task attribute>	=	<file name>
<integer name task attribute>	=	<integer expression>
<mnemonic task attribute>	=	<task mnemonic primary>
<name task attribute>	=	<name>
<real task attribute>	=	<real expression>
<string task attribute>	=	<string expression>
<title task attribute>	=	<file title>
<complex task attribute assignment>		

Explanation

Task attributes are used to monitor and control the execution of tasks. For details on the meanings and uses of task attributes, refer to the *Task Attributes Programming Reference Manual*. Task attribute assignments, in general, follow this format:

```
<attribute name> = <attribute value>
```

Each attribute accepts only a particular type of value: Boolean, or integer, for example. In most cases, WFL enables variables and expressions to be used for the attribute value. For string task attributes that can be null, you can set them to a null value by assigning the string "" or ".".

Task attribute assignments can be used in job attribute lists, task declarations, task assignment statements, and task equation lists. Certain restrictions apply to task attribute

assignments that occur in task declarations. Refer to [Task Variables](#) for more information.

Any task attributes that are not explicitly assigned values in the WFL job receive default values when the task is initiated. These default values come from the following sources:

- If task attribute values are assigned to an object code file when it is originally compiled, these become the default values whenever that object code file is executed. The task attributes compiled into an object code file can later be permanently changed with the MODIFY statement, without recompiling the source file.
- Certain task attributes inherit values from the attributes of the job. For example, the USERCODE attribute receives the value of the USERCODE attribute of the job. To determine whether a particular attribute is inherited in this way, refer to the appropriate attribute description in the *Task Attributes Programming Reference Manual*.
- Each attribute has a particular default value that is used if it is not overridden by any of the previously mentioned causes. Defaults vary from one attribute to another.

If a particular task attribute is assigned values more than once in a given job attribute list, task declaration, task assignment statement, or task equation list, then the last value specified for the task attribute overrides the previous ones.

The complex task attribute assignments are assignments to task attributes that require a fairly complex value. The syntax for each of these attribute assignments appears in [Complex Task Attribute Assignments](#).

The syntax for the various kinds of expressions referred to in the <task attribute assignment> syntax diagram is provided in [Section 7, Expressions](#). The following pages describe the syntax used in WFL for assigning values to the various kinds of task attributes.

Examples

Boolean Assignment

If a Boolean-valued attribute occurs in a task attribute assignment without any value specified for it, it is assigned TRUE. Thus, the following two examples are equivalent:

RUN OBJECT/NATTY;	RUN OBJECT/NATTY;
LOCKED;	LOCKED = TRUE;
TADS;	TADS = TRUE;

A Boolean attribute can also be assigned the result of various comparisons, as in the following example:

RUN OBJECT/KAYA;	
SW1 = X GEQ Y;	% Arithmetic comparison
SW2 = STRING1 EQL STRING2;	% String comparison
SW3 = INFILE (SECURITY) IS PUBLIC;	% File mnemonic comparison
SW4 = TVAR(HISTORYCAUSE)	
IS OPERATORCAUSEV;	% Task mnemonic comparison
SW5 = B1 OR B2;	% Boolean expression

Integer Assignment

The following are examples of integer attribute assignments:

```
PRIORITY = 50;           % Assigns an integer constant.
CLASS = X;               % Assigns an integer variable.
MAXPROCTIME = Y + Z;     % Assigns the value of an integer
                        % expression.
```

The value can also be taken from a task variable defined earlier in the program, if a task variable was associated with that task:

```
PRIORITY = TVAR1 (PRIORITY);
```

Mnemonic Assignment

Mnemonic attribute values can be assigned in one of the following ways.

- From a direct assignment:

```
RUN (WENDY)OBJECT/LPFINDER;
FILEACCESSRULE = DEFAULT;
```

- From a previously defined task variable:

```
TASKA (FILEACCESSRULE = DECLARER);
RUN (WENDY)OBJECT/LPFINDER;
FILEACCESSRULE = TASKA (FILEACCESSRULE);
```

- From a string value:

```
RUN (WENDY)OBJECT/LPFINDER;
FILEACCESSRULE = #STVAR1; % STVAR1 is a string variable.
```

Real Assignment

The following are examples of real attribute assignments:

```
MAXIOTIME = DECIMAL(STR1); % STR1 is a string variable.
MAXPROCTIME = X;           % X is a real variable.
```

Title Assignment

A title-valued attribute accepts a value that follows the syntax of a file title. This is shown in the following example:

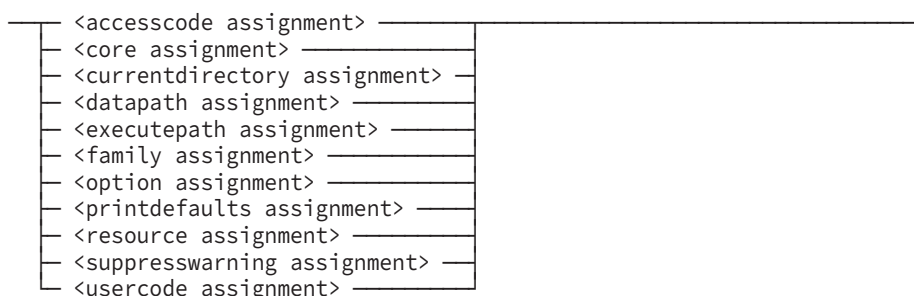
```
JOBSUMMARYTITLE = (WALLY)NUMB/CALC ON SHIPPK;
```

String expressions can also be used in a title assignment:

```
MYJOB(JOBSUMMARYTITLE =
      (#STR1)#STR2/#STR3); % Where STR1, STR2, and STR3
                        % are string variables.
MYJOB(JOBSUMMARYTITLE = #STR4); % Where STR4 contains a
                        % valid file title.
```

Complex Task Attribute Assignments

<complex task attribute assignment>

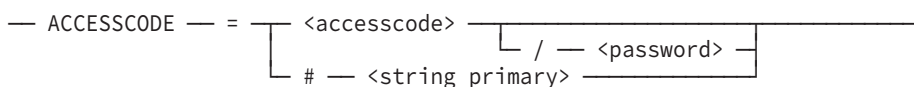


Explanation

The complex task attribute assignments are assignments to attributes that require complex or unusual values. The syntax for these assignments is discussed in the following pages. The uses of these attributes are described in the *Task Attributes Programming Reference Manual*.

ACCESSCODE Assignment

<accesscode assignment>



<accesscode>



Explanation

The ACCESSCODE attribute assigns an accesscode to a task.

The following is an ACCESSCODE assignment:

```
ACCESSCODE = CHARLY/V5;
```

The following assigns a null value to the ACCESSCODE attribute:

```
ACCESSCODE = " ";
```

For an explanation of string primaries, refer to [Using String Primaries](#).

To change the password of a job, you must use the ACCESS statement. Refer to [ACCESS Statement](#) for details.

When accesscode password-aging is enabled, the WFL compiler gives a warning if the accesscode password specified for a job is in the warning state. An error is given if the password has expired. Refer to the *Security Overview and Implementation Guide* for details.

Examples

The following example assigns a string variable that has the accesscode and password as its value:

```
ACCESSCODE = #ACCSTR;
```

The following example assigns separate string variables as values for the accesscode and password:

```
ACCESSCODE = #ACCSTR / #ACCPASTR ;
```

CORE Assignment

<core assignment>

— CORE — = — $\left[\begin{array}{l} \text{<total core>} \\ (\text{— <data core> — , — <code core> — }) \end{array} \right]$ —

<total core>

<data core>

<code core>

— <integer expression> —

Explanation

The CORE attribute provides an estimate of memory required to schedule a task.

Examples

The following example assigns an integer constant as the total core value:

```
CORE = 50;
```

The following example assigns the total core value of the product of X and Y, which are integer variables assigned earlier in the job:

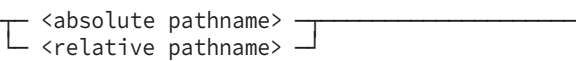
```
CORE = ( X * Y );
```

The following example assigns data core the sum of W and X, and assigns code core the difference of Y and Z. The letters W, X, Y, and Z are previously defined integer variables.

```
CORE = ( W + X, Y - Z )
```

CURRENTDIRECTORY Assignment

<currentdirectory assignment>

— CURRENTDIRECTORY — = 

Refer to the *Task Attributes Programming Reference Manual* for the definition of absolute and relative path names.

Explanation

The CURRENTDIRECTORY attribute specifies the directory that is the starting point for resolution of relative path names. It contains a character string, in path name format, which is prefixed to a file name during such operations as opening or searching for a file when the file name is relative and the SEARCHRULE attribute of the file is POSIX.

If the string that assigns the CURRENTDIRECTORY attribute represents a relative path name, the string is prefixed by the current value of the attribute, and the attribute is set from the combined string. Setting the CURRENTDIRECTORY attribute to a relative path name is permitted only on an active task and only by the associated process.

If the CURRENTDIRECTORY attribute is not explicitly specified on the task variable or code file, the attribute is inherited from the parent task if the parent task's CURRENTDIRECTORY attribute is set. Otherwise, the attribute defaults to a null string.

If a WFL job contains a USERCODE assignment in the job header, but does not contain a CURRENTDIRECTORY assignment, the job's CURRENTDIRECTORY attribute is set to the POSIXINITDIR (POSIX Initial Directory) value from the USERDATAFILE.

Examples

The CURRENTDIRECTORY of the job is set from the USERDATAFILE. PROG1 runs with the value. PROG2 runs with a value of "/-/NEWFAM/USERCODE/MYUC/DIR1", which references the directory (MYUC)DIR1 ON NEWFAM.

```
BEGIN JOB;
USER=MYUC/MYPW;
RUN PROG1;
RUN PROG2;  CURRENTDIRECTORY="/-/NEWFAM/USERCODE/MYUC/DIR1 ";
END JOB.
```

In the following example, PROG3 runs with a CURRENTDIRECTORY value of "/A/B/C", which references the directory *A/B/C on the family specified as the root family by the DL ROOT command:

```
BEGIN JOB;
CURRENTDIRECTORY="/A/B ";
MYSELF ( CURRENTDIRECTORY="C " );
RUN PROG3;
END JOB.
```

DATAPATH Assignment

<datapath assignment>

— DATAPATH — = — $\left[\begin{array}{l} \text{<path specification>} \\ (\text{— <path specification> —}) \end{array} \right]$ —

<path specification>

— $\left[\begin{array}{l} \text{<directory>} \\ \# \text{ —<string expr>} \\ \# \text{ —<string expr>} \end{array} \right] \text{ ON } \left[\begin{array}{l} \text{<family name>} \\ \# \text{ —<string expr>} \end{array} \right]$ —

<directory>

— $\left[\begin{array}{l} * \text{ — DIR — / —<subdirectory>} \\ (\text{— * —}) \\ (\text{—<usercode>—}) \\ * \end{array} \right]$ —

Explanation

The DATAPATH attribute determines the directories that the task uses when creating files or searching for files if the prefix and family name have not been explicitly specified.

String expressions and string primaries that evaluate to constants can be used for DATAPATH in the WFL job heading. WFL job parameters can be used in constant expressions.

When the DATAPATH assignment appears in the job heading or in task equation, parentheses are not allowed. When a DATAPATH assignment appears in a task attribute list, parentheses are required.

Examples

The following is an example of a DATAPATH assignment:

```
DATAPATH = DIR/SHARED ON PROJECTPACK, ( *) ON HOMEPACK;
```

The following DATAPATH assignment uses a previously defined string variable named SDATAPATH:

```
SDATAPATH := "DIR/SHARED ON PROJECTPACK, ( *) ON HOMEPACK";
.
.
.
RUN OBJECT/TEST;
  DATAPATH = #SDATAPATH;
```

The following DATAPATH assignment uses a string expression. SDIR, SFAM, and SPATHELEMENT are previously defined string variables:


```

SDIR := "*DIR/SHARED";
SFAM := "PROJECTPACK";
SPATHELEMENT := " (*) ON HOMEPACK";
.
.
.
RUN OBJECT/TEST;
  DATAPATH = #SDIR ON #SFAM, #SPATHELEMENT, * ON DISK;

```

The following is an example of a DATAPATH assignment in a task attribute list. It uses a string expression. SDIR, SFAM, and SPATHELEMENT are previously defined string variables:

```

SDIR := "*DIR/SHARED";
SFAM := "PROJECTPACK";
SPATHELEMENT := " (*) ON HOMEPACK";
.
.
.
MYJOB (DATAPATH = (#SDIR ON #SFAM,
  #SPATHELEMENT, * ON DISK), VALUE = 5);

```

EXECUTEPATH Assignment

<executepath assignment>

```

— EXECUTEPATH — = [ <path specification> ————— ]
                   ( — <path specification> — )

```

<path specification>

```

— [ <directory> ————— , <family name> ————— ]
   | # —<string expr> |   | # —<string expr> |
   | # —<string expr> |   | # —<string expr> |

```

<directory>

```

— [ * — DIR — / —<subdirectory> ————— ]
   | ( — * — ) ————— |
   | ( —<usercode> — ) ————— |
   | * ————— |

```

Explanation

The EXECUTEPATH attribute determines the directories that the task uses when processing code files if the prefix and family name have not been explicitly specified.

String expressions and string primaries that evaluate to constants can be used for EXECUTEPATH in the WFL job heading. WFL job parameters can be used in constant expressions.

When the EXECUTEPATH assignment appears in the job heading or in task equation, parentheses are not allowed. When an EXECUTEPATH assignment appears in a task attribute list, parentheses are required.

Examples

The following is an example of an EXECUTEPATH assignment:

```
EXECUTEPATH = DIR/SHARED ON PROJECTPACK, ( *) ON HOMEPACK;
```

The following EXECUTEPATH assignment uses a previously defined string variable named SEXECUTEPATH:

```
SEXECUTEPATH := "DIR/SHARED ON PROJECTPACK, ( *) ON HOMEPACK";  
.  
.  
.  
RUN OBJECT/TEST;  
EXECUTEPATH = #SEXECUTEPATH;
```

The following EXECUTEPATH assignment uses a string expression. SDIR, SFAM, and SPATHELEMENT are previously defined string variables:

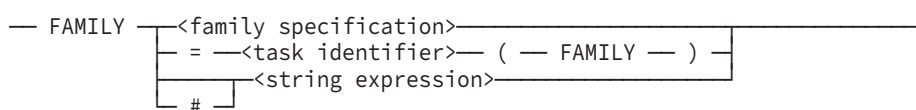
```
SDIR := " *DIR/SHARED";  
SFAM := "PROJECTPACK";  
SPATHELEMENT := " ( *) ON HOMEPACK";  
.  
.  
.  
RUN OBJECT/TEST;  
EXECUTEPATH = #SDIR ON #SFAM,  
#SPATHELEMENT, * ON DISK;
```

The following is an example of a EXECUTEPATH assignment in a task attribute list. It uses a string expression. SDIR, SFAM, and SPATHELEMENT are previously defined string variables:

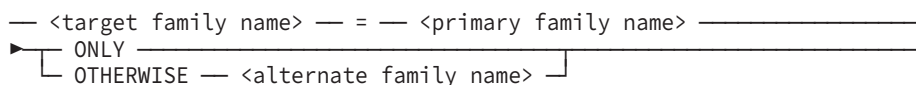
```
SDIR := " *DIR/SHARED";  
SFAM := "PROJECTPACK";  
SPATHELEMENT := " ( *) ON HOMEPACK";  
.  
.  
.  
MYJOB (EXECUTEPATH = (#SDIR ON #SFAM,  
#SPATHELEMENT, * ON DISK), VALUE = 5);
```

FAMILY Assignment

<family assignment>



<family specification>



<target family name>

<primary family name>

<alternate family name>

— <family name> —————|

Explanation

The FAMILY attribute determines the families that the task will use when creating files or searching for files.

String expressions and string primaries that evaluate to constants can be used for FAMILY in the WFL job heading. WFL job parameters can be used in constant expressions.

Examples

The following is an example of a FAMILY assignment:

```
FAMILY DISK = DISK OTHERWISE MYPACK;
```

Note: Most WFL statements can locate a file residing on either the primary family or the alternate family. However, the ADD, ALTER, ARCHIVE, CATALOG, CHANGE, COPY, MODIFY, MOVE, REMOVE, and SECURITY statements do not search the alternate family.

The following FAMILY assignment uses a previously defined string variable named FAMSTRING:

```
FAMSTRING := "DISK = DISK ONLY";  
.  
.  
.  
RUN OBJECT/FUGUE;  
FAMILY = #FAMSTRING1;
```

The following FAMILY assignment uses a string expression. STR1 and STR2 are string variables that were assigned values earlier in the job:

```
FAMILY = #("DISK = " & STR1 & " OTHERWISE " & STR2);
```

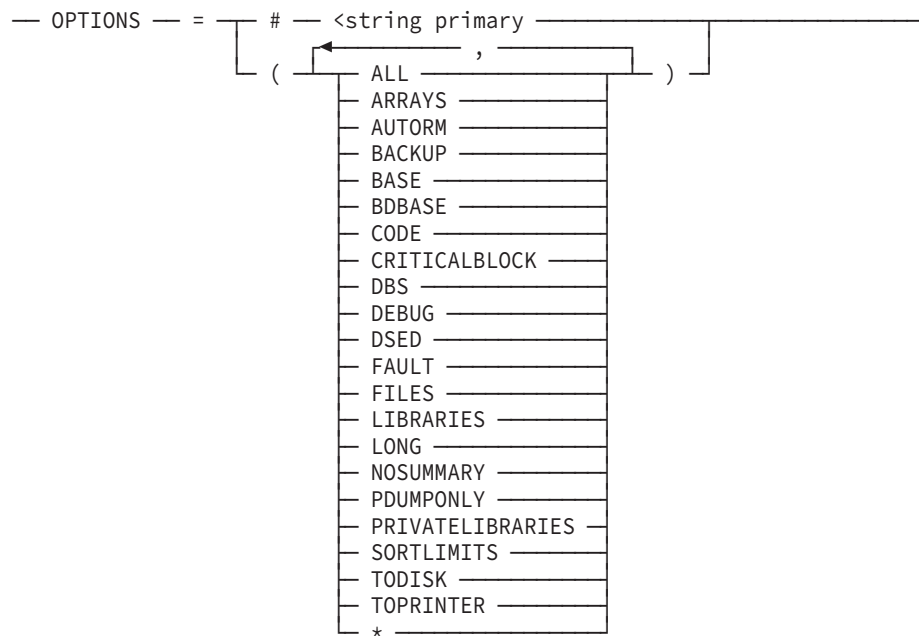
The following example assigns a task the family specification associated with the task variable named TASKVAR1. The task variable can receive a FAMILY assignment when it is declared, or it can inherit a FAMILY assignment from the task to which it was most recently assigned.

```
FAMILY = TASKVAR1 (FAMILY);
```

The FAMILY assignment is unique in that it does not require a number sign (#) before the expression <task identifier> (<attribute name>). Similar expressions, when used for other attributes, do require the number sign.

OPTION Assignment

<option assignment>



Explanation

The OPTION attribute sets options for the task. The options for the task affect program dumps, job summary printing, backup-file handling, and other areas. The options of the OPTION attribute are explained in the *Task Attributes Programming Reference Manual* under the discussion of the OPTION attribute.

If an asterisk (*) is included in the OPTION assignment, then any options provided by a previous assignment are to be merged with the options specified by this assignment. If an asterisk is not included, then any options specified by a previous assignment are not retained when this assignment occurs.

The asterisk option is particularly useful when assigning options to an object code file as defaults, because it enables additional options to be specified at run time without causing the default options being overwritten.

The ALL option is equivalent to specifying all the other options. The ALL option has no effect on the program dump destination. If the PDUMPNLY, TODISK, or TOPRINTER designation option is needed, it must be explicitly mentioned. ALL cannot be specified within a string that is assigned to the OPTION task attribute.

Examples

The following example illustrates the use of the asterisk option:

```

COMPILE OBJECT/
X WITH COBOL85 LIBRARY;      % Compiles source file X and
  COMPILER FILE CARD (TITLE=X,KIND=DISK); % saves object code file
  OPTION=(FAULT);             % OBJECT/
X with FAULT option
      .                        % as default
      .
      .
RUN OBJECT/X;                 % OBJECT/
X runs with OPTION = (FAULT)
      .
      .
      .
RUN OBJECT/X;                 % OBJECT/
X runs with OPTION = (FAULT,
  OPTION = (*,ARRAYS,FILES); % ARRAYS,FILES)

```

The `# <string primary>` syntax enables string variables and expressions to be used to assign the option values. The parentheses around the option values can be omitted when this form of the syntax is used. The following job excerpt illustrates a use of this syntax.

```
STRING S;           % Declares string variable S.
S:="LONG,FAULT";    % Assigns option values to S.
RUN OBJECT/WELLCOMP; % Runs the program with the options
    OPTION=#S;       % LONG and FAULT set.
```

The parentheses before and after the option list are optional when the option assignment occurs in a task equation list, job attribute list, or compiler task equation list. However, the parentheses are required if the `OPTION` assignment occurs in a task declaration or task assignment statement. The following example illustrates `OPTION` assignments in each of these contexts:

```
?BEGIN JOB;
  OPTION = NOSUMMARY, BASE;           % job attribute list
  TASK TVAR (OPTION=(ARRAY,AUTORM),   % task declaration
            MAXPROCTIME=73);
  COMPILE OBJECT/X WITH ALGOL LIBRARY;
    COMPILER FILE CARD (TITLE = X);
    ALGOL OPTION = ARRAY, DSED;       % compiler task equation list
  RUN OBJECT/Z;
    OPTION = LONG, FAULT;             % task equation list
    TVAR (OPTION=(BACKUP),PRIORITY=50); % task assignment statement
?END JOB.
```

PRINTDEFAULTS Assignment

<printdefaults assignment>

```
— PRINTDEFAULTS — = — ( — <printdefaults assignment list> — ) —
```

Explanation

The PRINTDEFAULTS attribute enables you to provide a different set of default values (in place of the system default values) for the print modifiers or print-related file attributes that control the creation, routing, and formatting of backup files. If the PRINTDEFAULTS attribute is set for a task, the default values specified are applied to all PRINT statements and any printer backup files created by the task.

The print attribute phrases and print modifier phrases that appear in the printdefaults assignment list are merged into the current print defaults. The print modifier and print attribute forms reestablish the system default value for that print modifier or print-related file attribute. Refer to [PRINT Statement](#) for the syntax of the <printdefaults assignment list>, and for more information about print modifiers and print-related file attributes.

The PRINTDEFAULTS attribute can be included in the USERDATAFILE associated with each usercode. For further information, see "MAKEUSER Utility" in the *System Software Utilities Operations Reference Manual*.

Examples

To establish print default values for a job, use a task assignment statement such as the following:

```
MYJOB (PRINTDEFAULTS=(DOUBLESPACE=TRUE,AFTER="20:00"));
```

The default values established for the job are inherited by any tasks initiated by the job. The defaults for the job can be varied for different parts of the job by using several task assignment statements. To establish default values for a task, you can assign values directly to the PRINTDEFAULTS attribute of a task, as in the following example:

```
RUN (SANTA)GIFT/LIST;  
PRINTDEFAULTS=(DESTINATION="LP14",SAVEPRINTFILE=TRUE);
```

The assigned values override the default values that the task would otherwise inherit from the job.

Each file declaration in a task can override the default file attributes and print modifier values assigned to that task. The attribute values specified in a file declaration in the task can, in turn, be overridden through the file equation. File equations can be used to assign file attributes to files used by a task.

For example, if a program named OBJECT/BOG prints a file called OUTFILE, the following example would modify the way the file is printed:

```
RUN OBJECT/BOG;  
FILE OUTFILE (SAVEPRINTFILE,PRINTCOPIES=3,AFTER="23:00");
```

Print modifiers cannot be specified in file equations; they can only be included in PRINT statements or PRINTDEFAULTS assignments.

RESOURCE Assignment

<resource assignment>

— RESOURCE — = —————→
 ▶ (— TAPE — = — <integer constant expression> —) —————|

Explanation

The RESOURCE attribute specifies how many tape units are needed by a task.

Example

The following example assigns values to the RESOURCE attribute:

```
?BEGIN JOB ATTRBTEST(INTEGER PARAM);
  RUN (RAJA)OBJECT/WEATHER/FORECAST;
  RESOURCE = (TAPE = PARAM);
?END JOB.
```

This resource assignment indicates that the program requires the number of TAPE drives passed to the job in the parameter PARAM. Job parameters, such as PARAM in the previous example, can be used in the RESOURCE assignment because they are constant identifiers rather than variables.

Note: The integer constant expression must be an integer in the range of 0 to 255 when it is completely evaluated.

SUPPRESSWARNING Assignment

<suppresswarning assignment>

— SUPPRESSWARNING — = — <suppresswarning list> —————|

<suppresswarning list>

ALL —————|
 NONE —————|
 <hyphen> <warning number> , <hyphen> <warning number> |

<warning number>

An unsigned integer in the range of 1 through 29999.

Explanation

The SUPPRESSWARNING attribute suppresses run-time warning messages for a process. Most of these messages are warnings that indicate the process has just used a feature that is scheduled for deimplementation on a future release.

Task Initiation

You can suppress specific run-time warning messages by assigning a set of warning numbers or warning number ranges to the SUPPRESSWARNING attribute. Each warning number corresponds to a particular run-time warning message. If the SUPPRESSWARNING list begins with a hyphen (-), the hyphen is interpreted as a minus sign and the warning numbers following the hyphen are deleted from the previously created SUPPRESSWARNING list.

Refer to the *Task Attributes Programming Reference Manual* for additional information regarding the SUPPRESSWARNING task attribute.

Examples

The following is a simple SUPPRESSWARNING assignment that causes all run-time warning messages to be suppressed:

```
SUPPRESSWARNING = "ALL";
```

The following assignment suppresses the run-time warning messages that are in the ranges 1 through 25, 32 through 45, and 100 through 199:

```
SUPPRESSWARNING = "1-25,32-45,100-199";
```

The following assignment deletes messages 10 through 20 from the SUPPRESSWARNING list that was created in the previous example:

```
SUPPRESSWARNING = "-10-20";
```

USERCODE Assignment

<usercode assignment>

```
— USERCODE — = ———— <usercode> ———— / ———— <password> ————  
                  |  
                  # — <string primary> ————
```

Explanation

The USERCODE attribute assigns a usercode and its associated privileges to a task. The password must be included if the usercode has a password associated with it.

The following assigns a null value to the USERCODE attribute:

```
USERCODE = " ";
```

For an explanation of string primaries, refer to [Using String Primaries](#).

A password-aging feature is available through the InfoGuard Password Management facility. If the password-aging feature is enabled, the WFL compiler gives a warning if the usercode password specified for the job is in the warning state. An error is given if the password has expired. The password-aging feature is discussed further in the *MCP Security Overview and Implementation Guide*.

The USER statement changes the usercode of the job or processed subroutine. The password associated with the usercode of the job can be changed with the PASSWORD statement. These statements are described later in [Section 6, Statements](#).

Examples

The following is a simple USERCODE assignment:

```
USERCODE = WALLY/BEAR;
```

The following example uses a string expression:

```
USERCODE = #(USER1/PASS1);
```

The following example takes the usercode value from a task variable:

```
TASK TASKVAR (USERCODE = RAJA/RANI);      % Task variable declaration
RUN (WALLY)OBJECT/ALPHA;                  % Run statement with
    USERCODE = #TASKVAR(USERCODE) / RANI; % usercode specification
```

Note: A string task attribute primary (such as **#TASKVAR(USERCODE)**, in the preceding example) gives the value of a usercode, but not its associated password. In this example, the password is given separately in the second USERCODE assignment.

Using Task Variables

Explanation

Task variables are variables that can be associated with particular tasks. The means of defining them are provided in [Task Variables](#). Task variables are used for two basic purposes: as a means of assigning task attributes or file equations to a task, and as a means of inquiring about the attributes of a task.

Any number of task variables can be declared and used in a job. Also, a particular task variable can be associated with more than one task in the course of a job. However, a task variable can only be attached to one task at any given time and cannot be reused until that task has terminated. Care should be taken when reassigning a task variable that was associated with an asynchronous task.

Examples

For example, the following statement is permissible:

```
RUN X [T];
INITIALIZE (T);
COPY A AS B [T];
```

However, a run-time error results from the following:

```
PROCESS COMPILE X WITH ALGOL LIBRARY [T];
RUN Y [T];
```

If a task variable is not associated with an asynchronous task, the WFL compiler will allocate a task variable internally and associate it with the task.

Note that a run-time error will result if the PROCESS statement, which initiates tasks asynchronously, is executed in either of the following ways:

- Repeatedly with the same task variable, while the previously processed task is still active.
- In a loop (one that uses the DO or WHILE statement, for example) with no task variable, while the previously processed task is still active. The error occurs because the internal task variable is associated with more than one task at the same time.

Assigning Task Attributes

When a task variable is associated with a task, any task attribute assignments or file equations currently assigned to that task variable are assigned to the task.

If a task variable is to be used to assign task or file attribute values to a task, these values can be specified in the task declaration that declares that task variable. (Refer to [Declaration Syntax](#).)

The attribute values associated with the task variable can be added to or changed in any of the following ways:

- Task assignment statements can be used to associate additional task attribute assignments and file equations with the task variable, or to override values that had been specified for that task variable previously. (Refer to the [Assignment Statements](#) for a description of the task assignment statement.)
- Task attribute assignments in a task assignment statement can be applied in any order, not necessarily in the order listed. If the order is important, the task assignments should be put into separate task assignment statements. Thus, a statement such as the following is undesirable:

```
TVAR( STATUS=NEVERUSED, FAMILY DISK = PARTS ONLY );
```

The FAMILY assignment in this example can be executed before the STATUS assignment. If it is, the FAMILY value will be discarded.

- When the task variable is used in a task initiation statement, and attribute assignments follow the task initiation statement, they are added to the values that are currently assigned to the task variable. Where there is a conflict, the values following the task initiation statement override those currently associated with the task variable.
- When the task variable is used in a task initiation statement, and the task is a program, any attribute assignments associated with the object code file of the program are added to those currently associated with the task variable. Where there is a conflict, the values currently associated with the task variable override those associated with the object code file.
- If, at any point in the job, the INITIALIZE statement is used to reinitialize the task

variable, then all task and file attributes previously associated with the task variable are discarded. For example, the following statement will reinitialize the task variable TVAR:

```
INITIALIZE (TVAR);
```

Examples

In the following example, the program (RAJA)OBJECT/ALT runs with the task and file attributes given for the task variable TVAR in the task declaration:

```
?BEGIN JOB;
  TASK TVAR (PRIORITY=50,                % Task variable declaration
            FILE INFILE (TITLE = F/T));
  RUN (RAJA)OBJECT/ALT [TVAR];          % Task initiation statement
?END JOB.
```

The following example shows several ways in which a task variable can be assigned task attribute values:

```
?BEGIN JOB ATTRIBUTES;
  TASK T (PRIORITY=50,USERCODE=JAMES/PW, % Task declaration
        FAMILY DISK = DISK ONLY);
  RUN (PARTS)PROG [T];                  % Task initiation
    MAXPROCTIME = 90;
  INITIALIZE (T);                      % Initialize statement
  T (MAXPROCTIME=30);                  % Task assignment statement
  RUN (PARTS)SUMMARY [T];
?END JOB.
```

Reusing Task Variables

When a task variable is to be reused for another task, the task variable should normally be reinitialized first. Otherwise, the task variable retains the task attribute values that were associated with the previous task, and these can sometimes cause undesirable side effects.

Examples

The following examples show some of the possible side effects of reusing task variables without reinitializing them.

The following example runs SYSTEM/CARDLINE twice:

```
?BEGIN JOB TASK/TEST;
  TASK T;
  RUN SYSTEM/CARDLINE[T];
    FILE CARD (KIND=DISK,TITLE=INPUT1);
    FILE LINE (KIND=DISK, PROTECTION=SAVE);
  RUN SYSTEM/CARDLINE[T];
    FILE CARD (KIND=DISK,TITLE=INPUT2);
?END JOB.
```

Task Initiation

This program takes an input file named CARD and produces an output file named LINE, which by default is a printer file.

The first time SYSTEM/CARDLINE is run, the file LINE is file-equated to have a KIND of DISK, and the output is thus sent to a disk file. This file equation is also associated with the task variable T. When SYSTEM/CARDLINE is run the second time, no file equation for the file LINE is explicitly included, but the file equation is passed on by task variable T, and the output is still sent to a disk file.

In the following example, T is the declared task used for two RUN statements. In the second RUN statement, the PRIORITY information, file equation information, and setting of the OPTION attribute are still in effect when Y is executed. When T is re-used, less obvious side effects might also occur. For example, if X changes its TASKVALUE, that TASKVALUE would still be in effect when Y is executed, which might cause undesirable side effects on Y.

```
?BEGIN JOB EXAMPLE1;
  TASK T;
  T(PRIORITY=50);
  RUN X[T];
    FILE F(KIND=DISK);
  T(OPTION=(FAULT));
  RUN Y[T];
?END JOB.
```

The following example includes two COMPILE and GO statements that both have the same task variable [T] associated with them. The task variable T has PRIORITY=50 assigned to it; this priority is applied to the GO part of both compiles.

File equations made to the first COMPILE statement also affect the second COMPILE statement, because the task variable retains the file equations. Because of this fact, the COMPILER FILE CARD equation after the first COMPILE statement has the (probably unintentional) effect of causing the second COMPILE to also read from the disk file named INPUT.

Both COMPILE statements are followed by file equations that affect the same file F; in this case, the first COMPILE statement treats F as a disk file, and the second COMPILE statement treats F as a remote file:

```
?BEGIN JOB EXAMPLE2;
  TASK T;
  T(PRIORITY=50);
  COMPILE OBJECT/XDISK [T] WITH ALGOL GO;
    COMPILER FILE CARD(KIND=DISK, TITLE=INPUT);
    FILE F(KIND=DISK);
  COMPILE OBJECT/XREM [T] WITH ALGOL GO;
    FILE F(KIND=REMOTE);
?END JOB.
```

The following example illustrates how the INITIALIZE statement can be used to prevent the attribute values of one task from being accidentally assigned to the next task:

```
?BEGIN JOB EXAMPLE3;
  TASK T;
  T(PRIORITY=50);
```

```

RUN X[T];
  FILE F(KIND=DISK);
  INITIALIZE(T);
  T(OPTION=(FAULT));
  RUN Y[T];
?END JOB.

```

The task variable T is reinitialized, after which the PRIORITY information and the previous file equation information are no longer in effect. Thus, program Y runs with the FAULT option set, but with no other task or file equations.

Interrogating Task Attributes

A very useful feature of task variables is that they enable you to inquire about the attributes of a task after it is initiated. The syntax of expressions that use task variables to return task attribute values is provided in [Section 7, Expressions](#). [Table 5-2](#) lists the expressions that are used to inquire about each type of task attribute.

Table 5-2. Expressions for Task Attribute Inquiry

Task Attribute Type	Expression Used
<Boolean task attribute>	<Boolean task attribute primary>
<file name task attribute>	<string task attribute primary>
<integer task attribute>	<integer task attribute primary>
<mnemonic task attribute>	<string task attribute primary> <task mnemonic comparison>
<name task attribute>	<string task attribute primary>
<real task attribute>	<real task attribute primary>
<string task attribute>	<string task attribute primary>
<title task attribute>	<string task attribute primary>

Table 5–2. Expressions for Task Attribute Inquiry (cont.)

Task Attribute Type	Expression Used
<complex task attribute>	<p><string task attribute primary></p> <p>This expression is used for these attributes:</p> <ul style="list-style-type: none">• ACCESSCODE• DATAPATH• EXECUTEPATH• OPTION• FAMILY• USERCODE <p>The following attributes cannot be interrogated:</p> <ul style="list-style-type: none">• CORE• PRINTDEFAULTS• RESOURCE

Interrogating Task Status

The status or history of a task can be inquired about in either of two ways:

- By using a string task attribute primary to interrogate the value of the STATUS attribute of the task.

Refer to the description of task mnemonic comparisons under [Boolean Expressions](#) for an example. Refer to the description of the STATUS attribute in the *Task Attributes Programming Reference Manual* for a list of the mnemonic values the attribute can have.

- By using the Boolean task state expression.

This is not actually a task attribute inquiry, but it does return information about a task. Refer to “Task State” under [Boolean Expressions](#).

Both of these inquiries return similar types of information about the task, but use different mnemonic values to express the information.

File Attribute Inquiry

The task variable cannot be used to interrogate the attributes of files used by a task. Refer to [Interrogating Complex Task Attributes](#) for a discussion of how to interrogate file attributes.

Interrogating Complex Task Attributes

The following points should be remembered when interrogating the values of complex task attributes.

- Inquiries about the USERCODE attribute return the usercode only, without the password.
- Inquiries about the ACCESSCODE attribute return the accesscode only, without the accesscode password.
- Inquiries about the FAMILY attribute return a value with one of the following forms:


```
<target family name> = <primary family name> ONLY
<target family name> = <primary family name> OTHERWISE
<alternate family name>
" " % EMPTY STRING
```
- Inquiries about the OPTION attribute return a string value that is a list of all the options that are set, separated by commas (,). The options can be listed in any order. Thus, the expression STR := TVAR(OPTION); could return any of the following values, among others:

```
"ARRAYS, FILES, LONG"
"CODE"
"CODE, BDBASE, AUTORM, LONG"
```

Example

The following is an example of an expression that can be used to extract the primary family name from the FAMILY attribute:

```
IF LENGTH(MYSELF(FAMILY)) GTR 0 THEN
  MYFAMILY := HEAD(DROP(TAIL(MYSELF(FAMILY),NOT "=",
                          2),NOT " "));
ELSE MYFAMILY := "DISK";
```

In this example, MYFAMILY is a string variable and MYSELF is the predefined task variable. Any string variable or task variable could be used in their places. Refer to [String Expressions](#) for explanations of the HEAD, DROP, and TAIL functions.

MYJOB and MYSELF Predeclared Task Variables

MYJOB and MYSELF are predeclared task variables that can be used to assign or interrogate task attribute values. The MYJOB task variable is always assigned to the job itself. If the MYJOB task variable is used in a task assignment statement, the values specified for it are assigned to the job.

The syntax that applies to declared task variables can be used with the predeclared task variables MYJOB and MYSELF, except for the specific cases documented in “Task State” under [Boolean Expressions](#).

Examples

For example, the following statement assigns a value to the FAMILY attribute of the job:

```
MYJOB(FAMILY DISK = MYPACK ONLY);
```

Any task attributes you wish to specify for the job will normally be included at the start of the job in the job attribute list. Thus, a task assignment statement using MYJOB is most useful when you want to change the job attributes later in the job.

The following example uses MYJOB in a task attribute inquiry:

```
RUN SKYVAL; STATION = MYJOB(SOURCESTATION);
```

This example takes the value of the SOURCESTATION attribute of the job and assigns it to the STATION attribute of a task.

The following statement assigns a value to the BDNAM attribute, so that backup files are created under the directory MYBACKUPFILES:

```
MYJOB(BDNAM = MYBACKUPFILES);
```

BDNAM can also be reset by equating it to an empty string (or a string expression that has the value of an empty string).

In an asynchronous subroutine, the MYSELF task variable is associated with that subroutine; otherwise, it is assigned to the job and is synonymous with MYJOB. For example, the following statement stores the value of the MIXNUMBER attribute of the asynchronous subroutine or job in the variable I:

```
I := MYSELF(MIXNUMBER);
```

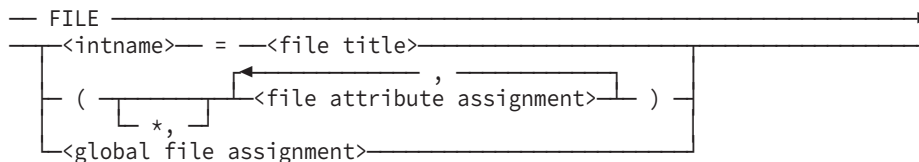
The following statement assigns the value 9 to the TASKVALUE attribute of the asynchronous subroutine or job:

```
MYSELF(TASKVALUE = 9);
```

These predeclared task variables lose their special meaning if they are explicitly declared in a job.

File Equations

<file equation>



Explanation

File equations can be used for any of the following reasons:

- To cause a task to read from or write to different files from those it normally would
- To change the attributes of files that a task reads from or writes to
- To cause a task to read from global data specifications or local data specifications instead of from the input files the task would normally have used

File equations can appear in task equation lists, compiler task equation lists, task declarations, and task assignment statements.

The inname used in a file equation should be the internal name used for a particular file in the task to which the file equation is applied.

FILECARDS is an acceptable synonym for FILE.

Causing the Task to Use a Different Input or Output File

The task can be made to use a different file than it would normally by file-equating the TITLE attribute of the file. The simplest way to do this is by using the following syntax:

```
<inname> = <file title>
```

If the file title includes "ON" the value of the KIND attribute will be set to PACK.

Examples

The following example illustrates the use of the <inname> = <file title>

```
RUN OBJECT/PROG;  
FILE INFILE = (JACOB)INPUT/DATA ON ORDSPK;
```

This form of the file equation syntax changes the TITLE attribute of the file. When the task tries to open the file that has the internal name INFILE, the file equation will cause it to open the file (JACOB)INPUT/DATA ON ORDSPK.

The following is an example of alternate syntax for changing the TITLE attribute of a file:

```
RUN OBJECT/PROG;  
FILE INFILE (TITLE = (JACOB)INPUT/DATA ON ORDSPK);
```

This syntax enables other attributes to be specified at the same time. For more information, refer to [Assigning File Attributes](#).

Changing the Attributes of Files Used by the Task

Any or all of the attributes of a file used by a task can be assigned values in a file equation.

When the program opens a file that has been file-equated, the attribute values specified for that file in the WFL job are merged with those specified in the file declaration in the program. If a file attribute is assigned a different value in the file equation than it is in the file declaration in the program, then the value assigned in the file equation takes precedence. If a file that is file-equated is not opened by the program, then the file equation has no effect.

Certain combinations of file attribute assignments might not be legal; for example, the value of the BLOCKSIZE attribute must be evenly divisible by the value of the MAXRECSIZE attribute. Thus, a file equation that sets MAXRECSIZE to a value incompatible with the BLOCKSIZE attribute would generate a run-time error.

In cases where a program uses a file that already exists (rather than creating one), any file equations included for that file in the WFL jobs do not affect the physical file that the program uses. Instead, such file equations alter the way the program reads from or writes to the file.

For more information about file attributes, refer to the *File Attributes Programming Reference Manual*.

Example

The following is an example of a file equation that assigns several attributes:

```
FILE INFILE (TITLE=(WALLY)OD/CON ON PARTS,KIND=DISK,  
MAXRECSIZE=12,SECURITYTYPE=PUBLIC) ;
```

Causing the Task to Read from a Data Specification

For examples of file equations that cause a task to read from data specifications instead of from its normal input files, refer to [Global Data Specifications](#) and to [Local Data Specifications](#).

How the Task Can Override WFL File Equations

It is possible for a task to override the WFL file equations that are assigned to it. WFL file equations are merged only with those attributes specified in the file declaration in the task. File attribute assignments made later in the task will override WFL file equations.

Resolving Repeated File Equations to the Same File

When a given task equation list, compiler task equation list, task declaration, or task assignment statement includes two or more file equations that apply to the same internal file, the WFL compiler uses the following rules to decide which of the specified file attribute assignments to use. These rules differ according to whether the inname used is an identifier or a string primary.

If the inname used is an identifier, then the following rules apply:

- If a file equation includes an asterisk (*) before the file attribute assignments, then its file attribute assignments are merged with any that have been specified for the file in a previous file equation.
- If a file equation does not include the asterisk, all task attribute assignments specified in previous file equations are discarded, and only the ones given in the latest file equation are used.

If the inname used is a string primary, then the following rules apply:

- The file equation cannot contain an asterisk; if it does, a syntax error is given.
- If the string primary evaluates to the same internal file name as was specified in a previous task equation, a run-time error is given.

Examples

In the following example, X1 is an identifier that specifies an inname. Because the second file equation contains no asterisk, the first file equation is discarded, and the values specified for the KIND and TITLE attributes are not used.

```
RUN (FOLKS)OBJECT/FREELoad;
  FILE X1(KIND=DISK,TITLE=TEST/X1);
  FILE X1(MAXRECSIZE=20,UNITS=CHARACTERS);
```

In the following example, X1 is still an identifier that specifies an inname. However, because the second file equation includes the asterisk, the attributes from both file equations are merged:

```
RUN (FOLKS)OBJECT/FREELoad;
  FILE X1(KIND=DISK);
  FILE X1(*,TITLE=TEST/X1);
```

Global File Assignment

<global file assignment>

```
— <name constant> — := — <file identifier> —————|
— <identifier> — := — <file identifier> —————|
```

Explanation

A global file assignment is used to replace a file used by a program with a file declared in the WFL job.

The identifier specified must be the internal name of the file used by the program. The file identifier specified is the identifier that was used in the WFL file declaration.

Example

The following is an example of a global file assignment:

```
FILE INFILE1 := GLOB2;
```

The colon (:) before the equal sign (=) is important because if it were left out the statement would be interpreted as a title assignment for the file rather than a global file assignment.

The attribute values of the file that is assigned can be supplied in the file declaration or in later file assignment statements; the system supplies default values for attributes that are not explicitly assigned values in the job.

When a global file assignment replaces a file used by a program, all the attributes associated with the original file are ignored. Therefore, if you want to alter only selected attributes of a file used by a program, a global file assignment should not be used. Instead, use the method described earlier in [Changing the Attributes of Files Used by the Task](#).

Note: *If a global file is used by a COBOL74 program and is used again in the same WFL job, the COBOL74 program cannot close the global file with the LOCK phrase. However, the global file can be closed with the SAVE phrase. The COBOL74 "CLOSE WITH LOCK" statement is specifically implemented for the purposes of preventing access from within the same program/run unit/job once the file has been closed.*

Global file assignments make it possible to find out whether a task has changed any of the attributes of a file. Refer to [Interrogating File Attributes](#) for details.

You should be cautious about using global file assignments to cause more than one program to use the same file. Refer to [Assigning File Attributes](#).

Using Remote Files

When a program needs to read from or write to a remote device attached to the system, it opens a remote file and associates it with that device. A common example of this is a program that sends data to or accepts input from the terminal where the program was initiated. Special measures are required when using WFL to initiate a program that reads from or writes to a remote file.

When a program tries to open a remote file, by default it tries to associate that file with the station the program was initiated from. However, if the program was initiated by a WFL job, then the program is not associated with a station, and an error is generated when it attempts to open the remote file.

To prevent this problem, you can include the following statement near the start of the WFL job, before any tasks are initiated:

```
MYJOB ( STATIONNAME = #MYSELF ( SOURCENAME ) ) ;
```

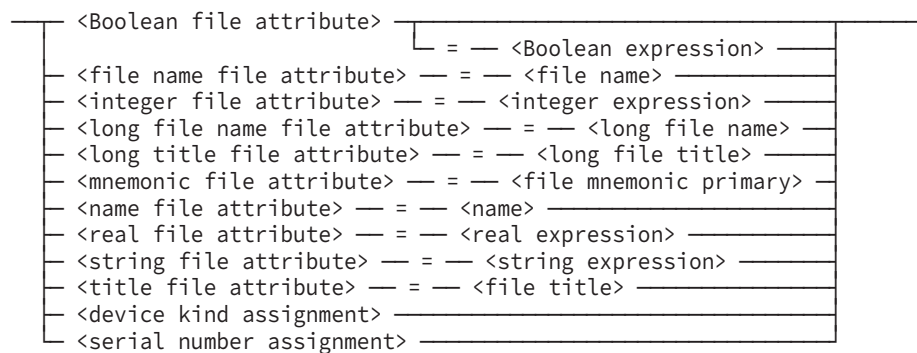
This statement causes the STATIONNAME attribute of the WFL job to store the name of the station that initiated the job. Any tasks of that job inherit the STATIONNAME value. When any of these tasks opens a remote file, the remote file is linked by default to the station that initiated the WFL job.

Before the STATIONNAME task attribute was implemented, the STATION task attribute was the only method of specifying the default station for remote files. The STATION attribute identifies a station by its logical station number (LSN). Use STATIONNAME rather than STATION because the station name is stable, whereas the LSN is subject to change.

The setting of the CANDE LAISSEZFILE option affects the ability of the program to open remote files that are not directly associated with its own session. According to the value of LAISSEZFILE, the task attribute assignment in the previous example might be sufficient to enable the program to open the remote file, or the system might ask for an OK from the station before the remote file is opened, or the program might be prohibited from opening the remote file. Refer to remote files in the *CANDE Configuration Reference Manual* for further information.

Another approach to dealing with programs that use remote files would be to file-equate the remote files to a different KIND, such as DISK or READER.

File Attribute Assignment



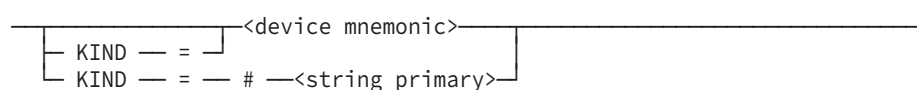
Explanation

The file attribute assignment assigns values to the attributes of files referenced in a WFL job. File attribute assignments can be used in file declarations, file assignment statements, and file equations. Attribute values in file declarations must be constants or constant expressions.

The correspondence between the types of file attributes referred to in this manual and the types described in the *File Attributes Programming Reference Manual* are described in [Using File Attributes](#) later in this section.

Device Kind Assignment

<device kind assignment>



Explanation

The KIND attribute specifies the type of device the file resides on.

The KIND = portion of the syntax is optional; it is enough to simply list the device mnemonic.

Examples

The following two examples are equivalent:

```
FILE FILEA (TITLE = A/B, KIND = DISK);  
FILE FILEA (TITLE = A/B, DISK);
```

The device mnemonic assigns a mnemonic value to the KIND attribute. For a list of the valid mnemonic values, refer to the description of the KIND attribute in the *File Attributes Programming Reference Manual*.

Serial Number Assignment

<serial number assignment>

— SERIALNO — = — <serial number list> —————

<serial number list>

— <serial number> —————
|
| (— /9999\ — <serial number> —) |

<serial number>

— <integer expression> —————
|
| <string expression> —————

Explanation

Serial number assignments assign values to the SERIALNO file attribute. When the WFL job or its tasks create or search for the file, they do so on the volumes with the specified serial numbers. For further details, refer to the *File Attributes Programming Reference Manual*.

The value assigned to the SERIALNO attribute is a string six characters long. If the serial number given is an integer constant, it must be a positive integer less than or equal to 999999. The integer constant is automatically converted to a string with the correct length. If the serial number given is a string constant less than six characters long, then it is automatically padded with blanks at the end to ensure that the resulting string is the correct length.

If a real constant or expression is specified for a serial number, it will be integerized with truncation. This is equivalent to the WFL statement INTEGER (<real expression>).

When specifying a list of serial numbers for the reels of a multiple reel tape file, an empty serial number can be specified for any of the reels.

Example

The following file equation assigns no serial number to the third reel of file T1:

```
FILE T1(KIND=TAPE, SERIALNO=(1,2,,4));
```

Using File Attributes

[Table 5–3](#) shows how the attribute types listed in the *File Attributes Programming Reference Manual* correspond to the types used in this manual. Attribute types that cannot be accessed through WFL are listed as not available.

Table 5–3. File Attribute Types

File Attribute Type	Corresponding WFL Type	Valid File Attributes
Boolean	<Boolean file attribute>	All Boolean attributes
Event	Not available	None of the attributes
Integer	<integer file attribute>	All integer attributes
Mnemonic	<mnemonic file attribute>	All mnemonic attributes
Pointer	<file name file attribute>	The following attributes are valid: <ul style="list-style-type: none">• FAMILYOWNER• FILENAME• PRINTCHARGE• STATIONLIST
Pointer	<long file name file attribute>	LFILENAME attribute
Pointer	<long title file attribute>	LTITLE attribute

Table 5–3. File Attribute Types (cont.)

File Attribute Type	Corresponding WFL Type	Valid File Attributes
Pointer	<name file attribute>	<p>The following attributes are valid:</p> <ul style="list-style-type: none"> • APPLICATIONGROUP • FAMILYNAME • HOSTNAME • INTNAME • MYHOST • MYHOSTGROUP • SCRATCHPOOL • YOURHOST • YOURHOSTGROUP • YOURUSERCODE
Pointer	<string file attribute>	<p>The following attributes are valid:</p> <ul style="list-style-type: none"> • AFTER • DESTINATION • FORMID • LICENSEKEY • NOTE • PATHNAME • RELEASEID • TRANSFORM • WARNINGS
Pointer	<title file attribute>	<p>The following attributes are valid:</p> <ul style="list-style-type: none"> • ALIGNFILE • COPYNAME • MYNAME • SECURITYGUARD • STATIONNAME • TITLE • YOURNAME
Real	<real file attribute>	All real attributes
Translatable	Not available	None of the attributes

Table 5–3. File Attribute Types (cont.)

File Attribute Type	Corresponding WFL Type	Valid File Attributes
Word	<real file attribute>	The following attributes are valid: <ul style="list-style-type: none"> • STATE • TIMESTAMP
Word	<serial number list>	SERIALNO attribute

Assigning File Attributes

The attributes of files that are referenced in a WFL job can be assigned values in file declarations and file assignment statements. One point to be aware of when using a file assignment statement is that the attribute assignments can be executed in any order, not necessarily in the order listed. Thus, file assignment statements such as the following should be avoided:

```
F (TITLE=(SUPPLIES)COUNTM,KIND=DISK,
  DEPENDENTSPECS=TRUE,OPEN=TRUE)
```

This example is intended to set certain attributes for a file, and then open it. However, the OPEN assignment might be executed before the others, which would not give the desired effect. The preferred method would be to use a separate OPEN statement following the file assignment statement.

The OPEN attribute of a file is implicitly set by the OPEN statement, and reset by the CRUNCH, LOCK, PURGE, RELEASE, and REWIND statements. These statements have additional effects aside from setting or resetting this attribute; see the descriptions of these statements in [Section 6, Statements](#).

The OPEN statement can have the additional effect of assigning the logical file all attributes of the physical file. This will occur if the physical file was pre-existing (not created by the job) and the file is declared in the job with the DEPENDENTSPECS attribute set to TRUE.

Example

The following file declaration associates the attributes of the physical file, (JACOB)OBJECT/LINE, with the logical file, FILEA, when the file is opened:

```
FILE FILEA (TITLE=(JACOB)OBJECT/LINE,KIND=DISK,
  NEWFILE=FALSE,DEPENDENTSPECS=TRUE);
```

This declaration provides the title and location of the file. The NEWFILE=FALSE assignment indicates that a file with the specified title and location already exists and is to be used rather than creating a new file of the same name.

When a file is declared in this way, the attributes that are not mentioned in the declaration assume their default values until the file is opened. Thereafter, the attributes will have the values of the physical file.

If a global file assignment is used to cause the task to use a file declared in the WFL job, then any changes that the task makes to the file attributes are retained by the file declared in the job. This is useful for making file attribute inquiries possible, but also can cause side effects if the file is reused by another task.

These side effects can arise because tasks might have conflicting expectations about whether a file is open or closed, and where the current record of the file is positioned. WFL does not automatically close a file at the end of a task; however, the task can close or rewind the file, and explicit file closing statements can be included in the WFL job to accomplish the same result.

Interrogating File Attributes

WFL includes a number of expressions that can be used to interrogate the attributes of files. Depending on the type of attribute involved, these expressions might return integer, real, Boolean, or string values and can be used wherever that type of value is permitted in the job.

[Table 5-4](#) lists all these expressions. Refer to [Section 7, Expressions](#), for the syntax of each expression.

Table 5-4. Expressions for File Attribute Inquiry

File Attribute Type	Expression Used
<Boolean file attribute>	<Boolean file attribute primary>
<file name file attribute>	<string file attribute primary>
<integer file attribute>	<integer file attribute primary>
<long file name file attribute>	<string file attribute primary>
<long title file attribute>	<string file attribute primary>
<mnemonic file attribute>	<string file attribute primary> <file mnemonic comparison>
<name file attribute>	<string file attribute primary>
<real file attribute>	<real file attribute primary>
<string file attribute>	<string file attribute primary>
<title file attribute>	<string file attribute primary>
KIND	<string file attribute primary>
SERIALNO	The value of this attribute cannot be interrogated.

In general, a file must have been declared in the job before its attributes can be interrogated. (The only exception is the file residence inquiry.) The syntax for file declarations is given in [Section 4, Declarations](#). The file should be declared with DEPENDENTSPECS set to TRUE, as described in [Assigning File Attributes](#). Once the file is opened, the job can inquire about any of the attributes of that file.

File attribute inquiries have a wide variety of applications in WFL jobs. For example, the attributes of a file can be interrogated both before and after they are used by a task to see if the task altered the attributes. However, the task must be made to use a file that was declared in the job, because only the attributes of declared files can be interrogated. A global file assignment can be used to make the task use a file. (Global file assignment is described in [File Equations](#).)

Another expression that is similar to a file attribute inquiry is the file residence inquiry, which checks on the residence of a file. This same information can be obtained by using a Boolean file attribute primary to interrogate the RESIDENT attribute of a file. However, the file residence inquiry has the advantage that it can be applied to files regardless of whether they were declared in the job. Refer to [Section 7, Expressions](#), for more information on file residence inquiry.

Examples

The following example checks the length of a file by interrogating the value of the LASTRECORD attribute. The file is copied to tape and removed from disk if it is larger than 10,000 records.

```
OPEN (FILEA);
I := FILEA (LASTRECORD) + 1;    % LASTRECORD is 0-relative
IF I GTR 10000 THEN
  BEGIN
    WAIT ("MOUNT TAPE NAMED FILETP FOR BACKUP",OK);
    COPY FILEA FROM ORDS (DISK) TO FILETP (TAPE) [T];
    IF T(TASKVALUE) EQL 0 THEN
      REMOVE FILEA;
    END;
```

The value of LASTRECORD is the zero-relative sequence number of the last line of the file. Therefore, FILEA (LASTRECORD) + 1 returns the number of records in the file. Before the disk file FILEA is removed, the TASKVALUE of the copy task is checked to ensure that the copy was successful.

The security characteristics of a file can be inquired about through the SECURITYUSE, SECURITYTYPE, and SECURITYGUARD attributes, each of which returns a string value. For example:

```
IF FILEA (SECURITYUSE) NEQ "IO" THEN
  SECURITY #FILEA(TITLE) IO;
```

This statement can be used to ensure that a file enables both input and output.

Nonresident Files

Statements can be initiated from different interactive sources such as MARC, CANDE, or the ODT as well as from programming languages such as WFL. If one or more of the necessary files are not present, the statement behavior can differ depending on how the statement is initiated. For this reason, not all aspects of statement behavior are documented.

To determine the commands that are acceptable when a NO FILE message is encountered, use the Y (Status Interrogate) system command. For more information about this command, refer to the *System Commands Reference*.

Library Equation

<library equation>

— LIBRARY — <intname> —————→
▶ ([* ,] <library attribute assignment>) —————|

<library attribute assignment>

— <library attribute> — = — <library attribute value> —————|

Explanation

Library equations can be used to change the attributes of libraries used by programs.

The intname in a library equation specifies the internal name used to identify a library.

The library attribute assignment assigns values to the attributes of libraries. For a list of valid library attributes and library attribute values, refer to the *Task Management Programming Guide*.

Examples

The following is an example of a library equation used after a COMPILE statement:

```
COMPILE OBJECT/PROG1 WITH ALGOL LIBRARY;  
COMPILER FILE CARD = PROG1 ON DISK;  
LIBRARY LIB1 (TITLE = OBJECT/LIB1, LIBACCESS = BYTITLE);
```

The following is an example of a library equation following a RUN statement:

```
RUN OBJECT/PROG2;  
LIBRARY LIB2 (TITLE = OBJECT/LIB2, LIBACCESS = BYTITLE);
```

Overriding WFL Library Equations

It is possible for a task to override the WFL library equations that are assigned to it. WFL library equations are merged only with those attributes specified in the library declaration in the task. Library attribute assignments made later in the task override WFL library equations.

Resolving Repeated Library Equations to the Same Library

When a given task equation list, compiler task equation list, task declaration, or task assignment statement includes two or more library equations that apply to the same internal library, the WFL compiler uses the following rules to decide which of the specified library attribute assignments to use. These rules differ according to whether the inname used is an identifier or a string primary.

If the inname used is an identifier, then the following rules apply:

- If a library equation includes an asterisk (*) before the library attribute assignments, then its library attribute assignments are merged with any that have been specified for the library in a previous library equation.
- If a library equation does not include the asterisk, all task attribute assignments specified in previous library equations are discarded, and only the ones given in the latest library equation are used.

If the inname used is a string primary, then the following rules apply:

- The library equation cannot contain an asterisk; if it does, a syntax error is given.
- If the string primary evaluates to the same internal library name as was specified in a previous task equation, a run-time error is given.

Examples

In the following example, L1 is an identifier that specifies an inname. Because the second library equation contains no asterisk, the first library equation is discarded, and the value specified for the TITLE attribute is not used.

```
RUN (TEST)OBJECT/CALC;  
  LIBRARY L1(TITLE = TEST/L1);  
  LIBRARY L1(LIBPARAMETER = "TEST1");
```

In the following example, L1 is still an identifier that specifies an inname. However, because the second library equation includes the asterisk, the attributes from both library equations are merged.

```
RUN (TEST)OBJECT/CALC;  
  LIBRARY L1(TITLE = TEST/L1);  
  LIBRARY L1(*, LIBPARAMETER = "TEST1");
```

Database Equation

<database equation>

— DATABASE —<database name>—————→
└ (— TITLE — = —<database title>—) —————→
 └ = —<database title>—————→

<database name>

└ <name constant> —————→
 └ # — <primary string> —————→
└ <identifier> —————→
 └ # — <primary string> —————→

<database title>

— <file title> —————→

Explanation

A database equation causes a program to use a different database than it would normally.

The database name is the name by which the database is referred to in the program.

Only the TITLE attribute of the database can be equated.

Example

The following is an example of a database equation following a RUN statement:

```
RUN OBJECT/UPDATE;  
  DATABASE TESTDB(TITLE=MYDB);
```

Local Data Specifications

<local data specification>

└ DATA —————→
 └ EBCDIC └ <file name constant> —————→
└ <data images> —————→
└ <i> —————→

Explanation

A local data specification supplies input data in the form of card images to a particular task.

The task reads from the local data specification as if it were an input file. A task that attempts to read from a card reader file will automatically read from a local data specification instead, if one is available. Tasks that read from other kinds of files can be file-equated to cause them to read from a local data specification instead.

Note: The default MAXRECSIZE of a READER file is 14 words (84 characters), but a record contains only 80 characters of valid data. Take this into consideration when using a local data specification, since a record in a file generated by CANDE with a FILEKIND = JOBSYMBOL contains data in columns 1 through 80, spaces in columns 81 and 82, and the sequence number in columns 83 through 90. To avoid getting unwanted information, equate UNITS to CHARACTERS and set the MAXRECSIZE to 80 when reading the local data specification.

The data images are records of EBCDIC data. DATA is a synonym for EBCDIC. The <i> construct that terminates the local data specification also separates the data specification from the next statement; it is not necessary to follow the data specification with a semicolon (;). For more information, see [Global Data Specifications](#).

A local data specification differs from a global data specification in that it can only be used by a single task. The syntax for a local data specification is the same as that for a global data specification, with the following exceptions:

- Local data specifications appear immediately after the task initiation statement for the task that it is going to read from. Only task attributes and file equations can be used between a task and its local data specifications.
- Local data specifications can be included in subroutines, while global data specifications cannot.
- Local data specifications do not have to include a file name; the file name is optional.

When a task tries to open a card reader file, it searches among the local data specifications associated with that task for the first unread local data specification with the correct file name or no file name.

Examples

The following example shows the simplest use of a local data specification. The program (WALLY)OBJECT/COUNTUP expects to read data from a single card reader file. In this situation, the local data specification does not need to be named, and no file equations are required.

```
RUN (WALLY)OBJECT/COUNTUP ;  
DATA  
  6  
  ? % End of data
```

It is a good idea to give each local data specification a title if more than one local data specification is being used by the task. This makes it obvious which local data specification is being substituted for which input file. The local data specification should have the same title as the input file it is replacing in the program, unless the input file has been file-equated to a different title.

In the following example, the program reads the local data specification titled TERMIN1 instead of the input file of the same name, and reads the local data specification titled READDAT instead of the input file titled TERMIN2:

Task Initiation

```
RUN (WALLY)OBJECT/COUNTTWO;
  FILE TERMIN1(KIND=READER);
  FILE TERMIN2(TITLE=READDAT,KIND=READER);
DATA TERMIN1
  3
  128
? % End of TERMIN1 data
DATA READDAT
  5
? % End of READDAT data
```

When used after a COMPILE or BIND statement, local data specifications can be interpreted as input to the compiler or to the program that is compiled depending on the syntax used. Refer to [Compiler Task Equation List](#) for more information.

By default, a compiler expects to find the program source in a card reader file named CARD. For this reason, it is not necessary to include a file equation telling the compiler to read from the appropriate local data specification. It is sufficient to assign CARD as the title of the local data specification, and precede the local data specification with the word COMPILER or a compiler name, as in the following example.

A data specification can appear only in jobs stored in disk files that are initiated by START.

```
COMPILE OBJECT/X WITH ALGOL GO;
  COMPILER PRINTLIMIT=1000;
  PRINTLIMIT=2000;
COMPILER DATA CARD      % Beginning of data for compiler
.
.                        % These lines contain an ALGOL program.
.
?                        % End of compiler data
DATA                    % Beginning of program data
.
.
.
?                        % End of program data
```

The form of the <name> construct that allows 17 EBCDIC characters other than quotation marks (") is not supported.

Section 6

Statements

Overview

The following section describes

- Functional groups of statements
- Statements in each group
- Each statement in alphabetical order

Many statements enable you to specify values for file attributes or task attributes. However, the descriptions of file and task attributes covered in this section are discussed only in terms of their use through WFL. If you require further information, refer to the *File Attributes Programming Reference Manual* for file attribute descriptions, and the *Task Attributes Programming Reference Manual* for task attribute descriptions.

Syntax Diagrams

The syntax diagrams for each statement often contain some elements that are explained in [Section 7, Expressions](#), and [Section 8, Basic Constructs](#). Any statement can be used wherever <statement> appears within a railroad syntax diagram.

Nested Statements

Statements in a job can be nested to a maximum depth of 19 levels. Examples of nested statements include

- A statement specified in an ON statement
- An IF statement within an IF statement
- Statements within a compound statement

WFL Statement Groupings

The following text describes each group of WFL statements and outlines the general functions of each group and the statements within each group.

Null

Takes no action. This statement is sometimes useful in task control or nested flowofcontrol statements. The null statement is the only statement in this group.

Assignment

Assigns values to declared variables. The assignment statement is the only statement in this group.

Compound

Groups other statements together so they can be included in the flow-of-control statements. The compound statement is the only statement in this group.

Flow-of-Control

Controls the order in which statements are executed, or causes other statements to be executed only if the specified conditions are met. Flow-of-control statements include:

- CASE
- DO
- GO
- IF
- WHILE

Subroutine Control

Invokes a subroutine, or causes it to be exited early. Subroutine control statements include:

- <subroutine invocation statement>
- RETURN

Task Control

Affects task execution; for example, by discontinuing, delaying, or rerunning a task. Task control statements include:

- ABORT
- INITIALIZE
- ON
- RERUN
- STOP
- WAIT

Task Initiation

Initiates a task, which is a process that runs in its own stack. Most task initiation statements enable the use of file equations and task attribute assignments to control the execution of the task. Refer to [Section 5, Task Initiation](#), for details. Task initiation statements include:

- ADD
- ARCHIVE DIFFERENTIAL
- ARCHIVE FULL
- ARCHIVE INCREMENTAL
- ARCHIVE MERGE
- ARCHIVE RESTORE
- ARCHIVE RESTOREADD
- ARCHIVE ROLLOUT
- BIND
- COMPILE
- COPY
- LOG
- PB
- PROCESS
- RESTORE
- RUN
- START
- WRAP
- UNWRAP

Task Security

Affects the security privileges of the job. A related statement is the SECURITY statement, which sets the security properties of files. Task security statements include:

- ACCESS
- PASSWORD
- USER
- VOLUME

Communication

Provides the user or the operator with information about the job. Communication statements include:

- INSTRUCTION
- DISPLAY

File Handling

Opens or closes files. There are several ways of closing files, each of which leaves the file in a different state. Refer to [File Handling](#) for more information. File handling statements include:

Statements

- CRUNCH
- LOCK
- OPEN
- PURGE
- RELEASE
- REWIND

File Management

Changes, removes, or prints disk files, and creates permanent directories. Changes file titles or security. The MODIFY statement permanently adds to or changes the attributes in an object code file. File management statements include:

- ALTER
- ARCHIVE PURGE
- ARCHIVE RELEASE
- CHANGE
- MKDIR
- MODIFY
- PRINT
- REMOVE
- SECURITY
- VOLUME

Note: The ADD, COPY, MOVE, RESTORE, and RESTOREADD statements are related to the file management statements, but are considered task initiation statements because they initiate a task when copying files.

Cataloging

Affects the cataloging of information about files, disk families, or tapes. Cataloging statements include:

- CATALOG
- VOLUME

ABORT Statement

<abort statement>

— ABORT — [— <task identifier> —] [<string expression>]

Explanation

The ABORT statement

- Discontinues a task, or a job, and any tasks initiated by the job.
- Discontinues a task associated with the task identifier specified in the ABORT statement.
- Discontinues a job and all the tasks initiated by a job if a task identifier is not specified in the ABORT statement.

- Displays up to a maximum of 430 characters of the value of the string expression prior to the abort, if the string expression is specified in the ABORT statement.
- Accepts up to 1799 characters.

Examples

The following are examples of the ABORT statement:

```
ABORT;

ABORT "THIS JOB HAS BEEN ABORTED";

IF T(TASKVALUE)=5 THEN ABORT;
```

In the following example, if the TASKVALUE attribute of task T2 has the value 3, the task associated with the task identifier T1 is discontinued after the message "TASK ABORTED" is displayed:

```
IF T2(TASKVALUE) = 3 THEN ABORT[T1] "TASK ABORTED";
```

In the following example, if the task state of the task T is not COMPILEDOK, the job is discontinued, or in the case of an asynchronous subroutine, the subroutine is discontinued:

```
IF T ISNT COMPILEDOK THEN ABORT[MYSELF];
```

ACCESS Statement

<access statement>

```
— ACCESS PASSWORD — = — <new accesscode password> —————|
```

<new accesscode password>

```
— <name constant> —————|
```

Explanation

The ACCESS statement changes the accesscode password of the current accesscode of the job. The password is changed in the USERDATAFILE. The job must have a valid accesscode.

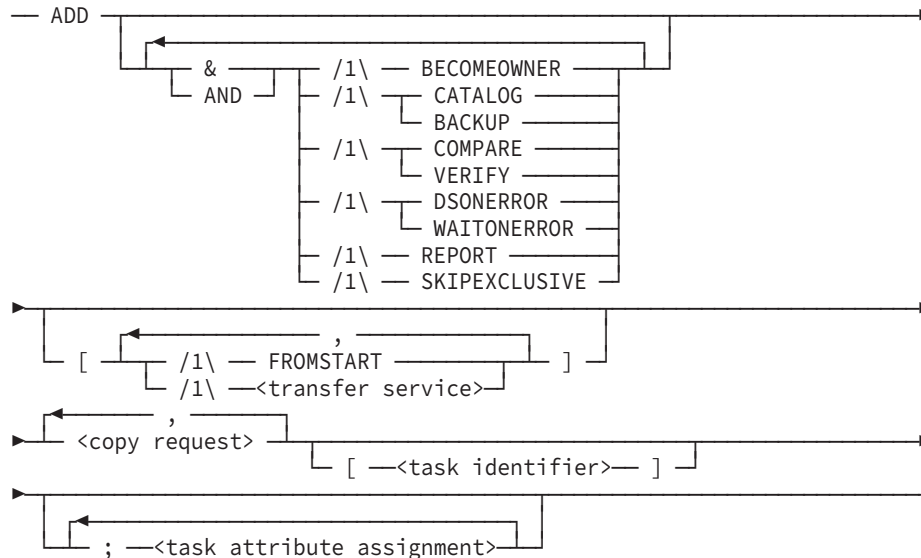
Example

The following is an example of the ACCESS statement:

```
ACCESS PASSWORD = ENTER
```

ADD Statement

<add statement>



Explanation

The ADD statement is similar to the COPY statement. It copies files between disks and tapes. It is particularly useful for adding a directory of files to a disk where some of the files are already resident and are to be preserved.

The ADD statement has the following effects which depend on whether a disk or tape destination is specified:

- For a disk destination, the ADD statement copies only those files that are not already resident on the specified disk destination.
- For a tape destination, it is equivalent to a COPY statement with a tape destination. If there were any files on the destination tape, they will be overwritten. The ADD statement will copy all the available requested files to the destination tape.

For a detailed explanation of the ADD statement syntax, refer to the [COPY or ADD Statement](#).

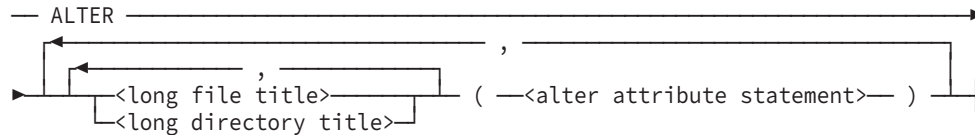
Example

The following example of the ADD statement copies files under the directory Z/= from tape T to disk R and to DISK. Any files already resident on the destination volumes are not copied. Different files might be copied to R and DISK, depending on what is already resident on each destination volume before the ADD is executed.

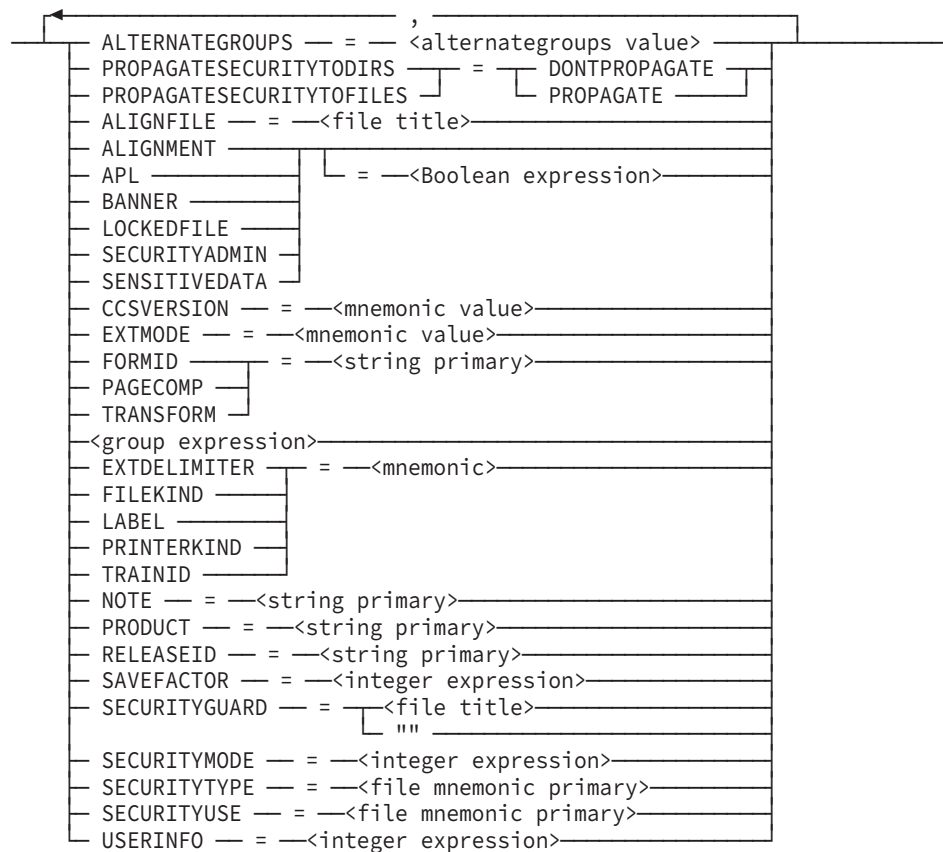
```
ADD Z/= FROM T(KIND=TAPE) TO R(KIND=DISK), TO DISK;
```

ALTER Statement

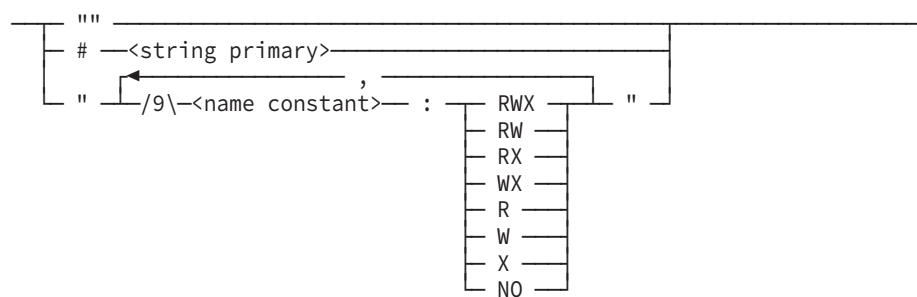
<alter statement>



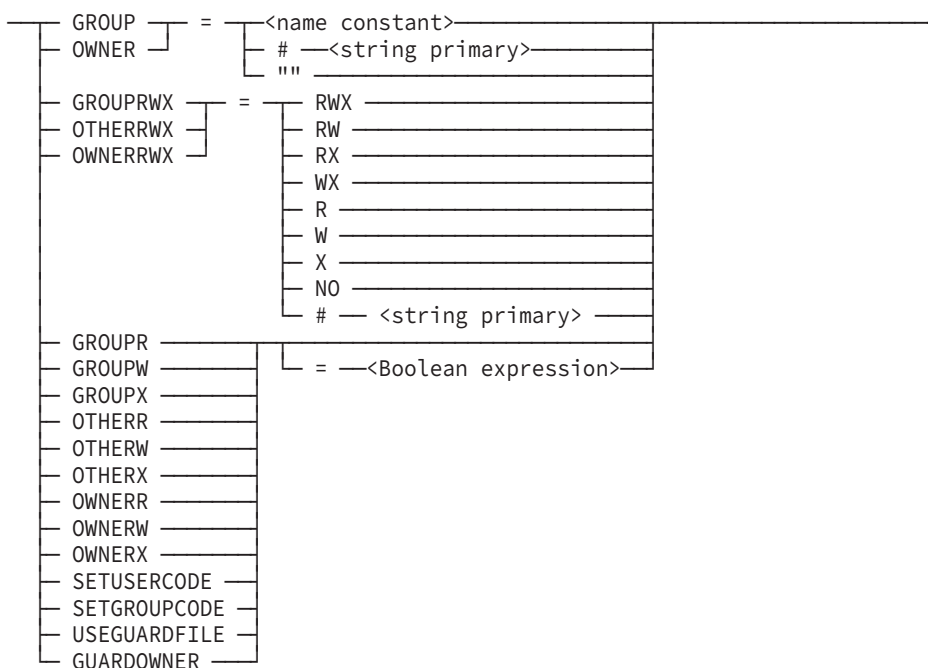
<alter attribute list>



<alternategroups value>



<group expression>



Explanation

The ALTER statement changes the file attributes of a disk file.

When ALTER encounters a file that is currently open with EXCLUSIVE = TRUE, ALTER enters a waiting state. To skip the file, enter <mixnumber> OF. To exit without processing any more files, enter <mixnumber> DS.

The following file attributes are available for the <alter attribute statement>. Refer to the *File Attributes Programming Reference Manual* for more information about each attribute.

Note: Many security attributes are interrelated; therefore changes to one attribute might affect another attribute.

ALIGNFILE

Specifies the name of a printer backup file which contains an alignment pattern for a particular form. This attribute is used by the Print System when the ALIGNMENT attribute is TRUE.

ALIGNMENT

When set to TRUE, the Print System performs alignment before printing the file.

ALTERNATEGROUPS

Specifies a list of up to nine groups whose members can access the file in the manner defined by the corresponding set of permissions. Any process executing a task with a

GROUPCODE or SUPPLEMENTARYGROUPS that matches the GROUP attribute of the file or matches one of the groups specified in the ALTERNATEGROUPS attribute—but is not the owner of the file—is granted the corresponding access permissions.

A process matching to multiple groups receives the union of the permissions granted for the individual matches. If the ALTERNATEGROUPS attribute is not set, then it has no effect on file access.

APL

When set to TRUE, specifies that only a program whose code file also has its APL attribute set to TRUE can access the file. If the APL attribute is set to FALSE, the file can be accessed by a program regardless of the APL attribute value in the program code file.

Note: *This file attribute can be modified only if the ALTER statement is executed from the ODT.*

BANNER

When set to TRUE, a banner page is printed ahead of the desired file to help identify the printed output.

CCSVERSION

Specifies the rules to be used for processing the character data in the file.

When you specify CCSVERSION, it is modified in the file and the EXTMODE attribute is modified to a value compatible with the CCSVERSION specified.

If you specify both CCSVERSION and EXTMODE, and they are compatible with one another, then both attributes are modified in the file. Incompatible values cause ALTER to fail and an error message results.

For character-oriented files, if modifying the value of CCSVERSION or EXTMODE causes the character size to change, ALTER fails and an error message results.

EXTDELIMITER

Specifies the delimiter characters, if any, for identifying the separation of records in the file. EXTDELIMITER is meaningful primarily for FILECLASS=CHARACTERSTREAM files.

EXTMODE

Specifies the external or physical character encoding of the records in the file.

When you specify EXTMODE, it is modified in the file. The CCSVERSION attribute is unaffected if it is compatible with the EXTMODE specified. Otherwise, CCSVERSION reflects a value of NOTSET (–1) in the file.

If you specify both `EXTMODE` and `CCSVERSION`, and they are compatible with one another, then both attributes are modified in the file. Incompatible values cause `ALTER` to fail and an error message results.

For character-oriented files, if modifying the value of `EXTMODE` or `CCSVERSION` causes the character size to change, `ALTER` fails and an error message results.

FILEKIND

Specifies the internal structure and purpose of the file.

FORMID

When specified, the file can be printed only on a device with a matching identification. This attribute is normally applied to indicate the kind of paper to be used.

GROUP

Specifies a group whose members can access the file in the manner defined by the `GROUPLWX` attribute. Any process executing a task with a `GROUPCODE` or `SUPPLEMENTARYGROUPS` that matches the `GROUP` attribute of the file—but is not the owner of the file—is granted the access permissions defined by the `GROUPLWX` attribute. If the `GROUP` attribute is not set, then group access is not granted to any process attempting to access the file.

GROUPL

When set to `TRUE`, grants group members read-access to the file.

GROUPLW

When set to `TRUE`, grants group members write-access to the file.

GROUPLX

When set to `TRUE`, grants group members execute-access to the file.

GROUPLWX

Specifies the manner in which members of the group matching the group attribute of the file are permitted to access the physical file. The following table lists the valid mnemonic values for `GROUPLWX`:

Mnemonic	Meaning
RWX	Read, Write, Execute
RW	Read, Write
RX	Read, Execute

Mnemonic	Meaning
R	Read
WX	Write, Execute
W	Write
X	Execute
NO	None

Family substitution is used if the job or task has an active family specification. Only the primary family name is used. Refer to [FAMILY Assignment](#) and [Interrogating Complex Task Attributes](#).

GUARDOWNER

Used with the USEGUARDFILE attribute to cause the guard file to define access permissions for the owner of the file.

Note: The *GUARDOWNER* attribute has no effect if the *USEGUARDFILE* attribute is reset.

LABEL

For tape files, controls whether labels are written and affects end-of-file action.

For disk and printer files, controls whether banner pages are printed and affects top-of-page action.

LOCKEDFILE

When set to TRUE, prevents disk files from being removed or replaced, and the file name from being changed. However, the locked file can be opened and updated, and the file attributes can be changed. When set to FALSE, this attribute enables files to be removed and changed.

NOTE

Stores a message of up to 250 characters to be printed on the banner page preceding the printer file, punch file, or disk file. The default value is a null string.

OTHERR

When set to TRUE, grants other users (excluding the owner and members of the groups specified by GROUP and ALTERNATEGROUPS) read-access to the file.

OTHERW

When this file attribute is set to TRUE, then other users (excluding the owner and members of the groups specified by GROUP and ALTERNATEGROUPS) are granted write-access to the file.

OTHERX

When set to TRUE, grants other users (excluding the owner and members of the groups specified by GROUP and ALTERNATEGROUPS) execute-access to the file.

OTHERRWX

Specifies the manner in which all other users (excluding the owner and members of the groups specified by GROUP and ALTERNATEGROUPS) are permitted to access the physical file.

The following table lists the valid mnemonic values for OTHERRWX:

Mnemonic	Meaning
RWX	Read, Write, Execute
RW	Read, Write
RX	Read, Execute
R	Read
WX	Write, Execute
W	Write
X	Execute
NO	None

Family substitution is used if the job or task has an active family specification. Only the primary family name is used. Refer to [FAMILY Assignment](#) and [Interrogating Complex Task Attributes](#).

OWNER

Modifies the owner of the file.

Note: This attribute can be set only within a permanent directory namespace.

OWNERR

When set to TRUE, grants the owner readaccess to the file.

OWNERW

When set to TRUE, grants the owner writeaccess to the file.

OWNERX

When set to TRUE, grants the owner execute access to the file.

OWNERRWX

Specifies the manner in which the owner of the file is permitted to access the physical file. The following table lists the valid mnemonic values for OWNERRWX:

Mnemonic	Meaning
RWX	Read, Write, Execute
RW	Read, Write
RX	Read, Execute
R	Read
WX	Write, Execute
W	Write
X	Execute
NO	None

Family substitution is used if the job or task has an active family specification. Only the primary family name is used. Refer to [FAMILY Assignment](#) and [Interrogating Complex Task Attributes](#).

PAGECOMP

Specifies formatting options to be used when printing a file.

PRINTERKIND

Specifies the kind of device the file is to be printed on.

PRODUCT

Stores a message of up to 250 characters to specify or determine that a piece of your software belongs to a product group. This attribute associates a piece of software with a specific application, product, and component as defined within the Unisys tracking and reporting system.

PROPAGATESECURITYTODIRS

When set to PROPAGATE on a permanent directory, causes subdirectories within the permanent directory—for which no security attributes have been explicitly set—to be assigned the same security attributes as the parent directory.

PROPAGATESECURITYTOFILES

When set to PROPAGATE on a permanent directory, causes files within the permanent directory—for which no security attributes have been explicitly set—to be assigned the same security attributes as the parent directory.

RELEASEID

Specifies or determines the release level of the file.

SECURITYADMIN

When set to TRUE, restricts access to the file to only those processes marked with the USERDATA granulated privilege. The USERDATA privilege is automatically given to users and processes marked with SECADMIN privilege, if security administrator status is authorized and to privileged users otherwise. This attribute can be set only by processes marked with the USERDATA privilege.

SAVEFACTOR

Indicates the expiration date of a file in terms of the number of days past the creation date.

SECURITYGUARD

Identifies the guard file to be invoked for the file if the SECURITYTYPE attribute is assigned GUARDED or CONTROLLED. For more information about guard files, refer to the *Security Overview and Implementation Guide*.

SECURITYMODE

Specifies the manner in which users are permitted to access the physical file, including the owner of the file.

SECURITYTYPE

Provides access control over users, other than the owner of a file, to a physical file. The SECURITYTYPE attribute can have a value of PRIVATE (default), PUBLIC, GUARDED, or CONTROLLED:

- PRIVATE files can be accessed or overwritten only by their owners and privileged users.
- PUBLIC files can be accessed by tasks with any usercode, as limited by the setting of the SECURITYUSE attribute.
- GUARDED files can be accessed by the owner, however, nonprivileged users and programs are granted access as defined by the guard file. The guard file, which defines the access rights to files, must be examined before access to a disk file is granted.
- CONTROLLED files can be accessed after the guard file is examined and access to your disk file is granted. If you are not defined in the guard file, you do not have access to the file.

SECURITYUSE

Specifies how a physical file that is protected by security can be accessed by nonprivileged users using nonprivileged programs. This attribute can have a value of IO (default), IN, or OUT. When a PUBLIC file is accessed by a task with a usercode that differs from the OWNER, the SECURITYUSE attribute permits the following actions based on its value:

- A value of IO permits reading, writing, overwriting, and purging.
- A value of IN permits reading, but not writing, overwriting, or purging.
- A value of OUT permits writing, overwriting, or purging, but not reading.

SENSITIVEDATA

When set to TRUE, causes the disk or pack areas assigned for a file to be overwritten with an arbitrary pattern before the disk space is returned to the system for reallocation.

SETGROUPCODE

When set to TRUE on a code file or job symbol, the program or job executes with an effective GROUPCODE of the group of the file.

SETUSERCODE

When set to TRUE on a code file or job symbol, the program or job executes with an effective USERCODE of the owner of the file.

TRAINID

Specifies the print train to be used on a train printer.

TRANSFORM

Specifies a transform function, which is used to manipulate the data before printing.

USEGUARDFILE

When set to TRUE, a guard file in addition to the SECURITYMODE attribute controls access to the physical file. For the guard file to control access to the file completely, the file access permission flags OWNERRWX, GROUPRWX, and OTHERRWX must be set to TRUE.

USERINFO

Saves site or application-specific information.

Examples

This ALTER statement prevents a file named FILEX from being removed or replaced, and the file name from being changed.

```
ALTER FILEX (LOCKEDFILE)
```

This statement locks the files called MYFILE and AFILE so that the files cannot be replaced or removed, and sets their SECURITYUSE attribute to permit reading only.

```
ALTER MYFILE, *SOURCE/AFILE (LOCKEDFILE=TRUE, SECURITYUSE=IN)
```

The first line of this statement locks the file called FILEY and all files in the directory called MYFILE, and changes the SENSITIVEDATA attribute for those files. The second line of this statement sets the expiration date of FILEX to 30 days past its creation, sets the NOTE attribute of FILEX to "Banner Page", and locks the file.

```
ALTER FILEY, MYFILE/= (SENSITIVEDATA, LOCKEDFILE),  
FILEX (SAVEFACTOR=30, NOTE="Banner Page", LOCKEDFILE)
```

Archive Subsystem

Note: *Non-privileged users cannot use ARCHIVE statements for files within the permanent directory namespace.*

The archive subsystem consists of six different ARCHIVE statements. These statements enable you to

- Copy files to archive backup tape and CD-ROM volumes.
- Maintain a record of the names, locations, and attributes of the archived files and directories.
- Remove archive backup information for specified files.
- Transfer archived files between backup tapes.
- Restore archived files to disk.
- Merge files onto a single tape or tape set.

The following paragraphs briefly describe the functions of the ARCHIVE statements:

- Copying, transferring, and restoring files

You can perform library maintenance operations that include copying and transferring disk files to backup tape and CD-ROM volumes, restoring disk files from backup tape and CD-ROM volumes, and merging backup tape and CD-ROM volumes to a single tape or tape set.

- Automatically maintaining an archive directory

The archive directory is a disk directory that records the names, locations, and attributes of disk files that have been transferred through archive operations to tape or CD-ROM volumes, or merged from many tape and CD-ROM volumes to one tape or tape set. The archive subsystem maintains an archive directory for each online disk family from which archive operations have been performed. These directories reside on the DL CATALOG family.

- Controlling access using the support library

The support library is used by the ARCHIVE statement to control which files are copied during an archive operation. The archive support library rejects files that are

requested by ARCHIVE statements based on various selection criteria. The support library is sometimes called the selector library.

Specifying Different ARCHIVE Statements

The following table lists all of the ARCHIVE statements and their functions.

At many sites, the use of the ARCHIVE statements to manipulate files under usercodes other than your own is limited to those usercodes whose security access is privileged. However, a WFL job that is started at an ODT without a usercode can use any one of the following ARCHIVE RESTORE statements.

ARCHIVE DIFFERENTIAL ARCHIVE FULL ARCHIVE INCREMENTAL

Copy resident disk files to library maintenance tape or CD-ROM volumes and are referred to as ARCHIVE backup statements.

ARCHIVE MERGE

Merges files from two or more library maintenance tape or CD-ROM volumes to a single tape or tape set.

ARCHIVE PURGE

Removes the backup records for specific files or directories from the archive directory of the specified disk family. This statement does not affect resident disk files in any way.

ARCHIVE RELEASE

Removes files from disk that are not in use and have up-to-date archive backup records. This statement is intended to free disk space for other uses. The removed files can be restored by the archive AUTORESTORE feature and the WFL statements ARCHIVE RESTORE and ARCHIVE RESTOREADD. In addition to the ARCHIVE RELEASE statement, see the WFL statements REMOVE and REMOVE DESTROY.

ARCHIVE RESTORE ARCHIVE RESTOREADD

Restore archived files from library maintenance tape and CD-ROM volumes to disk.

ARCHIVE ROLLOUT

Selects and copies disk files to a library maintenance tape and CD-ROM volumes. It then removes the original disk files from the disk. This statement is intended to free disk space for other uses.

Generations

The archive subsystem stores information about backups in the SYSTEM/ARCHIVE/<family name>/<number> directory for a family. The archive subsystem can store information about one to four different backups of the same

generation of any given disk file. The archive subsystem does not store information about different generations of the same file. When you execute an ARCHIVE backup, if the resident generation of a file is different from the generation listed in the archive directory, the archive subsystem deletes all the information about the old generation of the file. It deletes information including the backup tape, CD serial numbers and volume names, and stores the information about the new generation of the file and its backup tape(s) or CD into the ARCHIVE directory.

The archive subsystem calculates the generation of a file based on several attributes. The PD system command displays most of the attributes that the archive subsystem uses to calculate the generation of a file:

```
ARCHIVE ENTRY 0 (filekind):
  CYCLE: number VERSION: number
    TIMESTAMP:      date and time
    CREATION:        date and time
    LASTACCESS:      date and time
    ALTER/MODIFY:    date and time
```

The calculation does not use the LASTACCESS date or time. "ALTER/MODIFY" is the more recent of either the ALTERDATE and ALTERMTIME or the ATTMODIFYDATE and ATTMODIFYTIME. For example,

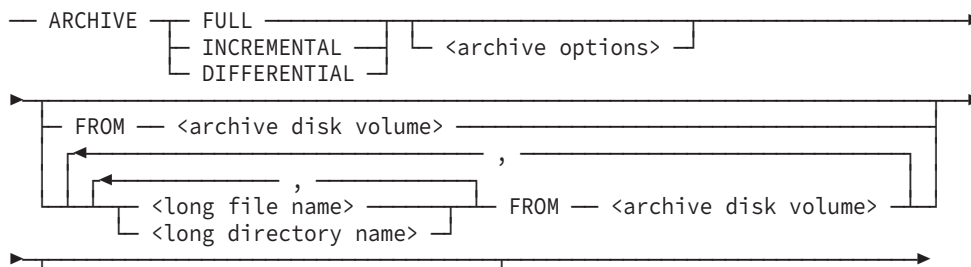
- ARCHIVE INCREMENTAL does not copy any resident file whose generation matches the generation of its existing backup. ARCHIVE INCREMENTAL deletes all existing backup information about any file it copies and stores the information about the resident version of the file, which it copied into the ARCHIVE directory.
- If ARCHIVE FULL of a file is done with existing backups and if the FILEKIND of the resident file differs from that of the backups, or if the ALTERDATE or ALTERMTIME of the resident file differs from that of the backups, then the archive subsystem erases all the information about the old backups and replaces it with that of the resident file's new backup.

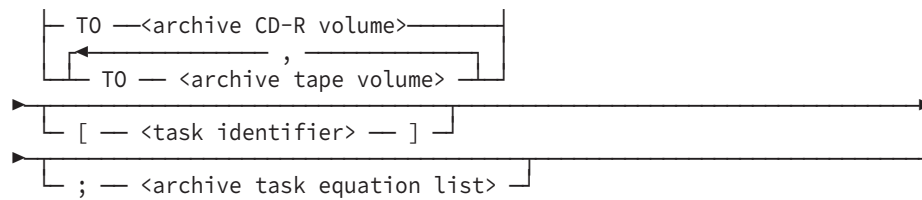
FAMILY Specifications

Family substitution is used if the job or task has an active family specification. Only the primary family name is used. Refer to [FAMILY Assignment](#) and [Interrogating Complex Task Attributes](#).

ARCHIVE Backup Statement

<archive backup statement>





Explanation

These ARCHIVE statement copies resident disk files to backup tape or CD-ROM volumes. You can use these statements to backup some or all of the disk files on various families; which files the archive subsystem selects for backup depends on which of the three statements you select.

The three variants of the ARCHIVE backup statement are as follows:

- **ARCHIVE FULL**
Copies all specified resident disk files on the named families to a CD-ROM volume or to one or more backup tapes.
- **ARCHIVE INCREMENTAL**
Copies to backup CD-ROM or tape volumes only those specified resident disk files that have been changed or added since the last archive backup procedure was performed. This statement does not copy files solely on the basis of the last full archive operation.
- **ARCHIVE DIFFERENTIAL**
Copies to backup CD-ROM or tape volumes the specified resident disk files that have been changed or added to the family or families since the last ARCHIVE FULL statement was executed.

For each of the ARCHIVE backup statements, files are selected by the disk subsystem and then accepted or rejected for the actual archive procedure. The archive support library determines which of the candidate files can and cannot be archived to a tape or CD-ROM volume.

If your installation has compiled its own selector support library, you can use library equation to direct the file selection process through that library.

You can use the backup CD-ROM and tape files created by these statements as input for the ARCHIVE MERGE, ARCHIVE RESTORE, and ARCHIVE RESTOREADD statements, and for the COPY and ADD statements.

Special Considerations for Backup Tape and CD-ROM Volumes

It is necessary to plan what volume names and, optionally, what serial numbers to use for the backup volume. Otherwise, archive restore requests result in the following RSVP messages for tape or CD-ROM volumes it decides to use:

```

NO FILE <volume name>/FILE000 (MT) [<serial number>]
NO FAMILY <volume name> (CD) [<serial number>]
  
```

You need to select scratch tapes to use in the archiving procedure from your tape library. It is recommended that you label the scratch tapes used in your archiving procedures with an unchanging serial number clearly visible on the tape exterior. Always use archive scratch tapes in numeric sequence.

The operator must have a method to find those tape or CD-ROM volumes and load them onto a tape or CD-ROM unit. Keep in mind the following considerations:

- Tape volumes always have serial numbers that you assign in advance with the SN (Serial Number) system command. Since the serial numbers for tape volumes are usually unique and assigned in numerical sequence, it is relatively easy to build a tape library with each volume stored according to its serial number.
- For CD-ROM volumes, you cannot assign the serial numbers in advance, and when you do specify a serial number in an archive backup request, the archive system puts the same serial numbers on all the CD-ROM volumes created by that backup request.

Therefore, you must systematically assign volume names and, optionally, serial numbers to CD-ROM volumes for each archive backup request you issue and store those CD-ROM volumes in an organized library so that operators can locate them when archive restore asks for them.

Checking the Progress of the Backup

You can use the HI (Cause Exception Event) system command to check the progress of an ARCHIVE backup statement. A command of the form <mix number> HI displays the number of files already copied and other information.

Examples

The following example illustrates how to perform a complete backup of the TESTPACK family:

```
ARCHIVE FULL FROM TESTPACK
```

The following example shows how to perform a complete backup of all online families:

```
ARCHIVE FULL
```

The following example creates backup copies of the files under the SYSTEM directory on the families DISK and HLUNIT that do not already have archive backup copies:

```
ARCHIVE INCREMENTAL & VERIFY SYSTEM/= FROM DISK,  
SYSTEM/= FROM HLUNIT TO SYSTAPE;
```

The following example illustrates how to perform a backup of files under the usercode DOE on the family TESTFAMILY that have been updated since the last ARCHIVE FULL statement:

```
ARCHIVE DIFFERENTIAL (DOE) = FROM TESTFAMILY
```

The following example shows how to back up files under the usercode MYCODE to a tape volume called NEWTAPE from the scratch pool POOL1:

```
ARCHIVE DIFFERENTIAL (MYCODE) = FROM TESTFAMILY
  TO NEWTAPE (KIND = TAPE, SCRATCHPOOL = POOL1);
```

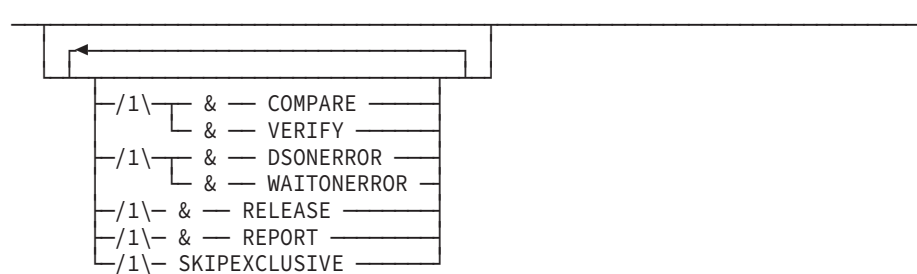
The following example shows how to perform a complete backup of the XDATA disk family to CD-ROM volumes. In this case, because CDCOPIES is 2, the system produces duplicate copies of the files onto two sets of CD-ROM volumes.

```
ARCHIVE FULL *= FROM XDATA
  TO XDARC (KIND=CD, CDCOPIES=2, SERIALNO=555555);
```

In this example, both sets of CD-ROM volumes get the name XDARC and the serial number 555555. If more files are on the disk family XDATA than fit on one CDROM volume, then the system requests additional CD-ROM volumes during the copy process. The system gives those additional CD-ROM volumes the volume name XDARC and the serial number 555555 also.

ARCHIVE Statement Options

<archive options>



Explanation

The following table options are available for all of the ARCHIVE statements.

COMPARE

Compares the input file and the output file bit by bit immediately after the file is copied. If a compare error occurs, the process will stop and ask the operator if the file must be recopied.

DSONERROR

Causes the system to discontinue the archiving process if any error is detected.

RELEASE

Causes library maintenance to execute an “archive release” operation for each file that library maintenance successfully copies and archives. The archive release operation removes the resident version of the file from the disk. You can use the RELEASE option only in ARCHIVE FULL, ARCHIVE INCREMENTAL, and ARCHIVE DIFFERENTIAL statements.

Note: Library maintenance does not release a file that is in use by another program.

REPORT

Causes library maintenance to print a report of the files it copied and any errors encountered. When & REPORT is specified, library maintenance does not write “file copied” messages in the job log or the system sumlog.

SKIPEXCLUSIVE

Causes the system to not copy those files from disk that are opened with EXCLUSIVE=TRUE or that are KEYEDIOII files marked as being updated.

WAITONERROR

Causes the archive process to issue an RSVP message whenever an error occurs during the archive process.

Examples of possible errors include: requesting a file or directory that is missing, or failing to open a tape successfully. The RSVP message halts the archive process until the operator or programmer responds with OK or DS. A response of OK causes the archive process to continue archiving with other files or tapes. A response of DS will terminate the archive process. After investigating the error which created the RSVP message, you can re-issue the archive statement.

VERIFY

This option is similar to the COMPARE option. However, instead of comparing the copied file bit by bit, the file is read again and its overall checksum is compared. When copying to a CD-ROM volume, the system ignores the VERIFY option.

ARCHIVE Disk Volume

<archive disk volume>

— <family name> —┐
 └ <archive disk volume attribute list> ┘

Explanation

The archive disk volume syntax designates the disk that contains the files where an archive function is going to be performed.

The archive functions that you can perform on the source disk include:

- Archiving files
- Restoring files
- Releasing files
- Purging archive information for files from a specific archive directory

Examples

The following example causes the system to generate a tape name of the form INCREMENTAL95302:

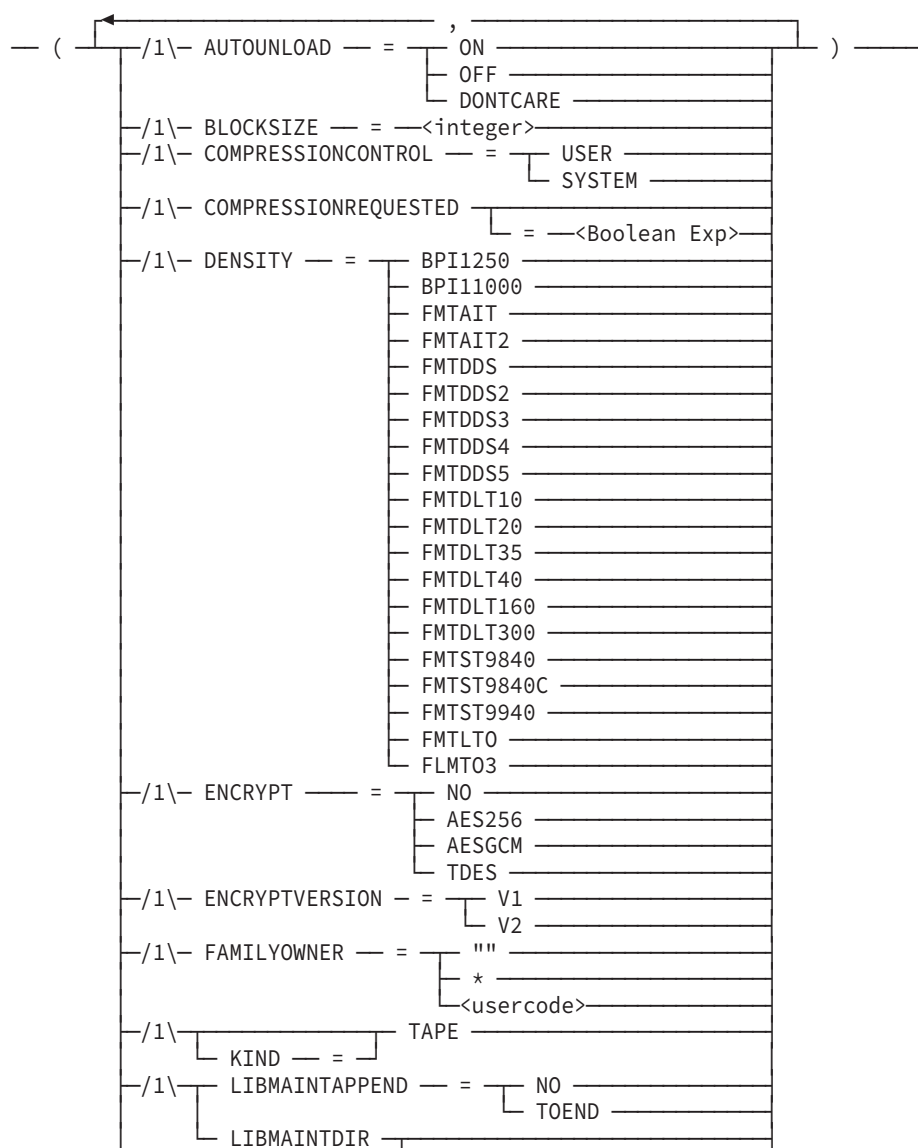
```
ARCHIVE INCREMENTAL = FROM DISK, = FROM HLUNIT;
```

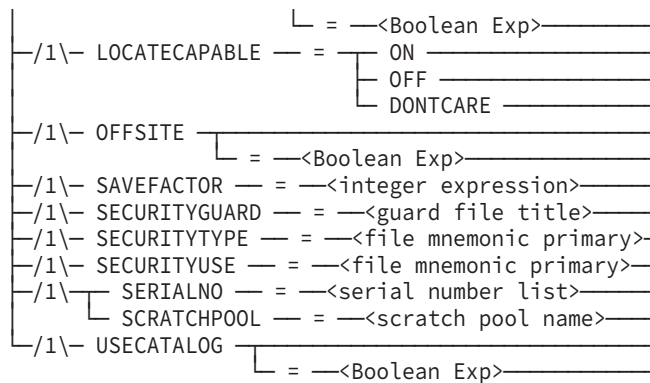
The following example causes the system to generate a tape name of the form DISK95302:

```
ARCHIVE FULL & COMPARE = FROM DISK;
```

ARCHIVE Tape Volume Attribute List

<archive tape volume attribute list>





<scratch pool name>

— <name>

Attributes

You can specify the following attributes for the archive tape volume attribute list:

- AUTOUNLOAD
- BLOCKSIZE
- COMPRESSIONCONTROL
- COMPRESSIONREQUESTED
- DENSITY
- ENCRYPT
- ENCRYPTVERSION
- FAMILYOWNER
- KIND
- LIBMAINTAPPEND
- LIBMAINTDIR
- LOCATECAPABLE
- OFFSITE
- SAVEFACTOR
- SCRATCHPOOL
- SECURITYGUARD
- SECURITYTYPE
- SECURITYUSE
- SERIALNO
- USECATALOG

AUTOUNLOAD

Determines whether or not a tape is unloaded when it is released by the system during a reel switch or a file close operation. If the value is ON, the tape is rewound and unloaded. If the value is OFF, the tape is not unloaded. If the value is DONT CARE, or if this attribute is not specified, the tape behavior is controlled by the setting of the AUTOUNLOAD option of the MODE (Unit Mode) system command. For more information, refer to the *System Commands Reference*.

BLOCKSIZE

Specifies the tape blocksize in words that library maintenance uses when copying files to and from tape. Library maintenance automatically rounds the value you specify up to an integer multiple of 900 words. If you specify a value larger than 64800 words, library maintenance uses 64800 instead. If you specify a value greater than 4500 words for a tape destination, then library maintenance automatically uses that blocksize for all disks involved in the operation.

Using the same blocksize for the source and all the destinations greatly improves the performance of the copy task. Using a larger blocksize increases the amount of data that can be stored on any given tape volume.

Notes:

- *Certain magnetic tape devices have limits on the maximum length I/O they can process. If you specify a blocksize larger than the limit for a tape device, library maintenance automatically reduces the blocksize to a valid value for that tape destination.*
- *If you do not specify BLOCKSIZE for a tape, library maintenance uses the default value established by the SYSOPS LMBLOCKSIZE system command; or if the default is zero (0), library maintenance uses a small default blocksize that it selects for each tape and disk.*

COMPRESSIONCONTROL

Enables you to control whether compression will be applied to the volume being created.

There are two values associated with this attribute:

- USER
- SYSTEM

When the value of USER is selected, compression will occur based on the value of the COMPRESSIONREQUESTED file attribute. When the value SYSTEM is selected, compression will occur based on the compression value in the tape label. SYSTEM is the default value. For further information, refer to the *File Attributes Programming Reference Manual*.

COMPRESSIONREQUESTED

This attribute only has significance for tape files when the COMPRESSIONCONTROL attribute is set to a value of USER.

- If COMPRESSIONCONTROL = USER and COMPRESSIONREQUESTED = TRUE, compression will occur.
- If COMPRESSIONCONTROL = USER and COMPRESSIONREQUESTED = FALSE, compression will not occur.

Note: Stating *COMPRESSIONREQUESTED* in a WFL statement implies *COMPRESSIONREQUESTED = TRUE*. For further information, refer to the *File Attributes Programming Reference Manual*.

DENSITY

Indicates the recording density of a magnetic tape volume. The default value is the density setting of the tape unit selected.

ENCRYPT

Specifies that library maintenance encrypt the data it writes to the specified destination tape. The following three encryption algorithms are provided:

- TDES
- AES256
- AESGCM

To use AESGCM, Media Encryption Version 2 is required.

If you specify LIBMAINTAPPEND, then library maintenance ignores the ENCRYPT setting and instead uses the original ENCRYPT value for that set of tapes.

The LOADER cannot read encrypted library maintenance tapes.

You cannot specify ENCRYPT for a source tape. Library maintenance automatically decrypts data when reading from an encrypted tape, so you can use a simple COPY statement to copy files from an encrypted tape:

```
COPY ... files ... FROM <tape name>;
```

ENCRYPTVERSION

Specifies the Media Encryption Version to be used. This overrides the value of the SYSOPS LMDEFENCRYPT version.

- V1
- V2

FAMILYOWNER

Indicates the usercode of the owner of a tape volume family. If you do not use the FAMILYOWNER attribute or if you specify a null string (""), the usercode of the archive process is used. If you specify an asterisk (*) with the FAMILYOWNER attribute, the tape volume becomes a nonusercoded volume.

Note: This attribute applies only to installations running the Security Accountability Facility software.

KIND

Specifies the kind of peripheral unit to use. KIND can be TAPE or CD (or CDROM).

LIBMAINTAPPEND

This attribute is used by library maintenance in the copy procedure.

Notes:

- *Library maintenance does not update the tape directories on any reels already copied with the file names of the files being added. Library maintenance only updates the LIBMAINTDIR tape directory disk files with the names of the new files.*
- *The message BACKUP START ON: data and time appears in the PD display for files that have archive backups. This message refers to the time of the original task that started the tape. It does not refer to the start time of the task with LIBMAINTAPPEND = TOEND specified.*

LIBMAINTAPPEND = NO

If you specify LIBMAINTAPPEND = NO, or do not specify LIBMAINTAPPEND, the copy procedure copies to a new tape.

LIBMAINTAPPEND = TOEND

If you specify LIBMAINTAPPEND = TOEND, library maintenance searches for an existing library maintenance tape with the name and serial number you specify. The tape you specify must be a tape created with the LIBMAINTDIR = TRUE specification. Library maintenance opens the LIBMAINTDIR disk file with the EXCLUSIVE file attribute set to TRUE. This prevents other archive, library maintenance, and FILEDATA processes from using that LIBMAINTDIR disk file while library maintenance is updating it. Library maintenance checks the LIBMAINTDIR tape directory disk file for that tape to determine the serial number of the last tape in that set of tapes. (Even if the tape contains no tape directory, it does contain a pointer to the LIBMAINTDIR file on disk, which is used to find the location of the end of the last file on the final tape.)

If necessary, library maintenance searches for that tape. Then library maintenance skips to the end of the last file copied to that tape and copies the files that you specified. The copy procedure expands the LIBMAINTDIR tape directory disk file for the tape with the names and status of all files copied to the LIBMAINTDIR file.

If an error occurs that stops the copy to the tape (for example, if the operator issues a <mix number> DS command to terminate the append task), then all the files successfully copied up to that point are on the tape and are listed in the LIBMAINTDIR file. However, no more files can be appended to the tape. Any attempt to use LIBMAINTAPPEND again to add more files to that tape is rejected with an error message.

The following conditions must be met to use LIBMAINTAPPEND = TOEND:

- The destination tape must have been created by a COPY or ARCHIVE statement that specifies LIBMAINTDIR = TRUE for that tape.
- You must specify & VERIFY for the append operation if the & VERIFY option was specified when the tape was originally created.
- You must not specify & VERIFY for the append operation if the & VERIFY option was not specified when the tape was originally created.

When library maintenance copies files to a tape with LIBMAINTAPPEND = TOEND, it does not add the names of those files to the tape directory for the tape. You cannot use the TDIR system command or the FILEDATA utility TAPEDIR request to view the names of files copied to tape with LIBMAINTAPPEND = TOEND. Instead, use the FILEDATA utility LIBMAINTDIR modifier to view the names of all the files copied to a tape.

If you specify a list of tape serial numbers for the SERIALNO attribute, library maintenance takes special action. Library maintenance uses the first serial number in the list to locate any existing tapes in the set of tapes to which the files are to be added or appended. Library maintenance then reads the LIBMAINTDIR file for that tape to determine the serial number of the last tape volume in the set.

Library maintenance immediately opens that tape, if necessary, then uses the other serial numbers you supplied when it reaches the end of that tape.

Example

The original tapes have the serial number 111111, 222222, and 333333, and the following COPY statement is specified:

```
COPY . . . TO <tape name> (LIBMAINTAPPEND = TOEND,
                           SERIALNO = ( 222222, "AAAAAA" ) );
```

The preceding COPY statement is processed as follows:

- The tape with serial number 222222 opens and the LIBMAINTDIR file is read. It is determined that the tape 333333 is the last tape in the set.
- Tape 222222 closes.
- Tape 333333 opens.
- Library maintenance moves to the end of the last file on tape 333333.
- Library maintenance appends three new files to the end of tape 333333.
- If tape 333333 fills before the copy operation completes, tape AAAAAA opens and the additional data is appended to tape AAAAAA.

LIBMAINTAPPEND = TOENDRESTORE

This case works similarly to the LIBMAINTAPPEND = TOEND case except when an error occurs while copying to the tape. If an error occurs that stops the copy to the tape, then the system restores the LIBMAINTDIR file to where it was at the start of the append request. This means none of the files were appended to the tape. However, you can retry the append request.

LIBMAINTDIR

Determines whether library maintenance should create a tape directory disk file on the DL LIBMAINTDIR disk family. The LIBMAINTDIR tape directory disk file describes the destination tape and the files copied to it. Library maintenance gives these files names of the form LIBMAINTDIR/<tape name>/<date>/<tape serialno>. It also puts them under the usercode that library maintenance is running under, or * if there is no usercode.

Library maintenance stores the following information in these files:

- Serial numbers of the tapes used
- Names of the files copied to those tapes
- Certain other attributes of those files

You receive a report of the information in tape directory disk files by running the SYSTEM/FILEDATA utility program and using the following syntax:

```
<filedata modifier> LIBMAINTDIR = <disk file name>
```

Note: When you specify LIBMAINTDIR=TRUE, library maintenance writes the tape with ANSI87 labels.

Any attempt to purge a library maintenance tape for which there is a tape directory disk file resident on the DL LIBMAINTDIR disk family requires approval by the operator. When library maintenance or any other program overwrites a library maintenance tape for which there is a tape directory disk file resident on the DL LIBMAINTDIR disk family the system automatically removes that LIBMAINTDIR file.

LOCATECAPABLE

Set the LOCATECAPABLE attribute to ON to indicate that the file requires a tape drive capable of processing the READ POSITION and LOCATE BLOCK ID tape commands for fast tape access.

If the assigned tape drive is locate capable, then library maintenance automatically takes advantage of this feature to do high-speed spacing in the following situations:

- COMPARE Option
Library maintenance uses the LOCATE BLOCK ID tape command to backspace to compare the file. If you receive a RECOPY REQUIRED message and respond "OK," library maintenance uses LOCATE BLOCK ID to backspace to the beginning of that file to recopy the file. If you respond with "OF," the file is erased.
- ARCHIVE Directory for Destination Tapes
For a tape that is locate capable, library maintenance stores the BLOCK ID of the start of each file it copies in the SYSTEM/ARCHIVE directory. If you specify LIBMAINTDIR = TRUE, library maintenance also stores BLOCK ID information in the LIBMAINTDIR directory.
- ARCHIVE Directory for Source Tapes

If the original tape was created on a locate capable tape drive, then library maintenance uses the LOCATE BLOCK ID information found in the ARCHIVE directory to rapidly space up to each of the files to be copied.

Note: *If you intend to add files to the tape later by specifying LIBMAINTAPPEND = TOEND, then specify LOCATECAPABLE = ON to get the correct type of tape unit.*

OFFSITE

When you specify OFFSITE for a tape destination, library maintenance updates the onsite/offsite status of that tape in the volume library or in the volume directory.

If library maintenance does not successfully copy a file to a destination tape, it purges the tape and does not update the onsite/offsite status for the tape.

If library maintenance successfully copies the files and OFFSITE is TRUE, then, it marks the archive entries for files copied to that tape as "offsite." When library maintenance closes the tape, it updates the entry for the tape in the volume library or in the volume directory to indicate that the volume or volumes are "offsite." If library maintenance successfully copies the files and OFFSITE is false, then, when library maintenance closes the tape, it updates the entry for the tape in the volume library or the volume directory to indicate that the volume or volumes are "onsite."

Note: *The volume library and volume directory entries are updated only at sites that use the OP + CATALOGING option and SECOPT TAPECHECK = AUTOMATIC option respectively.*

SAVEFACTOR

Indicates the expiration date of a tape volume in terms of the number of days past the creation date. The default value for archive and library maintenance tapes is 30 days.

When an ARCHIVE backup, ARCHIVE ROLLOUT, or ARCHIVE MERGE request makes a backup copy of a file that already has four backup copies, the system replaces the backup copy that has the earliest expiration date. If the expiration dates are the same, the system replaces the backup copy that has the earliest creation date.

SECURITYGUARD

Identifies the guard file to be invoked for the tape volume if the SECURITYTYPE attribute is assigned a value of GUARDED or CONTROLLED. The default value is a null string (""). For more information about guard files, refer to the *MCP Security Overview and Implementation Guide*.

Note: *This attribute applies only to installations running the Security Accountability Facility software.*

SECURITYTYPE

Identifies the tape volume security type. This attribute can have a value of PRIVATE (default), PUBLIC, GUARDED, or CONTROLLED. PRIVATE tape volumes can be accessed or overwritten only by their owners and privileged users. PUBLIC tape volumes can be accessed by tasks with any usercode, as limited by the setting of the SECURITYUSE attribute. The security of GUARDED and CONTROLLED tape volumes is determined by the guard file referenced by the SECURITYGUARD attribute.

Note: This attribute applies only to installations running the Security Accountability Facility software.

SECURITYUSE

Specifies how a tape volume that is to be protected by the SECURITYTYPE attribute can be accessed by nonprivileged users using nonprivileged programs.

Note: This attribute applies only to installations running the Security Accountability Facility software.

SERIALNO

Identifies the specific tape volumes to be used when copying files. This attribute does not have a default value. For more information, refer to [Serial Number Assignment](#).

SCRATCHPOOL

Identifies the scratch pool from which the tape is retrieved for archiving. The scratch pool name can be a 17character identifier. This attribute does not contain a default value.

USECATALOG

If true, indicates that a listed tape in the volume library should be used. The archive procedure uses this attribute at cataloging installations only.

ARCHIVE CD Volume

<archive cd volume>

—<CD name>—<archive CD volume attribute list>—————|

Explanation

The volume name specifies the name of the destination CD-ROM volumes.

You can optionally specify one SERIALNO for a CD-ROM destination. If you do include a serial number, the system uses that serial number for all CDCOPIES it creates and for any extra CD-ROM volumes it needs to contain all the files being copied. If you do not include a

serial number, then the system uses a serial number of the form YYMMDD (a 6digit number in which the first two digits correspond to the year, the next two digits correspond to the month, and the last two digits correspond to the day of the month). For example, an archive backup to CD-ROM done on February 10, 2001 would use the serial number 010210 if you did not specify a serial number in the ARCHIVE backup statement.

Normally tape volumes have unique serial numbers. Therefore, when an ARCHIVE RESTORE process issues a "NO FILE" RSVP message for a tape, including the serial number, it is relatively easy for the operator to locate and load the requested volume. When making an archive backup to a CD-ROM volume, try to pick a volume name and optionally a serial number to help you locate the correct CDROM volume during an ARCHIVE RESTORE. When restoring from a CD-ROM volume, the "NO FAMILY" RSVP message includes the volume name, the serial number, and the volume sequence number (which is displayed as the family index number).

```
NO FAMILY <volume name> (CD) [<serial number>]
# <family index number>
```

The volume sequence number identifies which volume is needed in cases where the ARCHIVE backup process filled up one CD-ROM volume and continued copying files to one or more additional CD-ROM volumes.

Note: Unlike tape volumes, you cannot put a serial number on a CD-ROM volume in advance. When you include a serial number in an ARCHIVE backup statement for a CDROM volume, the system gives all the CD-ROM volumes it copies files to the same serial number.

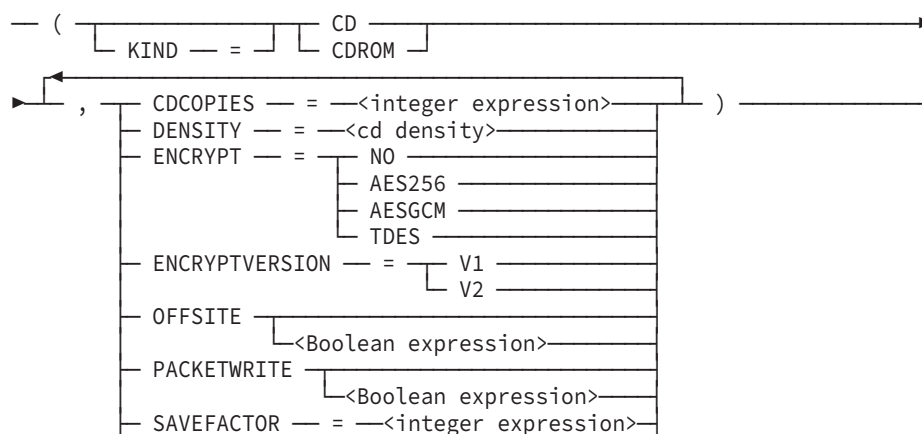
Example

The following example causes the system to back up all the files on PACK to one or more CD-ROM volumes

```
ARCHIVE FULL & REPORT *= FROM PACK TO PACK20010510 (CD);
```

ARCHIVE CD Volume Attribute List

<archive cd volume attribute list>



└ SERIALNO — = —<serial number>——┐

Explanation

You can specify the following attributes for the archive tape volume attribute list:

- CDCOPIES
- DENSITY
- ENCRYPT
- ENCRYPTVERSION
- MULTIVOLUME
- OFFSITE
- PACKETWRITE
- SAVEFACTOR
- SERIALNO

CDCOPIES

The CDCOPIES value specifies how many duplicate CD-ROM volumes should be created. A value of 2 indicates one original and one duplicate. A value of 3 indicates one original and two duplicates. Do not get CDCOPIES mixed up with the extra CD-ROM volumes the archive process creates when the files it is copying overflows the first volume. The archive process creates duplicates of each overflow volume according to the value of CDCOPIES.

DENSITY

The DENSITY value identifies the size of the CD image that needs to be created and burnt on to the CD or DVD media. Refer to the section on DENSITY in the *File Attributes Programming Reference Manual* for the allowable values.

ENCRYPT

Specifies that library maintenance encrypt the data it writes to the specified destination CD. The following three encryption algorithms are provided:

- TDES
- AES256
- AESGCM

To use AESGCM, Media Encryption Version 2 is required.

Note: You cannot specify ENCRYPT for disk destinations.

You cannot specify ENCRYPT for a source CD. Library maintenance automatically decrypts data when reading from an encrypted CD, so you can use a simple COPY statement to copy files from an encrypted CD:

```
COPY ... files ... FROM <CD name>;
```

ENCRYPTVERSION

Specifies the Media Encryption Version to be used. This overrides the value of the SYSOPS LMDEFENCRYPT version.

- V1
- V2

MULTIVOLUME

If you specify the MULTIVOLUME attribute, the WFL compiler issues a syntax error. The archive system automatically sets the MULTIVOLUME attribute.

OFFSITE

If you set the OFFSITE attribute to TRUE, the system marks the backup records it creates as referring to an offsite volume. During an archive restore, the system does not use volumes marked as offsite unless there are no better alternatives.

PACKETWRITE

The PACKETWRITE attribute determines whether the archive process writes the CDROM volumes in track-at-once mode or packet mode. Track-at-once mode (the default) is more efficient, and the resulting CD-ROM volumes can be read by CD-ROM and CD-R units. Packet mode avoids possible buffer underrun I/O errors during the write process and so is safer, but packet mode CD-ROM volumes can be read only by CD-R units.

The MCP enables the buffer underrun prevention feature on CD-RW drives that support it. With this feature enabled, these drives resume writing after a buffer empty condition. Thus, buffer underrun does not occur in track-at-once mode, and there is no need to use the packet mode.

SAVEFACTOR

The SAVEFACTOR attribute indicates the expiration date of the CD-ROM volume in terms of the number of days past the creation date. The default value for archive CDROM volumes is 30 days.

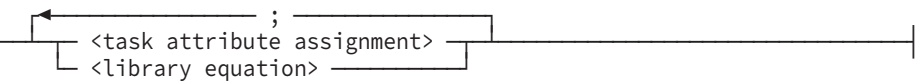
When an ARCHIVE backup, ARCHIVE ROLLOUT, or ARCHIVE MERGE request makes a backup copy of a file that already has 4 backup copies, the system replaces the backup copy that has the earliest expiration date. If the expiration dates are the same, the system replaces the backup copy that has the earliest creation date.

SERIALNO

You can optionally specify one serial number. The system uses that serial number for each of the CD-ROM volumes it creates during the backup process. If you do not specify a serial number the system gives the output CD-ROM volumes a serial number of the form YYMMDD, where the first two digits indicate the year, the next to digits indicate the month, and the last two digits indicate the day of the month.

ARCHIVE Task Equation List

<archive task equation list>



Explanation

The following table describes the archive task equation list.

Name	Description
<task attribute assignment>	Assigns task attributes to the ARCHIVE statement. For a complete explanation of task attribute assignments, refer to Section 5, Task Initiation .
<library equation>	Changes the attributes of libraries. The archive subsystem uses library equation to change the selector library. For more information, refer to Section 5, Task Initiation .

Library Equation

Library-equating SELECTOR causes the archive task to use the <file title> library instead of the standard SL ARCHIVESUPPORT library:

```
LIBRARY SELECTOR (TITLE = <file title>, LIBACCESS = BYTITLE);
```

TASKSTRING assignment

You can set TASKSTRING to specify the disk family or families that archive backup requests should use as temporary storage when copying to a CD-ROM volume. If you specify the following, then the archive backup system stores the temporary CDIMAGE disk file on that disk family:

```
TASKSTRING = "FAMILYNAME= <family name>"
```

If you specify the following, the archive backup system starts with the first family in the list:

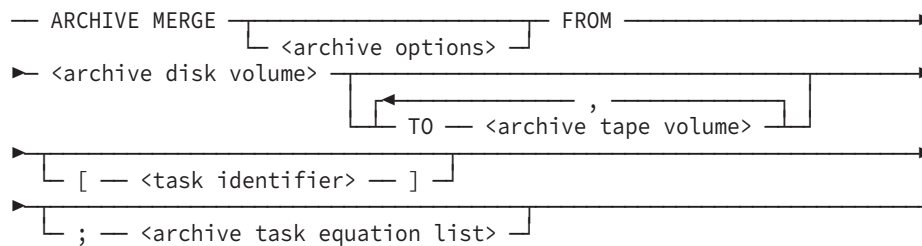
```
TASKSTRING = "FAMILYNAME= (<family name>,  
  <family name>, ... <family name>)"
```

If you have installed a custom ARCHIVESUPPORT library that uses the SFORKV multitasking feature then tasks rotate through the list of family names you specify.

For information about the multitasking feature, refer to the MCP System Interfaces Programming Reference Manual.

ARCHIVE MERGE Statement

<archive merge statement>



Explanation

The ARCHIVE MERGE statement transfers backup copies of files that reside on one or more backup tapes to a single tape. If there is insufficient space to merge all requested backup tape files to one tape, a reel switch condition results and the remaining files are merged on another tape. This produces a single tape set of merged files.

The primary purpose of this operation is to enable more efficient use of tape resources. When you merge files to a single backup tape or to a tape set, space is freed on other tape volumes for other uses.

The ARCHIVE MERGE statement requests as input any file that has been archived to a backup tape by any ARCHIVE backup statement, ARCHIVE MERGE statement, or ARCHIVE ROLLOUT statement.

Note: Use of the ARCHIVE MERGE statement usually requires privileged access. However, a job can run this statement if it was started from an ODT without a usercode. In any case, this statement affects all tape files that were backed up or rolled out to a backup tape through the archive subsystem.

As a merge operation executes, it requests as input the backup tape files that were created by preceding ARCHIVE statements. During the operation, the archive subsystem prompts you to load the input tapes on the tape drives twice: once while the system creates the output tape directory, and again when it copies the selected files.

You can use the OF (Optional File) system command to cause the merge operation to skip an input tape value. You can enter <mix number> OF when the archive subsystem prompts you to load the input tape. You can use the HI (Cause Exception Event) system command to check the progress of an ARCHIVE MERGE statement. A command of the form <mix number> HI displays the number of files already copied and other information.

Example

The following example illustrates how to consolidate archive backup tapes for the family DISK onto a new backup tape named MERGEDISK:

```
ARCHIVE MERGE FROM DISK TO MERGEDISK
```

ARCHIVE PURGE Statement

<archive purge statement>

— ARCHIVE PURGE —
 └─> <long file name constant> , ─┐
 └─> <long directory name constant> ─┐
 └─> FROM ─┐
▶<archive disk volume>────────────────────────────────┘

Explanation

This statement purges archive backup records from the archive directory without affecting resident or nonresident files in any way. The ARCHIVE PURGE statement does not affect catalog backup information.

Purging archive backup records serves to remove only references to backup files. To remove an actual resident disk file and all archive records associated with it, issue a REMOVE statement followed by an ARCHIVE PURGE statement. Both statements must identify the name of the target file. You can also refer to the REMOVE statement with the DESTROY option that is described later in this section.

If you remove a file, but you do not purge its corresponding archive backup records, other ARCHIVE statement operations will continue to search for that file. Searches for disk files that have been removed from disk, but for which archive backup records still exist, result in NO FILE conditions.

If your installation uses the archive AUTORESTORE feature, the archive subsystem responds to each NO FILE condition by reloading the missing archived files from tape.

Example

The following example purges the entries for FILE/1 and FILE/2 from the archive directory for the family MCPMAST:

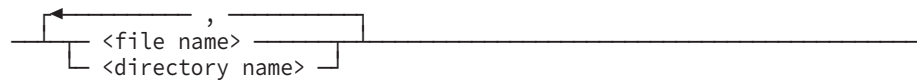
```
ARCHIVE PURGE FILE/1, FILE/2 FROM MCPMAST
```

ARCHIVE RELEASE Statement

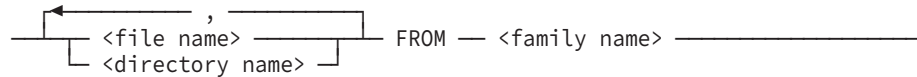
<archive release statement>

— ARCHIVE RELEASE —
 └─> <archive list> ─┐
 └─> <archive from group> ─┐
 └─> , ─┐
 └─> <archive list> ─┐

<archive list>



<archive from group>



Explanation

The ARCHIVE RELEASE statement removes files from disk that are not in use and have up-to-date archive backup records.

The removed files can be restored by the archive AUTORESTORE feature or the ARCHIVE RESTORE and ARCHIVE ADDRESTORE statements.

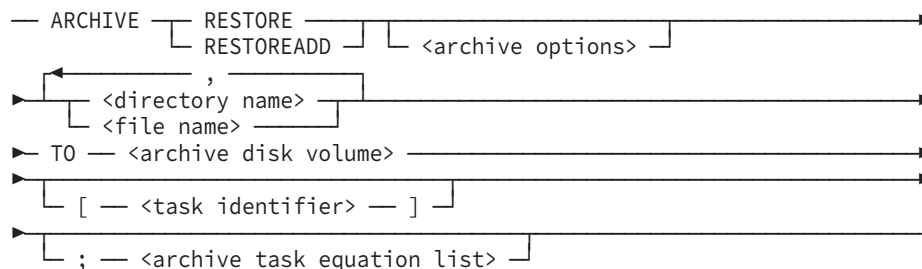
In addition to the ARCHIVE RELEASE statement, see also the REMOVE and REMOVE DESTROY statements described later in this section.

The first name in a remove list can be a file title or a directory title; that is, it can contain an ON family name part. Subsequent names in the remove list can only contain an ON family name part if they contain a string primary as well.

In the remove from group, the FROM clause applies to all the file names and directory names in that remove from group.

ARCHIVE RESTORE Statement

<archive restore statement>



Explanation

The ARCHIVE RESTORE statement reloads backup copies of disk files to disk. You can use this statement to recover disk files that have been damaged, lost, or inadvertently removed from disk. Only backup files that were copied through one of the ARCHIVE backup statements, the ARCHIVE MERGE statement, or the ARCHIVE ROLLOUT statement, can be restored to disk through the ARCHIVE RESTORE statement.

Note: WFL includes another statement called simply *RESTORE*, rather than *ARCHIVE RESTORE*. The two statements serve different purposes. The *ARCHIVE RESTORE* statement reloads files that have archive backup directory entries. The *RESTORE* statement reloads files from any library maintenance tape, regardless of whether or not the files have archive backup directory entries.

The *ARCHIVE RESTORE* statement has the following variants, which function differently:

- **ARCHIVE RESTORE**
This statement reloads selected backup copies of files to disk even if versions of those files already exist on the destination disk.
- **ARCHIVE RESTOREADD**
This statement reloads only those selected files for which versions do not already reside on the destination disk.

With both the *ARCHIVE RESTORE* and the *ARCHIVE RESTOREADD* statements, the archive subsystem selects files by comparing the files you request with records in the archive directory. The restore and restoreadd processes do not affect the archive directory in any way. You can remove a restored file and restore it to disk again, even if an intervening archive backup process has not been performed on that file.

Note: In most installations, you must have privileged access to reload files other than your own. However, you can reload another user's files if the job running the *RESTORE* or *RESTOREADD* process runs without a usercode and starts from an ODT.

The archive subsystem usually restores only the most recently written archived files from the most recently written archive tapes. Under the following circumstances, however, the archive subsystem restores older versions of files:

- At installations that use the catalog volume library or the tape volume directory, the archive subsystem checks the status of the most recent backup tape. If the archive subsystem finds that the contents of the most recent backup tape have been purged or overwritten, or if the archive subsystem finds that the most recent backup tape has been marked as destroyed or offsite with a *VOLUME DESTROY* or *VOLUME OFFSITE* statement, then the archive subsystem does not select that tape unless there are no better backup tapes or CDs for a given file.
- If the *ARCHIVE* backup statement for the most recent backup tape or CD included the attribute *OFFSITE* or the phrase *OFFSITE=TRUE*, then the archive subsystem does not select that tape or CD unless there are no better backup tapes or CDs for a given file.
- If an *ARCHIVE VOLUME OFFSITE* statement was executed that affected the status of the most recent backup tape or CD, then the archive subsystem does not select that tape or CD unless there are no better backup tapes or CDs for a given file.
- Your installation uses a custom version of the archive support library and that library selects an older version of a backup tape file for the restore or restoreadd operation.

You can use the *HI* (Cause Exception Event) system command to check the progress of an *ARCHIVE RESTORE* statement. A command of the form *<mix number> HI* displays the number of files already copied and other information.

Examples

The following example illustrates how to manually restore an entire family:

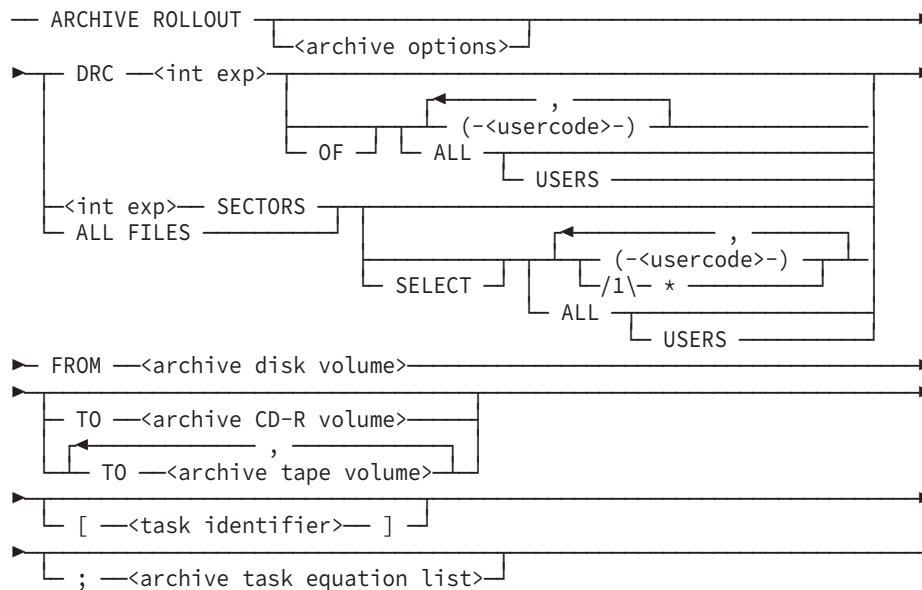
```
ARCHIVE RESTORE = TO USERPACK
```

The following example shows how to restore only those files that do not already reside on the disk where the family TESTPACK is located:

```
ARCHIVE RESTOREADD = TO TESTPACK
```

ARCHIVE ROLLOUT Statement

<archive rollout statement>



Unable to split diagram to fit file

Explanation

This statement moves selected disk files to archive backup tape media. The primary function of this option is to free disk space for other uses, particularly when resources are limited.

You can use this option to either

- Specify the amount of disk space (in sectors) you want to make available.
- Specify a percentage of the authorized DRC disk space by which each user's space is to be reduced.

DRC Option

You can specify disk space by using the DRC option of the ARCHIVE ROLLOUT statement. The DRC option is available to you even if your installation does not actively run the DRC subsystem. This option affects only those users for whom DRC limits are assigned in the USERDATAFILE.

As each selected file is rolled out to tape, the archive subsystem records the transfer in the archive directory and the file is removed from the disk. You can use rolled out files as input to any library maintenance command or statement.

Note: *If the archive directory includes references to archived copies of the files that your rollout operation is requesting, the rollout process removes the resident versions of the files without recopying those files to tape. Therefore, it is possible to complete a rollout operation without receiving a system request for a backup tape.*

Before the archive statement actually performs a rollout operation, it evaluates files for possible selection. This process is based on your usercode, or the usercode of the job that is running the ARCHIVE ROLLOUT statement.

- If you do not list specific usercodes with your ARCHIVE ROLLOUT statement, only files under the usercode of the job that is running the rollout process are evaluated for possible selection.
- If the job that is running the ARCHIVE ROLLOUT statement started at an ODT without a usercode, then only files without a usercode are evaluated for selection.
- If your usercode enables you privileged access and you specify ALL USERS in the ARCHIVE ROLLOUT statement, all files are evaluated for possible selection.

Specifying SECTORS or Using the DRC Option

You can use the ARCHIVE ROLLOUT statement to make available a particular number of in-use sectors on a disk family in either of two ways:

- Use the SECTORS option of this statement.
- Invoke the DRC option.

The following discussion describes the differences between these two approaches to freeing disk space.

In general, using the ARCHIVE ROLLOUT statement on files other than your own requires privileged access to files.

ARCHIVE ROLLOUT statements that include the SECTORS options do not require that DRC limits be set in the USERDATAFILE.

If you use the DRC option, the rollout operation affects only the files under usercodes for which DRC limits are assigned in the USERDATAFILE.

When you use the ARCHIVE ROLLOUT statement to free a specified number of in-use sectors on a disk family (by using the SECTORS option), the ARCHIVE statement can be used to copy files to tape and remove files from disk to satisfy your request. If you have specified more sectors than are already in use under your usercode on the disk family, the archive subsystem selects all of your files for the rollout operation.

In this case, the operation is executed as if you had specified the ALL FILES option in the statement.

Checking the Progress of the Rollout

You can use the HI (Cause Exception Event) system command to check the progress of an ARCHIVE ROLLOUT statement. A command of the form <mix number> HI displays the number of files already copied and other information.

Examples

The ARCHIVE ROLLOUT statement is used in the following example to free 1000 sectors from files that are under usercode TEMP or DONTCARE:

```
ARCHIVE ROLLOUT 1000 SECTORS SELECT (TEMP), (DONTCARE) FROM MYPK
```

The following example moves all files belonging to user MIKEB to MIKESAVE:

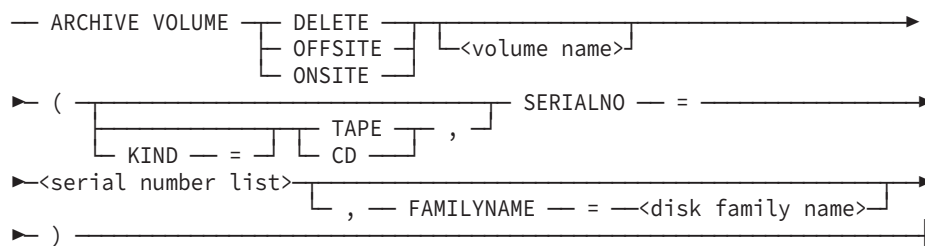
```
ARCHIVE ROLLOUT ALL FILES (MIKEB) FROM DISK TO MIKESAVE
```

The following example uses the DRC option of the ARCHIVE ROLLOUT statement. If the DRC authorization for usercode XYZ on PACK is 20000 sectors, then this example rolls out files belonging to XYZ until that user's files occupy 18000 sectors or less.

```
ARCHIVE ROLLOUT DRC 10 (XYZ) FROM PACK
```

ARCHIVE VOLUME Statement

<archive volume statement>



Explanation

Use the ARCHIVE VOLUME DELETE statement to remove archive backup references to CD-ROMs or tape volumes that have been purged, overwritten, or damaged. This action prevents archive restore tasks from requesting CD-ROMs and tape volumes that cannot be used.

Use the ARCHIVE VOLUME OFFSITE statement to mark archive backup references to selected CD-ROMs or tape volumes as not located at the site. The archive restore and merge tasks then avoid requesting offsite CD-ROMs and tape volumes, unless there are files with no other archive backups.

Use the ARCHIVE VOLUME ONSITE statement to mark archive backup references to selected CD-ROMs or tape volumes as onsite. You can also use this statement to reverse the actions of a previous ARCHIVE VOLUME OFFSITE statement, and you can use it to mark volumes as onsite that were originally created as offsite.

Notes:

- *Only privileged jobs and jobs started from an ODT can issue an ARCHIVE VOLUME statement.*
- *The ARCHIVE VOLUME statement does not affect the volume library at sites that use cataloging, and it does not affect the volume directory at sites that use SECOPT TAPECHECK = AUTOMATIC.*

If you do not specify KIND, then the system assumes KIND = TAPE.

The serial number list is a list of the serial numbers of one or more CD-ROM or tape volumes that the system can act on. If you specify more than one serial number, then the system works on the archive backup information for all entries that match any of the serial numbers you specified.

If you specify the volume name, then the system only works on those archive records that have a matching volume name and a matching serial number. If you do not specify a volume name, then the system works on all archive records that have a matching serial number.

If you specify FAMILYNAME = <disk family name>, then the system works on the archive directory for that disk family only. If you do not specify FAMILYNAME, then the system works on the archive directories for all the online disk families.

Options

The following options are available in the ARCHIVE VOLUME statement.

DELETE

If you specify DELETE, the system removes all references to the specified CD-ROM or tape volumes from the specified archive directory or all archive directories.

OFFSITE

If you specify OFFSITE, the system marks all references to the specified CD-ROM or tape volumes from the specified archive directory or all archive directories as offsite. The WFL ARCHIVE RESTORE and ARCHIVE RESTOREADD statements do not try to copy a file from an offsite CD-ROM or tape volume unless it is the only CD-ROM or tape volume on which there is a copy of a specific file.

Specifying OFFSITE marks the tapes in the archive directory the same way that the following ARCHIVE backup statement does:

```
ARCHIVE FULL X/= FROM MYPACK TO MYTAPE (OFFSITE);
```

ONSITE

If you specify ONSITE, the system marks all references to the specified CD-ROM or tape volumes from the specified archive directory or all archive directories as not offsite.

Examples

The following examples illustrate the use of the ARCHIVE VOLUME statement syntax.

The following statement deletes all references to the tape with serial number 555555 from all active archive directories:

```
ARCHIVE VOLUME DELETE (SERIALNO = 555555);
```

The following example marks all references to tape volumes with the name DISK06122 and serial numbers 111111 or XXX in the archive directory for the family DISK as offsite:

```
ARCHIVE VOLUME OFFSITE DISK06122 (SERIALNO=(111111,"XXX"),
                                     FAMILYNAME=DISK);
```

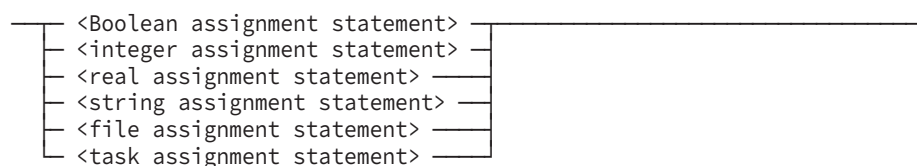
The following example shows an archive backup statement creating a tape with offsite status and then an ARCHIVE VOLUME ONSITE statement giving that tape volume onsite status:

```
ARCHIVE FULL MYFILE FROM PACK
                                     TO MYTAPE (OFFSITE, SERIALNO="Q5");
ARCHIVE VOLUME ONSITE MYTAPE (SERIALNO="Q5");
```

Note: For large archive directories, the ARCHIVE VOLUME statement runs for a correspondingly long time.

Assignment Statements

<assignment statement>



<Boolean assignment statement>

— <Boolean identifier> — := — <Boolean expression> —————

<integer assignment statement>

— <integer identifier> — := — <integer expression> —————

<real assignment statement>

$$\text{— } \langle \text{real identifier} \rangle \text{ —} := \text{— } \langle \text{real expression} \rangle \text{ —}$$

<string assignment statement>

$$\text{— } \langle \text{string identifier} \rangle \text{ —} := \text{— } \langle \text{string expression} \rangle \text{ —}$$

<file assignment statement>

— <file identifier> — (— <file attribute assignment> —) —

<task assignment statement>

$$\text{— } \langle \text{task identifier} \rangle \text{ — } (\text{— } \left. \begin{array}{l} \langle \text{task attribute assignment} \rangle \\ \langle \text{file equation} \rangle \end{array} \right\} \text{— }) \text{ — }$$

Explanation

The assignment statement assigns values to declared variables.

For more information about the file assignment statement, refer to [Using File Attributes](#).

For more information about the task assignment statement, refer to [Using Task Variables](#).

Example

The first section of the following example declares variables of type Boolean, integer, real, string, file, and task. The statements in the next section of the example assign values to the declared variables.

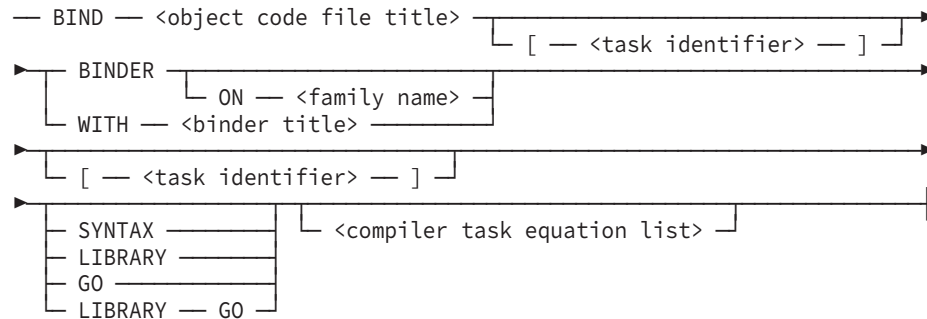
```

BOOLEAN B;
INTEGER I;
REAL R;
STRING S;
FILE F;
TASK T;
.
.
.
B:=FILE A/B ON PACK IS RESIDENT;
I:=INTEGER(R);
R:=17.5;
S:="ABC";
F(AREASIZE=1008,FLEXIBLE); % File Attribute Assignment
T(PRIORITY=70); % Task Attribute Assignment
.
.
.

```

BIND Statement

<bind statement>



<binder title>

— <file title> —

Explanation

The BIND statement invokes the Binder to combine object code files.

Binder uses the following sources of input:

- A primary input file titled CARD, which contains directions to the Binder
- A host file titled HOST, which is the object code file to which the subprograms are to be bound
- Subprogram files, which contain the subprograms to be bound to the host program

For a more detailed explanation and the complete syntax, see [COMPILE or BIND Statement](#) later in this section. Refer to the *Binder Reference Manual* for instructions regarding using Binder.

Example

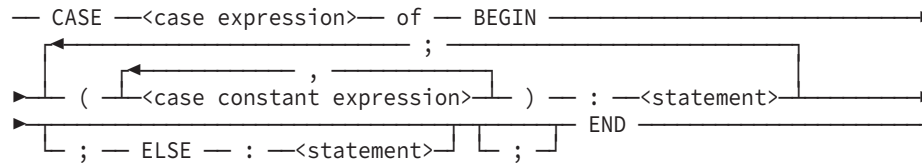
The following is an example job that uses the BIND statement:

```

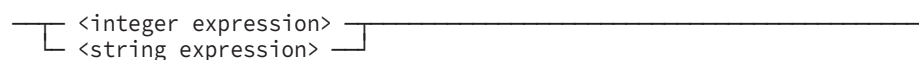
?BEGIN JOB BIND/RESULT;
  BIND COBOL85/EXAMPLE WITH BINDER LIBRARY;
  BINDER DATA CARD % This local data specification
    HOST IS COBOL85/HOST; % replaces the input file CARD.
  USE S1 FOR PROG;
  BIND S1 FROM COBOL85/PROG;
? % End Binder data
?END JOB.
  
```

CASE Statement

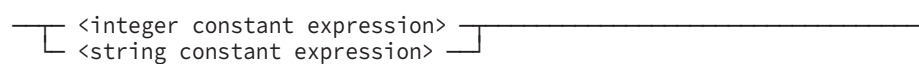
<case statement>



<case expression>



<case constant expression>



Explanation

The CASE statement provides a way to dynamically select one of several alternative statements, depending on the value of the case expression.

The case constant expressions, in the parentheses that precede the statements, must be of the same type as the case expression. In addition, no two case constant expressions within the same CASE statement can have the same value.

The ELSE specification specifies an action to take if the value of the case expression is not equal to any of the case constant expressions. If there is no ELSE specification, and the case expression does not equal any of the case constant expressions, then a fatal runtime error occurs.

Example

The following is an example job that uses the CASE statement:

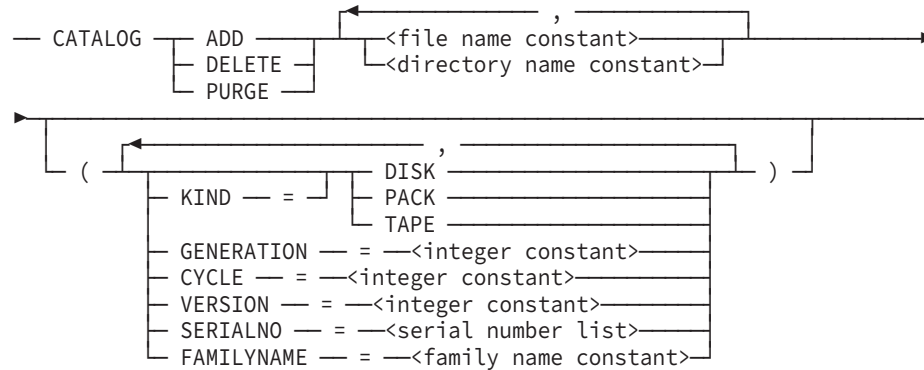
```

?BEGIN JOB CASE/EXAMPLE(STRING DAY);
  CLASS=2;
  CASE DAY OF
    BEGIN
      ("MONDAY",
       "TUESDAY",
       "WEDNESDAY",
       "THURSDAY"): RUN OBJECT/DAILY/UPDATE;
      ("FRIDAY"): RUN OBJECT/WEEKLY/UPDATE;
      ("SATURDAY",
       "SUNDAY"): ; % NO RUN NEEDED
    ELSE: ABORT "INVALID INPUT STRING:" & DAY;
  END;
?END JOB.

```


CATALOG Statement

<catalog statement>



Explanation

Note: Cataloging is not supported for files within the permanent directory namespace.

The CATALOG statement applies only to a cataloging system. Cataloging provides an automated method for locating copies of files that are on disk or tape. Cataloged files can be stored only on disk or tape that has been entered into the cataloging volume library with the VOLUME ADD statement. Different copies of a file are referred to as generations.

For tape, the CATALOG ADD statement enters the name of a permanent file or directory into the catalog. For disk, CATALOG ADD marks the requested generation of each of the requested files as cataloged.

The CATALOG DELETE statement removes all references to the specified generation of the specified files from the system catalog. If the specified generation is resident on disk, the statement also marks that file as not cataloged.

The CATALOG PURGE statement removes all the backup file information for all generations of the specified files. If there is a resident generation on disk, the statement also marks that file as not cataloged.

The CATALOG DELETE and CATALOG PURGE statements delete only catalog entries; the resident and backup files are still available but cannot be accessed through the catalog.

The generation is determined by the integer file attributes GENERATION, CYCLE, and VERSION, as well as by the time stamp automatically maintained by the system. The most recent generation is given by a value of 0 for GENERATION or, if GENERATION is not specified, by the highest value of CYCLE and the highest value of VERSION within that cycle. For the CATALOG DELETE and CATALOG ADD statements, you can use these file attributes to identify a specific file generation.

The SERIALNO option is required in either of the following cases:

- If KIND = TAPE
- If KIND = PACK and the family is offline

Family substitution is used if the job or task has an active family specification. Only the primary family name is used. Refer to [FAMILY Assignment](#) and [Interrogating Complex Task Attributes](#).

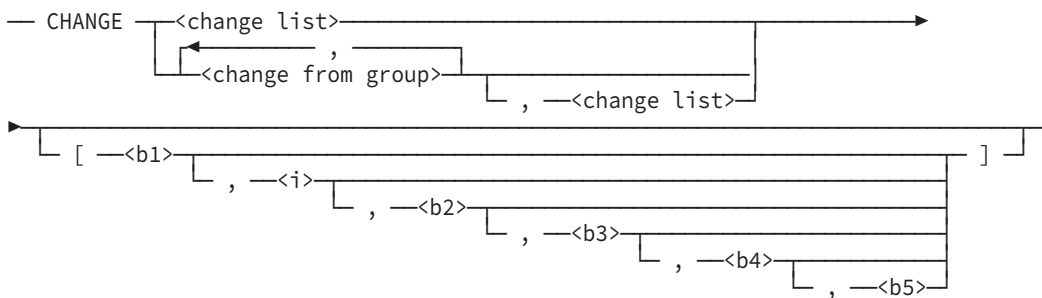
Examples

The following are examples of the CATALOG statement:

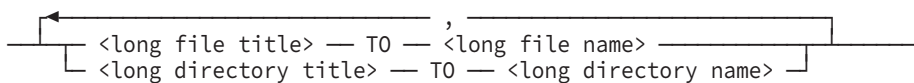
```
CATALOG ADD FILEA (KIND=PACK, SERIALNO=123456)
CATALOG DELETE = (KIND=TAPE, VERSION=12, CYCLE=5)
CATALOG PURGE FILEA, FILEB (KIND=PACK)
```

CHANGE Statement

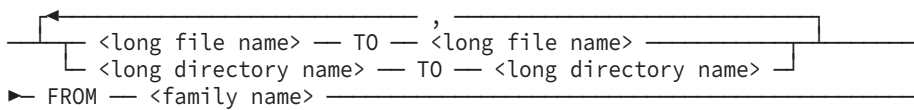
<change statement>



<change list>



<change from group>



Explanation

The CHANGE statement changes the names of files on disk.

Conflicting File Name

If you specify a file name and a file already exists with that file name, the existing file with the same name is removed before the file name specified is changed to the new file name.

Conflicting Directory Change Requests

Because directory change requests are processed in groups of files, two simultaneous change requests that affect the same set of files produce unpredictable results. For example, simultaneous `CHANGE A/= TO B/=` and `CHANGE B/= TO A/=` statements, where both directories `A/=` and `B/=` exist with nonconflicting file names, can result in all files in directory `A/=`, all files in directory `B/=`, or with the files shared between the two directories.

Directory Changes

If you specify a directory, the names of the files in that directory are changed. If the new directory name already exists, the files are added to that directory, and any files that already belonged to the new directory are not changed. The directories `*=` and `=` cannot be used in the `CHANGE` statement.

LOCKEDFILE Attribute

If you use the `CHANGE` statement to change the name of a file whose `LOCKEDFILE` file attribute is set to `TRUE`, the file name is not changed. The system displays the following message to indicate that the file name was not changed:

```
<file name> NOT CHANGED (LOCKEDFILE).
```

See [ALTER Statement](#) in this section for more information about changing the `LOCKEDFILE` file attribute. For additional information about the `LOCKEDFILE` file attribute, refer to the *File Attributes Programming Reference Manual*.

Change From Group

When you use a “change from” group, the `FROM` clause applies to all of the file names and directory names in that “change from” group.

Active Family Specification

Family substitution is used if the job or task has an active family specification. Only the primary family name is used. For more information on active family specification, refer to [FAMILY Assignment](#) and [Interrogating Complex Task Attributes](#).

DATAPATH Attribute Specification

Path substitution is used if the job has an active `DATAPATH` attribute specification. Only the first path element is used. For more information on active `DATAPATH` attribute specification, refer to [DATAPATH Assignment](#) and [Interrogating Complex Task Attributes](#).

Change Privileges

You can use the `CHANGE` statement to change the name of a file or directory if any one of the following conditions is true:

- You are a privileged user.
- You are the owner of the file or directory.

Statements

- You have write access to the file or directory.

Change Results

The CHANGE statement can be followed by an optional results structure: [b1, i, b2, b3, b4, b5]. Parameters are optional. For example, [b1, i] is syntactically correct. However, if a parameter is specified, all preceding parameters must also be specified. The parameters return the following information:

Variable	Type	Description
b1	Boolean	TRUE if a failure occurred. FALSE if all files and directories were changed.
i	Integer	Total number of changed files
b2	Boolean	TRUE if at least one file or directory was not changed because it was not present.
b3	Boolean	TRUE if at least one file was not changed because the file was a locked file.
b4	Boolean	TRUE if at least one file was not changed because of a security error.
b5	Boolean	TRUE if at least one file was not changed because the destination file already existed.

If you run a job that uses this syntax on a 12.0 or earlier MCP system, it is aborted.

Examples

This CHANGE statement changes the name of file X on DISK to Y:

```
CHANGE X TO Y;
```

This statement changes the name of file A/B on USERS to C/D:

```
CHANGE A/B ON USERS TO C/D;
```

This statement changes the names of all the files under the directory A/= on PACK to B/= on PACK:

```
S1:="A/=";  
S2:="B/=";  
CHANGE #S1 ON PACK TO #S2;
```

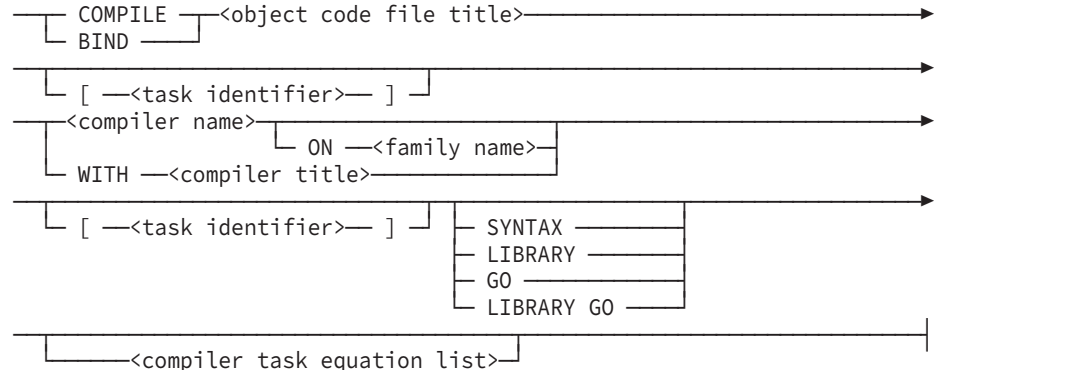
```
CHANGE X TO Y, X/X TO Y/Y FROM MYPACK,
      XX TO YY FROM PACK,
      Z TO ZZ;
```

```
BEGIN JOB DEMONSTRATE/CHANGE;
BOOLEAN BFAILED, BMISSING, BLOCKED, BSEC, BDUP;
INTEGER COUNT;
```

```

DISPLAY "Change test: " & STRING (COUNT, *) & " files changed";
IF NOT BFAILED THEN
    DISPLAY "All files changed";
IF BMISSING THEN
    DISPLAY "Missing files";
IF BLOCKED THEN
    DISPLAY "Locked files seen";
IF BSEC THEN
    DISPLAY "Security error seen";
IF BDUP THEN
    DISPLAY "Duplicate files seen";
END JOB

```



<compiler name>

ALGOL	_____
BINDER	_____
CC	_____
C	_____
COBOL74	_____
COBOL85	_____
DCALGOL	_____
DMALGOL	_____
FORTRAN77	_____
MODULA2	_____
NDLII	_____
NEWP	_____
PASCAL	_____
RPG	_____
SORT	_____

Explanation

The COMPILE statement initiates compilation of a program, and optionally can also cause the resulting object code file to be executed.

Naming the Object Code File

The object code file title construct specifies the name of the object code file that results from the compilation. The source file is assumed to be a card reader file named CARD. However, a file equation can be included in the compiler task equation list to cause a file with a different title or kind to be used instead.

Example

The following example will cause a source file titled COUNTER to be compiled. The resulting object code file will be titled OBJECT/COUNTER.

```
COMPILE OBJECT/COUNTER WITH PASCAL LIBRARY;  
COMPILER FILE CARD(TITLE = COUNTER, KIND = DISK);
```

Choosing a Compiler

The name of the compiler to be used is usually the same as the name of the language that the program is written in. The phrase WITH compiler title specifies the name of the compiler desired, as in the previous example which uses the Pascal compiler.

The prefix SYSTEM/ is always assumed to precede the first node of the compiler title (after any usercode or asterisk (*)). Thus the prefix SYSTEM/ should not be explicitly specified. The compiler title has the same form as a file title, except that the maximum number of nodes is 11.

Examples

The following statement uses SYSTEM/ALGOL:

```
COMPILE OBJECT/X WITH ALGOL LIBRARY;
```

The following statement uses (SITE)SYSTEM/COBOL74 ON PACK:

```
COMPILE OBJECT/X WITH (SITE)COBOL74 ON PACK LIBRARY;
```

If the specified compiler is not a compiler object code file, the job is discontinued.

If the compiler is one of the standard compilers recognized by WFL, the word WITH can be omitted, and the compiler name can be entered by itself. The characters C or CC are synonyms for the C language compiler name.

Binding

The Binder is used to combine two or more object code files into one object code file. The object code files are the result of successful compilations of source files. To use the Binder, include the word BIND instead of COMPILE at the start of the COMPILE or BIND statement, and use as the compiler name.

Binder uses the following sources of input:

- A primary input file titled CARD, which contains directions to the Binder
- A host file titled HOST, which is the object code file to which the subprograms are to be bound
- Subprogram files, which contain the subprograms to be bound to the host program

Example

The following is an example of a bind statement:

```
?BEGIN JOB BIND/RESULT;  
  BIND COBOL85/EXAMPLE WITH BINDER LIBRARY;  
  BINDER DATA CARD % This local data specification  
    HOST IS COBOL85/HOST; % replaces the input file CARD.  
  USE S1 FOR PROG;  
  BIND S1 FROM COBOL85/PROG;  
  ? % End Binder data  
?END JOB.
```

Refer to the *Binder Reference Manual* for instructions regarding using the Binder program.

Object Code File Disposition

A phrase can be included to indicate the disposition of the object code file. The disposition determines whether the object code file is saved or executed. If no disposition is included in the COMPILE statement, a disposition of GO is assumed.

Statements

The following are the possible dispositions and their meanings.

Disposition	Definition
GO	The object code file is executed but not saved. If there are syntax errors, execution does not occur.
LIBRARY	The object code file is saved but not executed. If there are syntax errors, the object code file is not saved.
LIBRARY GO	The object code file is saved and also executed. If there are syntax errors, the object code file is not saved or executed.
SYNTAX	The object code file is not saved and not executed. This disposition is used to check the program for syntax errors only.

Task Variables

Task variables can be included in the COMPILE statement in either of two places. The position determines whether a task variable is associated with the compilation or the execution of the program. The same task variable cannot be assigned to both the compilation and execution, because these are separate tasks.

A task identifier included after the object code file title associates a task variable with the execution of the object code file. This task identifier is not used if the disposition of the COMPILE statement does not cause the object code file to be executed. Because this is usually a programming error, a warning is given to indicate that the task identifier is not used.

A task identifier included after the compiler name or compiler title associates a task variable with the compilation of the object code file.

The task state of the task variables included in the COMPILE statement can be interrogated later to find out if the compilation and execution of the program were successful.

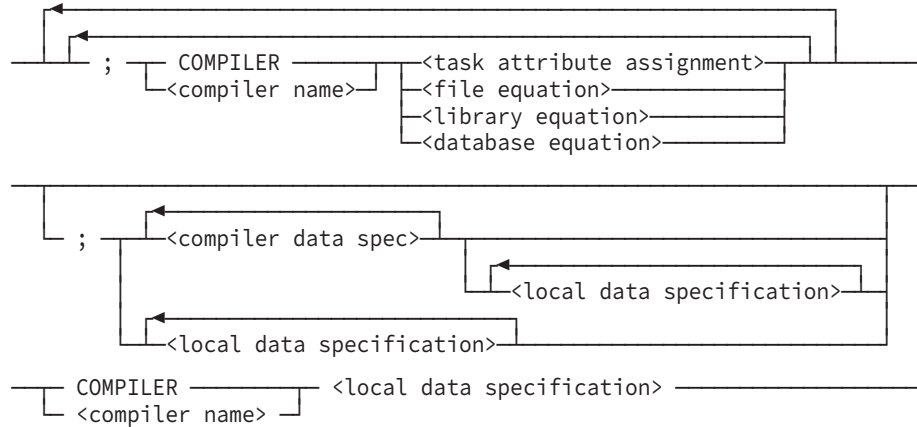
Example

The following example includes task variables with COMPILE statements:

```
COMPILE (RAS)OBJECT/ITAL [TRUN] WITH ALGOL [TCOMP] LIBRARY GO;  
  COMPILER FILE CARD (TITLE = (RAS)ITAL, KIND = DISK);  
IF TCOMP ISNT COMPILEDOK THEN  
  ABORT "UNSUCCESSFUL COMPILE OF OBJECT/ITAL";  
IF TRUN ISNT COMPLETEDOK THEN  
  ABORT "RUN-TIME ERROR IN OBJECT/ITAL";
```


Compiler Task Equation List

<compiler task equation list>



Explanation

A compiler task equation list specifies task equations for use either during compilation or during execution of a program. Task attribute assignments, file equations, library equations, database equations, and local data specifications are all defined in [Section 5, Task Initiation](#).

File, Library, and Database Equations and Task Attributes

Task attributes, file equations, library equations, and database equations are preceded by the word COMPILER or a compiler name if they are to be applied to the compilation. Otherwise, they are permanently attached to the resulting object code file. These values are assigned to the task whenever the compiled program is executed, unless the attribute values are overridden by run-time task equations.

The MODIFY statement permits task attributes, file equations, library equations, and database equations that are compiled into an executable object code file to be added to or changed, without recompiling the source file. For further information, refer to the description of the MODIFY statement provided later in this section.

Task attributes, file equations, library equations, and database equations that apply to the compilation can be mixed with ones that apply to the object code file in any order.

All expressions in task attribute assignments and file attribute assignments for the program are evaluated prior to compilation. The values obtained are stored in the object code file as if they had been specified as constants.

Typically, the source file to be used by the compiler is specified by a file equation. For an example, see "Naming the Object Code File" earlier in this section.

Examples

File equations that change the attributes of the object code file are enabled, but most are overridden by the compiler and have no effect. However, the security attributes of the object code file can be set through a file equation, as in the following example:

```
COMPILE OBJECT/TEST WITH ALGOL LIBRARY;  
COMPILER FILE CODE (SECURITYTYPE=PUBLIC, SECURITYUSE=IN);
```

The resulting object code file has a security of PUBLIC IN. The same effect can be achieved by using the SECURITY statement to change the security of the object code file after it is compiled.

Global file equation of the object code file produced by the compiler is not permitted. For example, if the following statement is specified, a syntax error is given:

```
COMPILE OBJECT/TEST WITH ALGOL;  
COMPILER FILE CODE:=GLOBALFILE; % Illegal syntax
```

Local Data Specifications

Local data specifications in a COMPILE or BIND statement are applied either to the compilation or the resulting object code file, according to the following rules:

- A local data specification that is to be applied to the compilation must be preceded by the word COMPILER or a compiler name.
- All the local data specifications to be applied to the compilation must precede all the local data specifications to be applied to the execution of the object code file.

Example

Local data specifications can be applied to the compilation to replace input files used by the compiler. In the following example, a local data specification takes the place of the CARD file:

```
COMPILE OBJECT/RUNNER WITH ALGOL LIBRARY GO;  
  COMPILER FILE TAPE (TITLE = RUNNER/B);  
COMPILER DATA CARD  
? % End data CARD
```

Local data specifications can be applied to the execution of the object code file to replace input files that would normally be read by the program at run time.

Local data specifications that follow a COMPILE or BIND statement are reread if the COMPILE or BIND statement is executed more than once.

Run-Time Overriding of Compiler Task Equation

The following example shows how task equations set for a program at compile time can be overridden at run time:

```

COMPILE OBJECT/X WITH ALGOL LIBRARY;
  COMPILER FILE CARD (TITLE=X,KIND=DISK);
  ALGOL PRIORITY=50; % Sets priority of ALGOL compilation.
  PRIORITY=60; % Sets priority in object code file X.
RUN OBJECT/X; % Runs at priority 60.
RUN OBJECT/X;
  PRIORITY=70; % Overrides compiled-in priority
               % and runs at priority 70.

```

Another example of interaction between compile-time and run-time task equations appears under “OPTION Assignment” in Section 5.

COMPILE and BIND Statements

The following example illustrates several COMPILE and BIND statements:

```

COMPILE X/Y WITH COBOL85 LIBRARY GO;
  COBOL85 FILE CARD(TITLE = C/D, KIND = DISK);
  COBOL85 PRIORITY = 55;
  FILE F(TITLE=Y/Z);
COMPILE A WITH C SYNTAX;
COMPILE X ALGOL;
COMPILE B WITH COBOL85 ON PACK GO;
BIND X/Y WITH BINDER LIBRARY GO;
COMPILE A/B WITH COBOL85 LIBRARY;
  COMPILER PUNCLIMIT = 100;
  COMPILER PRINTLIMIT = 130;
COMPILER DATA CARD
.
.
.
? % End of CARD data

```

Compound Statement

<compound statement>

— BEGIN — <statement list> — END —————|

Explanation

A compound statement groups one or more statements as a logical entity. Refer to [Section 3, Job Structure](#), for the syntax and explanation of a statement list.

Example

The following is an example job that uses the compound statement:

```

IF T IS COMPLETEDOK THEN
  BEGIN
    RUN X;
    RUN Y;
  END

```

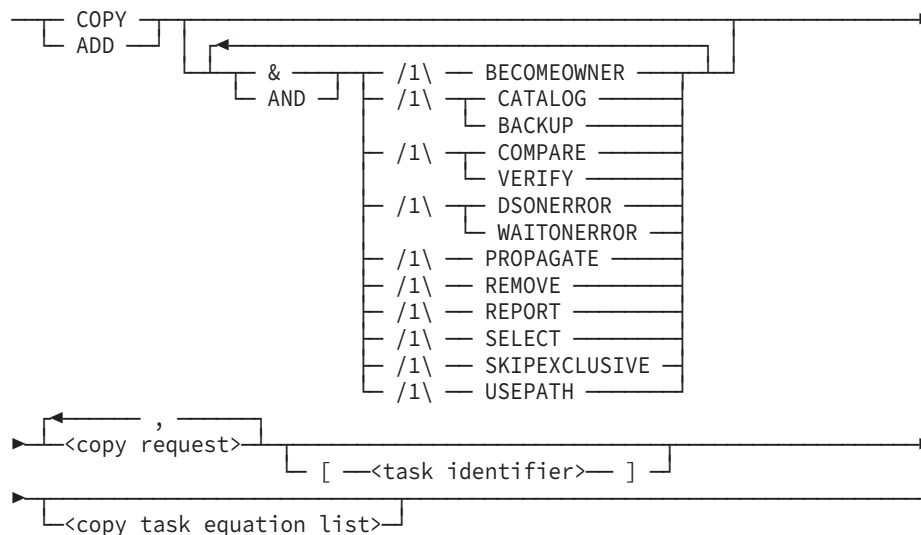
```

ELSE
  BEGIN
    COPY T/INPUT AS T/INPUT/SAVE;
    ABORT;
  END;

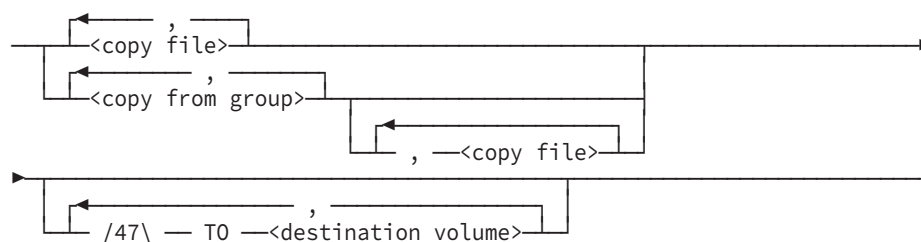
```

COPY or ADD Statement

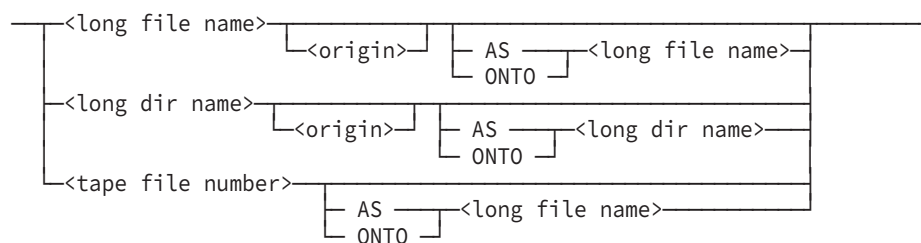
<copy or add statement>



<copy request>



<copy file>



<copy from group>



<origin>

— ORIGIN —<family name>_____

<tape file number>

— # —<integer constant>_____

<source volume>

—<volume name>_____

└─<source volume attribute list>┘

<destination volume>

— <volume name> _____

└─<destination volume attribute list> ┘

Explanation

The following text describes the various components of the COPY or ADD statement. The information is organized according to its frequency of use and importance.

The COPY statement copies disk files from disks, tapes, or CD-ROMs to disks, tapes, or CD-ROMs. Some of the uses of the COPY statement are as follows:

- Copying disk files onto tape or CD-ROM volumes or onto other disk families for safety backup. You can also selectively restore some or all of these files back to disk.
- Copying disk files from one disk family to another.
- Copying new software and disk files from tape or CD-ROM volumes received from Unisys and other installations to your disks, or copying files to tape or CD-ROM volumes for transfer to other installations.
- Copying disk files from one host system to another within a network.
- Copying CD-ROM images to write-once CD-ROMs.

Note: When copying to a CD-ROM, use the MULTIVOLUME attribute if more than one CD-ROM is required due to the amount of data. Refer to the MULTIVOLUME description in "Additional File Attributes" later in this subsection. The ADD statement, like the COPY statement, copies disk files between disks and tapes. The ADD statement has the following effects based on the specified destination:

- For a disk destination, it copies only those files that are not already resident on the specified destination disk.
- For a tape, it is equivalent to a COPY statement with a tape destination. If there were any files on the destination tape, they are overwritten. The ADD statement copies all the available requested files to the destination tape or CD-ROM volume.

Note: The ADD statement is not valid for CD-ROM volumes.

The ADD statement is particularly useful for adding a directory of files to a disk where some of the files are already resident and are to be preserved.

Refer to the MOVE statement for further information regarding moving files from one disk to another. Refer to the RESTORE statement for information regarding copying files from tape to disk. Refer to the REPLACE statement for information on replacing existing files to disk (but not for adding new files to disk).

For all of the following topics, unless otherwise mentioned, explanations about the COPY statement also apply to the ADD statement.

<copy from group>

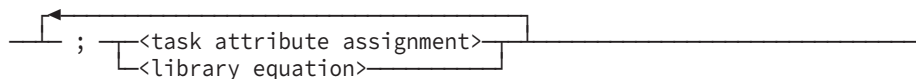
Enables you to specify one or more source volumes. The source volume applies to the entire list of file names and directory names in the copy from group construct. It is best to group files to be copied from the same tape volume into one copy from group construct, because the tape is rewound for each copy from group construct operation if the same tape volume is specified more than once.

Using Task Variables

The task identifier construct is used to assign a task variable to a COPY statement. The TASKVALUE attribute can be tested to determine the success or failure of the copy. When the value of the TASKVALUE attribute is 0, all requests were satisfied, possibly with retries. When the value of the TASKVALUE attribute is not 0, one or more files were not copied. A nonzero value is also returned if the copy is discontinued, or in the case of an RSVP condition, if the failing action is not retried successfully.

Copy Task Equation List

<copy task equation list>



Task Attribute Assignment

Assigns task attributes to the library maintenance task. For a complete explanation of task attribute assignments, refer to Section 5 in this manual. For information on using the TASKSTRING task attribute when copying to CDs, see "Copying Files to CD-ROMs" later in this manual.

Library Equation

Changes the attributes of libraries. You can use library equation to change the REPORTER library used by library maintenance for the REPORT and SELECT options.

```
LIBRARY REPORTER (TITLE = <file title>, LIBACCESS = BYTITLE)
```

For a statement that specifies the REPORT option or the SELECT option, library maintenance normally calls the SL ARCHIVESUPPORT library. Library-equating REPORTER causes the library maintenance task to use the <file title> library instead of the standard SL ARCHIVESUPPORT library.

Copying Files

When you copy files involving disks, tapes, or CD-ROMs on the local host (that is, the COPY statement does not include any HOSTNAME specifications), the copy process is done by the internal system program called *LIBRARY/MAINTENANCE which is described in the following section. When you copy files involving disks, tapes, or CD-ROMs on other host systems, the copy process is handled by a file transfer service.

The COPY statement can copy files from the following types of media:

- Disks
- Tapes in library maintenance format
- CD-ROMs in library maintenance format

The COPY statement can copy files to the following types of media:

- Disks
- Tapes
- CD-ROMs

Notes:

- *The COPY statement cannot copy files from tapes that were written by programs other than library maintenance, or from CD-ROMs that are not in library maintenance format. Disks do not have this restriction.*
- *When copying to a CD-ROM, use the MULTIVOLUME attribute if more than one CD-ROM is required due to the amount of data. Refer to the MULTIVOLUME description in "Additional File Attributes" later in this subsection.*

When the COPY or ADD statement operates on a file with a FILEKIND of FIFO, and you either copy or add files within the same host or you copy or add files using the NFT file transfer service, then a description of the FIFO is copied. However, any queued data is not copied.

When the COPY or ADD statement operates on a file with a FILEKIND of FIFO, and the statement uses other file transfer services such as Host Services, then the FIFO is not copied and an error occurs.

For further information about the features and limitations of the various file transfer services, see "COPY File Transfer Services" later in this section. Most of the descriptions of the COPY or ADD statement apply to both library maintenance and file transfers.

Library Maintenance

Library maintenance refers to the name of the internal system program that performs the copy process, called *LIBRARY/MAINTENANCE. When you use the COPY statement to copy a file to tape or CD, the tape or CD-ROM volume to which a file is copied is marked as a library maintenance tape or CD-ROM.

To copy files to or from ordinary tapes, use the DUMPALL utility. Normal programs cannot read files directly from tapes that are written using the library maintenance program or from CD-ROMs in library maintenance format, nor can they create tape or CD-ROM volumes in library maintenance format.

To list the names of the files on a library maintenance tape or CD-ROM, use the system FILEDATA utility program or the TDIR system command. See the *System Software Utilities Operations Reference Manual* and the *System Commands Reference* for descriptions of the DUMPALL and FILEDATA utilities and the TDIR command.

For tapes with a LIBMAINTDIR file to which you have appended files by using the LIBMAINTAPPEND=TOEND option, use the following FILEDATA syntax to get a printed list of the files on the tape:

```
RUN *SYSTEM/FILEDATA ("FILENAMES: LIBMAINTDIR=<file title>");
```

The library maintenance program checks for errors and discrepancies while copying, comparing, or verifying files. Whenever library maintenance detects an error, it issues an error message. Errors can cause library maintenance to do one of the following:

- Stop all copying and terminate.
- Stop copying from a particular source or to a particular destination.
- Stop copying a particular file.
- Issue an RSVP message asking if the file should be recopied.

Certain I/O errors or data discrepancies can generate a message that requires an answer from the operator before continuing, such as whether a given file should be recopied, copied with errors as a BADFILE, skipped, or whether a tape with a bad copy of a file should be purged or the file skipped.

The recopy facility is not available when copying to or from quarter-inch cartridge tapes.

To check the progress of a COPY or ADD statement, use the HI (Cause Exception Event) system command. A command of the form <mix number> HI displays the number of files already copied and other information.

Notes:

- *In most cases, library maintenance takes special action whenever you attempt to copy KEYEDIOII files from disk. Library maintenance waits until the files are no longer being updated and then blocks usage of the files and any related files, such as audit and key files, until all the specified files are copied. However, if you use the AX (Accept) system command on the KEYEDIOII/LIBRARY to set COPYINQONLY to FALSE, library maintenance immediately copies the KEYEDIOII files. (See the System Commands Reference for details on the AX command and the KEYEDIOII Programming Reference Manual for details on the COPYINQONLY runtime parameter.)*
- *Library maintenance takes special action to ensure that the copies of the various files making up a KEYEDIOII set are consistent with each other. In general, when you copy*

a KEYEDIOII file, it is advisable to copy all the related files in the same <copy request>. If the file is part of a <copy from group> then all the related files should be in the same <copy from group>.

To copy a SYSTEMDIRECTORY file, you must specify its file name in full. For example, the statement `COPY *SYSTEMDIRECTORY/= AS SYSDIR/= FROM . . .` does not copy a system directory file but the following statement does:

```
COPY *SYSTEMDIRECTORY/001 AS SYSDIR/1 FROM
```

When library maintenance makes a copy of a SYSTEMDIRECTORY file, it erases the special system file mark from the copy. System directory files with the system file mark cannot be removed with the WFL REMOVE statement or be renamed with the WFL CHANGE statement.

To copy a JOBDESC file, which is a file with a FILEKIND attribute value of JOBDESCFILE, you must specify the file name in full. For example, the statement `COPY *= FROM . . .` does not copy a job description file but the following statement does:

```
COPY *JOBDESC FROM . . .
```

When library maintenance makes a copy of a JOBDESC file, it changes the FILEKIND attribute value of the copy to DATA.

COPY Options

You can specify the following different options in a COPY statement:

- | | |
|---------------|-----------------|
| • BACKUP | • REMOVE |
| • BECOMEOWNER | • REPORT |
| • CATALOG | • SELECT |
| • COMPARE | • SKIPEXCLUSIVE |
| • DSONERROR | • USEPATH |
| • FROMSTART | • VERIFY |
| • PROPAGATE | • WAITONERROR |

BACKUP

Description

Places a backup reference in the system catalog directory of the source disk for each cataloged file that is copied.

Constraints

- Applies only to a cataloging system.
- If you specify AS or ONTO with the BACKUP option, you get a WFL syntax error.

- If you specify a tape source, you get a WFL syntax error.
- You cannot use this option with CD-ROM sources or destinations.
- The source and destination volumes must be listed in the volume library before the BACKUP option can be used.
- You cannot use this option with FTP, NFT, or Host Services File Transfer.

BECOMEOWNER

Description

Controls the ownership of the destination directories and files moved within the permanent directory namespace and controls the ownership of the destination backup files in the reserved backup directories. This option causes the OWNER attribute to be set to the usercode of the task performing the operation rather than having the value copied from the source.

All other attribute values, including those for GROUP and SECURITYMODE, are copied from the source. If a nonprivileged user issues a COPY or ADD statement without the BECOMEOWNER option, only the source directories and files already owned by that user are included.

Constraints

- Applies only within the permanent directory and reserved backup directory namespaces.
- You cannot use this option with FTP, NFT, or Host Services File Transfer.

CATALOG

Description

Marks copied files as cataloged files and lists the source versions as backup copies in the catalog directory for the destination disk.

If you use the Native File Transfer (NFT) Service to transfer the files between hosts, the source versions are not marked as backup copies at the destination host, but the destination files are marked as cataloged files.

The source and destination volumes must be listed in the volume library of the applicable source or destination host before you can use the CATALOG option.

Constraints

- Applies only to a cataloging system.
- If you specify AS or ONTO with the CATALOG option, you get a WFL syntax error.
- If you specify a tape destination, you get a WFL syntax error.
- You cannot specify a CD-ROM destination.

- You cannot use this option with FTP or Host Services File Transfer.

COMPARE

Description

Ensures that the new copies of the file are written correctly.

Ensures that the new copies of the files are readable, and that the data in the files matches the data in the source files.

Compares the copied file and the original file bit by bit, immediately after the file is copied.

Gives you a chance to approve a recopy if an error occurs while comparing a file.

Constraints

- Not available for quarter-inch cartridge tapes; use the VERIFY option instead.
- Not available with Open Systems Interconnection (OSI), Transmission Control Protocol/Internet Protocol (TCP/IP) File Transfer Protocol (FTP), or Host Services File Transfer (HS).

DSONERROR

Description

Causes library maintenance to terminate with a DS response whenever

- A file or directory to be copied is missing, and library maintenance issues a "<filename> FILE NOT ON <source volume>" message.
- An error prevents a file from being copied to one or more destinations.
- Library maintenance issues a "RECOPY REQUIRED" RSVP, and the operator replies with DS, FR, or OF.

If a COPY or ADD statement terminates abnormally when the DSONERROR option is used, library maintenance purges all of the output tapes it is currently copying.

If a COPY or ADD statement terminates abnormally and multivolume output tapes are being used, library maintenance purges only the current volume.

In the case of multiple output tapes, an error termination causes all destination volumes to be purged.

Example 1

```
COPY & DSONERROR F1 TO T1, TO T2;
```

An error termination causes both T1 and T2 to be purged.

In the case of multiple copy requests, an error termination causes one of the destination tapes to be purged.

Example 2

```
COPY & DSONERROR F1 TO T1, F2 TO T2;
```

An error termination causes either tape T1 or tape T2 to be purged, but not both tapes.

If a COPY or ADD statement with both the BACKUP and DSONERROR options terminates abnormally, then in addition to purging any output tapes, library maintenance erases or deletes any catalog backup information for files that refer to the purged tapes.

In the case of multivolume output tapes, library maintenance erases or deletes the catalog backup information for files that reference any of the volumes in the specified set of tapes.

Constraint

You cannot use this option with FTP, NFT, or Host Services File Transfer.

FROMSTART

Description

Overrides any file transfer resumption if the transfer had been previously interrupted, and instead completely transfers again all of the files, including those already transferred with the NFT Service before the interruption.

Constraints

This option applies only when NFT is used to transfer files between hosts.

PROPAGATE

Description

Enables copied files and permanent directories to inherit the security of the destination directory.

Constraints

- This option applies only when the destination PROPAGATESECURITYTOFILES or PROPAGATESECURITYTODIRS attribute of the directory so specifies.
- You cannot use this option with FTP, NFT, or Host Services File Transfer.

REMOVE

Description

Causes library maintenance to remove files from the source disk being copied. Library maintenance removes source files even if they are in use by another program.

- If you specify the VERIFY option when copying to tape, library maintenance verifies the destination tape before removing the source files.
- If you specify the DSONERROR option, library maintenance delays removing the source files until all the files have been successfully copied.
- If a program updates or replaces a source file before library maintenance removes it, then library maintenance removes the updated or replaced version of the file.

Constraints

You cannot use this option with tape or CD-ROM sources, with NFT specified, or in any host to host transfer.

REPORT

Description

Causes library maintenance to print a report of the files it copied and any errors encountered. When & REPORT is specified, library maintenance does not write "file copied" messages in the job log or the system sumlog.

Constraints

You cannot use this option with FTP, NFT, or Host Services File Transfer.

SELECT

Description

Causes library maintenance to call the LIBMAINTSELECTOR procedure in the ARCHIVESUPPORT library for each file to be copied. You or your site can code a special version of the LIBMAINTSELECTOR procedure that indicates which destinations, if any, a file should be copied to.

For information on how to code a custom LIBMAINTSELECTOR procedure, refer to the SYMBOL/ARCHIVESUPPORT DCALGOLSYMBOL file and the MCP System Interfaces Programming Reference Manual.

Constraints

You cannot use this option with FTP, NFT, or Host Services File Transfer.

SKIPEXCLUSIVE

Description

Causes the system to not copy those files from disk that are opened with EXCLUSIVE=TRUE or that are KEYEDIOII files marked as being updated.

Constraints

You cannot use this option with FTP or Host Services File Transfer.

USEPATH

Description

Causes the DATAPATH attribute to be used when resolving nonqualified file names. This includes file names without a usercode or system (*) prefix and without a family name or with a family name of DISK.

Constraints

Only the first element in the DATAPATH attribute string is used. Substitution is only applied when the source volume is disk. It is not used if the source is tape or CD-ROM. You cannot use this option with FTP, NFT, or Host Services File Transfer.

Examples

The following example shows the use of the USEPATH modifier to copy the files *DIR/SHARED/PROJECTFILES/MYFILE from PROJECTPACK and *YOURFILE from DISK to tape:

```
COPY & USEPATH MYFILE, *YOURSELF TO BACKUPTAPE;  
    DATAPATH = *DIR/SHARED/PROJECTFILES ON PROJECTPACK;
```

The following example shows the interaction of the USEPATH modifier with the AS library maintenance option:

```
COPY & USEPATH A AS D/E, B AS *F/G
```

The command in the previous example when run with the following DATAPATH attribute copies the files.

```
DATAPATH = *DIR/SHARED/PRODUCTION ON SHAREDFILES, (*)ON DISK
```

The following is the result:

```
*DIR/SHARED/PRODUCTION/A  
    (retitled *DIR/SHARED/PRODUCTION/D/E) and  
*DIR/SHARED/PRODUCTION/B (retitled *F/G)  
    from SHAREDFILES to SHAREDFILES
```

VERIFY

Description

Ensures that the new copies of the file are written correctly and readable, and the data in them is not garbled.

Verifies files by calculating the checksums for those files as they are copied, and compares those values as follows:

- Disk source
No checksum is calculated for files read from disk. Library maintenance does not verify files read from disk.
- Disk destination
Library maintenance calculates a checksum for a file as the file is copied to disk. The verify process re-reads the file from disk when the copy is completed; calculates the checksum of the new file; then compares that value with the value calculated while writing the file to disk. If a mismatch or I/O error occurs, library maintenance issues an error message, and it might issue a recopy RSVP message.
- Tape source
The copy procedure calculates a checksum for the file as it is read from the tape. That value is compared with the checksum record, at the end of the file, read from the tape. If the VERIFY option was not specified when the tape was created, there is no checksum record on the tape and library maintenance issues a warning.

If a mismatch occurs, library maintenance issues an error message; it might also issue an RSVP message.
- Tape destinations
The copy procedure calculates a checksum for each file as the file is written to the tape. After each file an extra record containing a checksum is written on the tape. When all the files have been written to the tape, the tape is rewound, and the verify process starts. The verify procedure reads each file from the tape and calculates a checksum for the file. If that checksum does not match the checksum on the tape, or if there are I/O errors while reading from the tape, library maintenance issues an error message, and it might issue a special RSVP message to check whether the tape is to be purged or the bad file is to be skipped.
- CD-ROM sources or destinations
Library maintenance does not calculate a checksum when reading from or writing to CD-ROMs.

Constraints

This option is not available with FTP or Host Services File Transfer.

WAITONERROR

Description

Causes library maintenance to issue an RSVP message whenever an error occurs. Examples of possible errors include

- Requesting a file or directory that is missing
- Failing an attempt to open a tape

The RSVP message halts library maintenance until the operator or programmer responds with either OK or DS.

A response of OK causes library maintenance to continue the COPY with other files or tapes.

A response of DS will terminate the library maintenance program. After investigating the error which created the RSVP message, you can re-issue the COPY or ADD statement.

Constraints

You cannot use this option with FTP, NFT, or Host Services File Transfer.

COPY Request

Basic Copy Request

A basic copy request consists of a COPY or ADD verb, followed by a list of files and/or directories to be copied, an optional FROM <source volume> specification, and an optional TO <destination volume> specification.

FROM

You can optionally include a FROM source volume phrase that designates from where the files are to be copied. If you do not specify a FROM source volume construct, the files are copied from the disk family named DISK.

TO

You can optionally include a TO destination volume phrase that designates where the files will be copied. If you do not specify a TO destination volume construct, the files are copied to the disk family named DISK.

Family Substitution

Family substitution occurs for one or more of the source and/or destination disk family names specified in the COPY statement when either of the following conditions exist:

- The target family name in the FAMILY assignment statement matches the source and/or destination disk family names specified in the COPY statement.
- You omitted either the FROM source volume or the TO destination volume phrase in the COPY statement, and the implied family name DISK matches the name DISK in the FAMILY statement.

Library maintenance never uses the name supplied after OTHERWISE in a family substitution statement.

COPY Constructs

The copy file, source volume, destination volume, and volume name constructs are described in the following material.

<copy file>

Specifies the names of the files or directories to be copied. If you use the AS option, specifies the names to which the files or directories should be copied. If you do not use the AS option, the output file names are exactly the same as the input file names. Output file names might differ from input names. The following conditions also apply when you use the AS option:

- If you specify a universal file name, the new copy of the file is given the new name you specify.
- If you specify a directory name, then the new copy of each file copied from that directory is given the new directory name you specified instead of the old directory name, and the portion of the file name that follows the old directory name is retained.
- If you specify a file or directory name, if the copy process runs under a usercode and you do not include a usercode or an asterisk (*) with the new file name or new directory name, the new file names will all be prefixed with the usercode under which the copy process is running. The ONTO option is limited to use with data files whose integer value of the FILEKIND attribute is greater than or equal to 64.

The ONTO option cannot be used for object code files, compilers, or system files. If the NOCOPYONTO security option is set, then the ONTO option cannot be used for any file. The NOCOPYONTO option is available only on a system that contains the Security Accountability Facility software.

<source volume> <destination volume>

Specifies the source volume of the files to copy and the destination volume(s) for those files. If the source or destination volume is not specified, the volume DISK is assumed. If all the copy file constructs contain an AS name, you do not have to specify a source or destination volume. If more than one destination volume is specified, all files are copied at the same time and in the same order, to all destination volumes, thus creating multiple copies. In addition, you can indicate certain file attributes and tape volume attributes for files being copied from a source volume or to a destination volume with the source volume attribute list and destination volume attribute list constructs.

<volume name>

Indicates the name of the volume from which or onto which the files are copied. If the volume name is DISK or PACK, the default file kind is DISK; otherwise, the default file kind is TAPE. If # <string primary> is specified for the volume name, the default file kind is TAPE. When the volume name includes a <string primary>, embedded periods will terminate the volume name unless the entire name is enclosed in quotes. If the file kind is DISK, the volume name cannot contain a period. A volume name cannot contain a period for a file kind of DISK.

Family substitution is used if the file kind is DISK and the job or task has an active family specification. Only the primary substitute family name is used by Library Maintenance and NFT. (Refer to [FAMILY Assignment](#) and [Interrogating Complex Task Attributes](#).)

<tape file number>

Specifies the position number of the file on the source tape or source CD-ROM. The position number of the first file on a tape is 1, and so forth. The <tape file number> must be a positive integer that is greater than zero (0) and has less than 12 digits (counting leading zeros).

To determine the tape file number of a particular file on a CD or a tape that does not have an associated LIBMAINTDIR directory, use the TDIR system command or use FILEDATA:

```
RUN *SYSTEM/FILEDATA ("TAPEDIR: <volume name>")
```

or

```
RUN *SYSTEM/FILEDATA ("TAPEDIR: <volume name> CD")
```

To determine the tape file number of a particular file on a CD or tape that has an associated LIBMAINTDIR directory, use FILEDATA:

```
RUN *SYSTEM/FILEDATA ("NAMELIST: LIBMAINTDIR=<file title>")
```

The <file title> variable is the file title of the LIBMAINTDIR file associated with the tape or CD.

Note: To see the file numbers in the FILEDATA LIBMAINTDIR report, you can use the output option PRINTER. If you use the output option SCREEN, SPO, TTY, or FILENAME, then you need to include a DEFINEOUTPUT request that sets LINEWIDTH to at least 81.

For more information on using FILEDATA, see the *System Software Utilities Operations Reference Manual*.

You can use the tape file number in ordinary library maintenance copy statements and in NFT file transfer copy statements. You cannot use the tape file number with other file transfer protocols. You cannot specify the ORIGIN family name for a tape file number.

ORIGIN Clause

When copying from tape or CD-ROM, you can use the ORIGIN clause to select those files that were originally copied to tape or CD-ROM from the specified disk family. This can be useful if you previously copied files with the same file names from different disk families to the same tape or CD-ROM. For example, the following form of the statement copies two versions of SYSTEM/ALGOL to the SA tape:

```
COPY SYSTEM/ALGOL FROM PACK, SYSTEM/ALGOL FROM SYS (PACK) TO SA;
```

You can then retrieve the version of the file that was copied from either one of the disk families by specifying the disk family name as in the following statement:

```
COPY SYSTEM/ALGOL ORIGIN SYS FROM SA TO NEWSYS (PACK),  
                                TO SYSTAPE;
```

You can use ORIGIN in the file list for a FROM clause that refers to a tape or CD-ROM source. You cannot, however, use ORIGIN for disk sources. You can use ORIGIN with NFT file transfer requests but not with other kinds of file transfer requests.

You cannot use ORIGIN to select files on tapes if the tapes do not have origin information. Tapes created before SSR 41.2 and tapes created by the NFT file transfer service before SSR 50.1 do not have origin information. Any file copied from a tape or CD-ROM without origin information to another tape or CD-ROM does not have origin information either. You can use the FILEDATA TAPEDIR report to check which files on a tape or CD-ROM has origin information.

ONTO Clause

Note: The ONTO clause was originally intended for copying data to Installation Allocated Disk (IAD) files. Because IAD files are no longer supported, there is little reason to use the ONTO clause.

Disk Destinations

The ONTO clause changes the handling of cases where a file of the requested name already exists at a disk destination. In such a case, library maintenance

- Opens the existing disk file at the destination. (If no file of the same name exists at the destination, library maintenance does not perform the copy.)
- Copies data from the source file to the existing destination file.

By contrast, when ONTO is not specified, library maintenance normally does the following:

- Creates a new file on disk.
- Copies the data and file attributes from the source file to the new file.
- Locks the new file into the directory. If a permanent file of the same name already exists at the destination, the system removes that file when the new file is locked.

Tape and CD-ROM Destinations

For tape and CD-ROM destinations, the ONTO clause works like an AS clause. Library maintenance always writes a new file on the tape or CD-ROM.

Caution

When you use the ONTO clause, the integrity of the copy is not assured. If library maintenance terminates while copying data onto an existing disk file, the disk file might be left with some data copied from the source file and some data leftover from the destination file.

Example

```
COPY F ONTO OLDF, D/= ONTO OLDDD/=  
FROM DPMAS(T(PACK) TO UDDOCS(PACK)
```

Copies file F from DPMAS family to UDDOCS family. Library maintenance copies the source F contents onto the existing destination file OLDF ON UDDOCS.

All files are copied in directory D/= from DPMAS family to UDDOCS family. For each file, if a file of the same name already exists on UDDOCS family, the source file contents are copied onto the existing destination file.

Restrictions

The WFL compiler returns a syntax error if the ONTO clause is used in a COPY & BACKUP or a COPY & CATALOG statement.

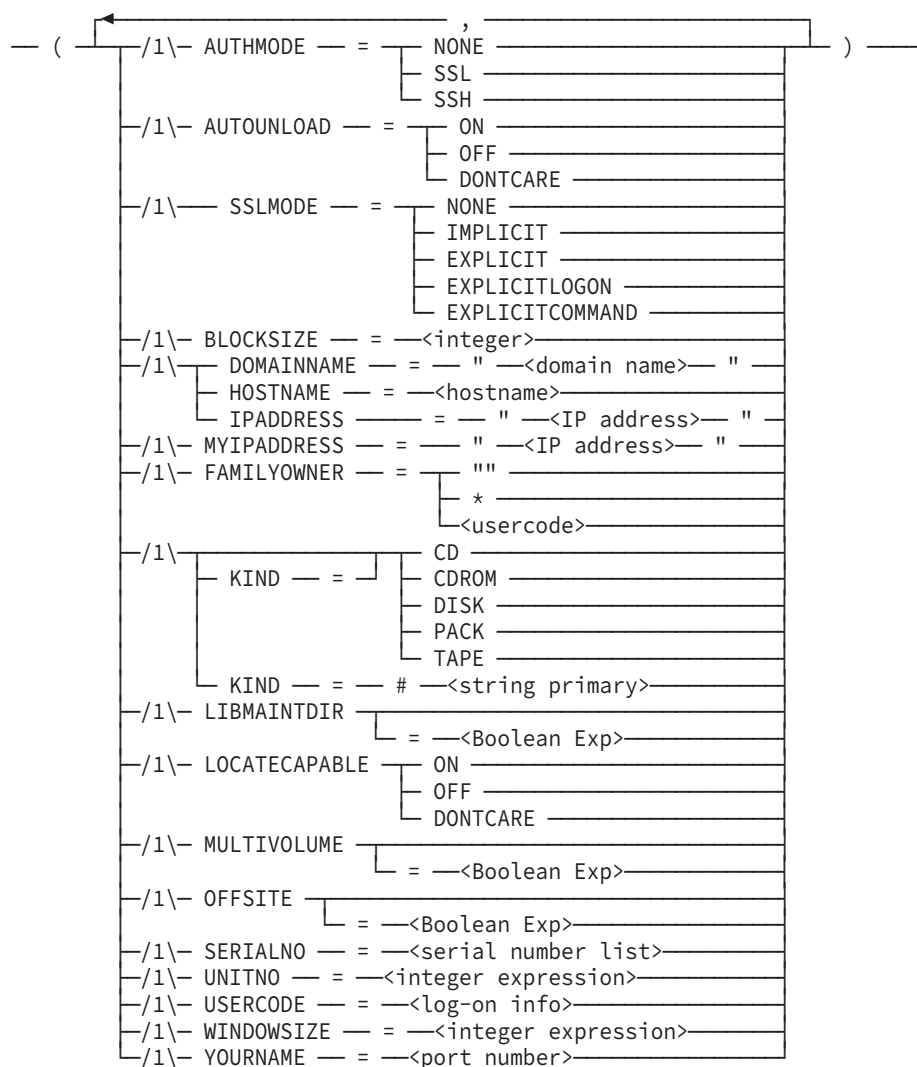
Library maintenance will not copy a file onto an existing disk file if any of the following restrictions are violated:

Type of Restriction	Details
AREAS, AREASIZE, FILESTRUCTURE	Values for the source and destination file must match.
EOFBITS, EOFSEGMENT	Values for the source and destination file must match if the destination file is crunched.
Areas allocated	Any allocated area of either the source file or the destination file must also be allocated in the other.
Destination file usage	Cannot be in use by any program.

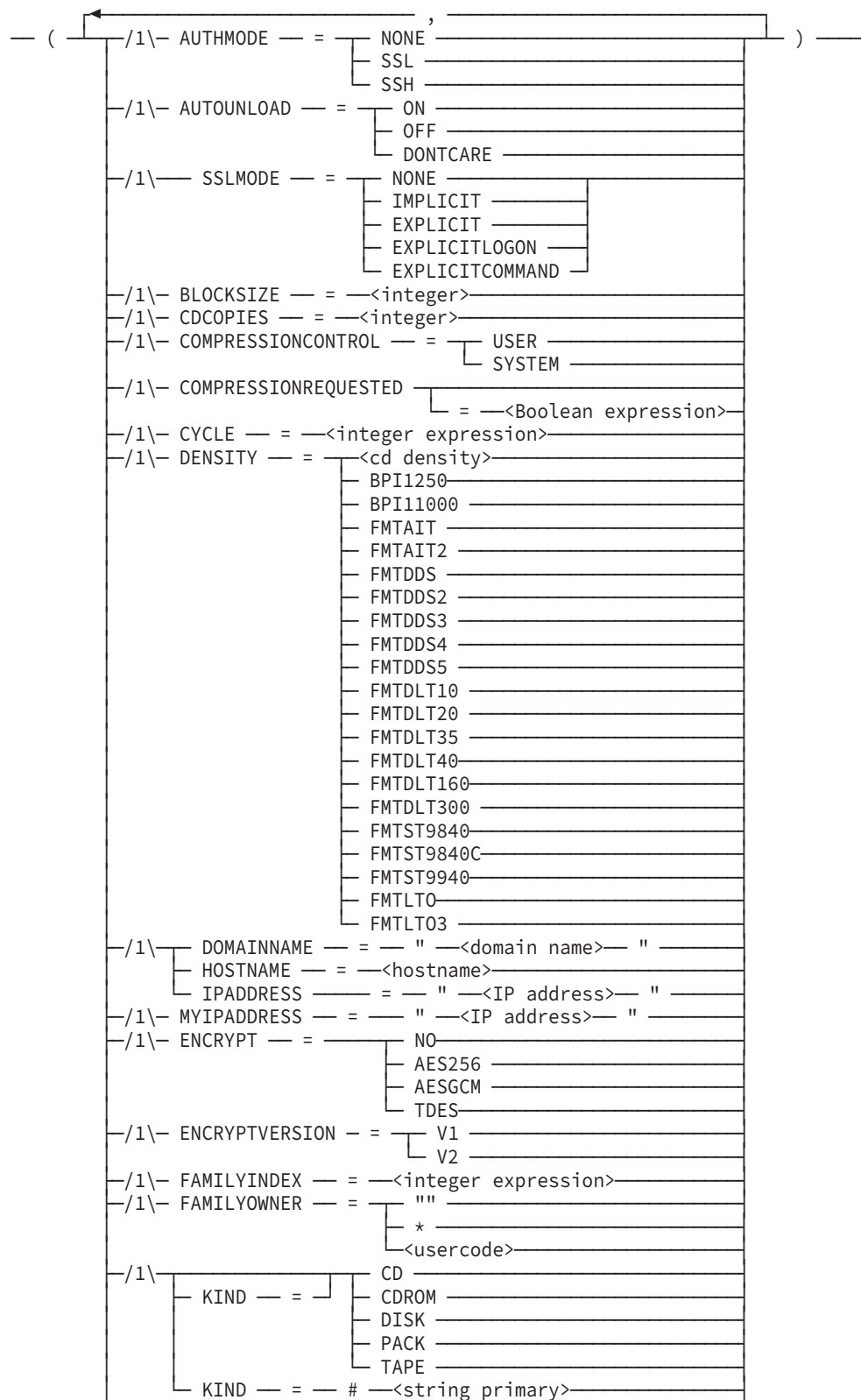
Type of Restriction	Details
Destination file characteristics	<p>Cannot be</p> <ul style="list-style-type: none"> Object code file BADDISK file Printer backup file File status marked as nonremovable SYSTEMFILE Any other special system file

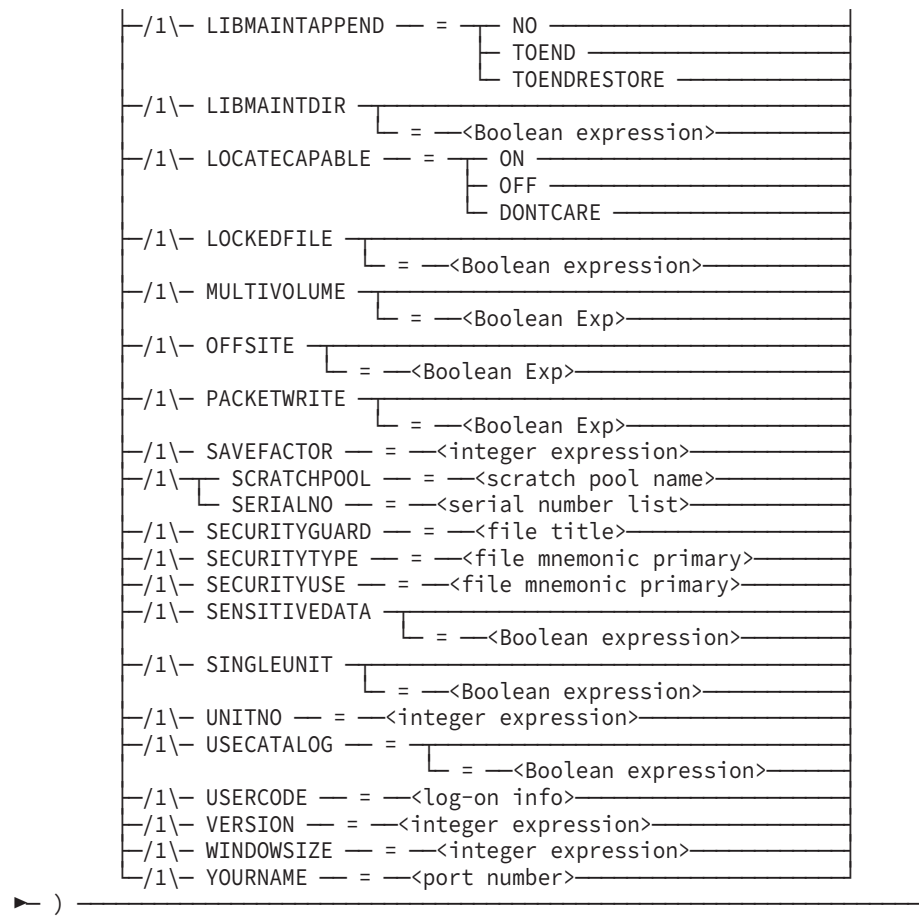
Attribute Lists

<source volume attribute list>

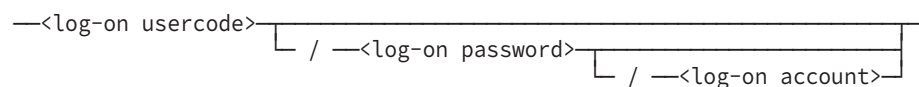


<destination volume attribute list>





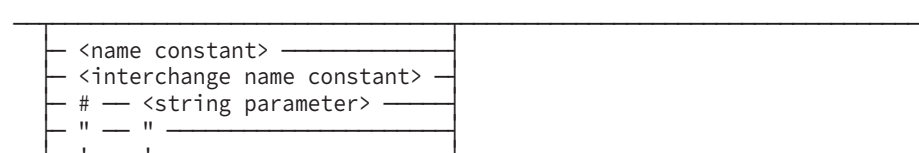
<log-on info>



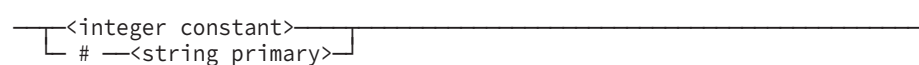
<log-on usercode>

<log-on password>

<log-on account>



<port number>



You can use the # <string primary> syntax to dynamically specify a port number. The <string primary> must evaluate to an integer.

Explanation

When you use the COPY statement, you can indicate certain file attributes for files being copied and certain file attributes for each source volume or destination volume. For more specific information on an attribute, refer to the *File Attributes Programming Reference Manual*.

You can specify the following file attributes for either the source volume attribute list or the destination volume attribute list:

- | | | |
|---------------|-----------------|--------------|
| • AUTHMODE | • IPADDRESS | • SERIALNO |
| • AUTOUNLOAD | • KIND | • SSLMODE |
| • BLOCKSIZE | • LIBMAINDIR | • UNITNO |
| • CYCLE | • LOCATECAPABLE | • USERCODE |
| • DOMAINNAME | • MYIPADDRESS | • WINDOWSIZE |
| • FAMILYOWNER | • OFFSITE | • YOURNAME |
| • HOSTNAME | • SCRATCHPOOL | |

AUTHMODE

You can use the AUTHMODE attribute in FTP file transfer requests only. When you use AUTHMODE=SSL, the FTP system uses the Secure Socket Layer encryption algorithm for data transfers. If set to SSL, the SSLMODE attribute defaults to implicit SSL. When you use AUTHMODE=SSH, the FTP system uses the SFTP file transfer protocol over the SSH secure shell protocol. Explicitly setting AUTHMODE overrides any setting in the FTP configuration file or an FTP startup file.

AUTOUNLOAD

Determines whether or not a tape is unloaded when it is released by the system during a reel switch or a file close operation. If the value is ON, the tape is rewound and unloaded. If the value is OFF, the tape is not unloaded. If the value is DONTCARE, or if this value is not specified, the tape behavior is controlled by the setting of the AUTOUNLOAD option of the MODE (Unit Mode) system command. Refer to the System Operations Guide for further information. If a reel is switched during a COPY and COMPARE operation, intermediate reels are not unloaded, regardless of the AUTOUNLOAD value, until the COMPARE phase is finished.

BLOCKSIZE

Specifies the blocksize in words that library maintenance uses when copying files to or from disk or tape volume. Library maintenance automatically rounds the value you specify up to an integer multiple of 900 words. If you specify a value larger than 64800 words, library maintenance uses 64800 instead. If you specify a value greater than 4500 words for a tape destination, then library maintenance automatically uses that blocksize for all disks involved in the operation.

Using the same block size for the source and all the destinations greatly improves the performance of the copy task. Using a larger blocksize increases the amount of data that can be stored on any given tape volume.

Notes:

- *Certain magnetic tape devices have limits on the maximum length I/O they can process. If you specify a block size larger than the limit for a tape device, library maintenance automatically reduces the blocksize to a valid value for that tape destination.*
- *If you do not specify BLOCKSIZE for a tape, library maintenance uses the default value established by the SYSOPS LMBLOCKSIZE system command; or if the default is zero (0), library maintenance uses a small default blocksize that it selects for each tape and disk.*
- *Whenever you copy from a tape or CD-ROM, library maintenance will use the BLOCKSIZE that was used when the tape or CD-ROM was created, and ignore the value that is specified.*

The following table lists the maximum block sizes for the specified tape drives.

Tape Drive Name	Density	Maximum Library Maintenance Blocksize
CLU9710-DLT4	FMTDLT10, FMTDLT20	64800 words
CLU9710-DLT7	FMTDLT10, FMTDLT20, FMTDLT35	64800 words
CLU9710-DLT8	FMTDLT20, FMTDLT35, FMTDLT40	64800 words
CTS9840, CTS9840B	FMTST9840	42300 words
CTS9840C	FMTST9840C	42300 words
CTS9940B	FMTST9940	42300 words
ALP430	FMTDDS, FMTDDS2, FMTDDS3	10800 words
ALP440	FMTDDS, FMTDDS2, FMTDDS3, FMTDDS4	10800 words
ALP450	FMTDDS3, FMTDDS4, FMTDDS5	10800 words 10800 words 64800 words

Statements

Tape Drive Name	Density	Maximum Library Maintenance Blocksize
ALP920	FMTAIT, FMTAIT2	64800 words
HS8500(C) (8mm)	11000	40500 words
LTO Gen 2	FMTLTO	64800 words
LTO Gen 3	FMTLTO3	64800 words
MA150T(U), QIC1000	1250	10800 words
SDLT320	FMTDLT20, FMTDLT35, FMTDLT40, FMTDLT160	64800 words
SDLT600	FMTDLT160, FMTDLT300	64800 words
TAL408012	FMTDDS, FMTDDS2, FMTDDS3, FMTDDS4	10800 words

In addition to these limits, any tape connected to the system through a “soft” or “emulated” SCSI DLP has a library maintenance blocksize limit of 10800 words.

Library maintenance automatically limits the blocksize it uses when writing to tape volumes on these tape units so that it will not exceed their blocksize limits.

Library maintenance complies with these limits even if you specify a SYSOPS LMBLOCKSIZE that exceeds these limits. If you are going to be copying files to a tape volume on a unit that does not have limits, but might someday need to be read on a tape unit that has limits, then you should ensure that library maintenance writes the tape with a blocksize that can be read on the target tape unit.

CYCLE

Designates the specific generation of a tape volume family. This file attribute is used in conjunction with the VERSION attribute. The default value is 1.

DOMAINNAME

Indicates the domain name of the remote host where the source or destination volume is located. This can be used for FTP only.

FAMILYOWNER

Indicates the usercode of the owner of a tape volume. If you specify a usercode with the FAMILYOWNER attribute, the usercode becomes the owner. If you do not use the FAMILYOWNER attribute or you specify a null string (“”), the usercode of the copy process is used. If you specify an asterisk (*) with the FAMILYOWNER attribute, the tape volume becomes a nonusercoded volume.

The FAMILYOWNER attribute can be used only if the Security Accountability Facility software is installed, and the TAPECHECK option of the SECOPT system command is set to AUTOMATIC. This attribute is ignored if the SECOPT TAPECHECK attribute is set to NONE.

HOSTNAME

Indicates the host name of the remote host where the source or destination volume is located. The default value is the name of the local host. The HOSTNAME attribute can be specified for the source or destination volume to copy files from or to a remote host; or it can be specified for both source and destination volumes to enable files to be copied from one foreign host to another.

Note: *Although the local host name is the default value, assigning the local host name to the HOSTNAME attribute might not produce the same results as omitting the attribute. If the assignment is through a constant (HOSTNAME = <local host>), then the assignment is ignored, and the behavior is the same as if the HOSTNAME attribute had not been specified. If the assignment is through a variable (HOSTNAME = #HN where HN is a string variable with a value of "<local host>"), the COPY or ADD statement is handled by DSS, which imposes file transfer restrictions on the request (for example, a CD destination is disallowed).*

IPADDRESS

Indicates the numerical designator that uniquely identifies the remote host where the source or destination volume is located. Can be used for FTP only.

KIND

Describes the peripheral unit associated with the logical file. The default value is TAPE, unless the volume name is DISK, PACK, or CD. If you want to indicate the kind of tape to be used, use the DENSITY attribute.

LIBMAINTDIR

Determines, on destination tapes, whether library maintenance should create a tape directory disk file on the DL LIBMAINTDIR disk family. The LIBMAINTDIR tape directory disk file describes the destination tape and the files copied to it. Library maintenance gives tape directory disk files names of the form LIBMAINTDIR/<tape name>/<date>/<tape serialno>. It also puts them under the usercode that library maintenance is running under, or * if there is no usercode.

Library maintenance stores the following information in these files:

- Serial numbers of the tapes used
- Names of the files copied to those tapes
- Certain other attributes of those files

You receive a report of the information in tape directory disk files by running the SYSTEM/FILEDATA utility program using the following syntax:

<filedata modifier> LIBMAINTDIR = <disk file name>

Note: When you specify LIBMAINTDIR = TRUE library maintenance writes the tape with ANSI87 labels.

On source tapes, if you

- Specify LIBMAINTDIR = FALSE, library maintenance uses the tape directory on the tape volume itself to determine what files are on the tape.
- Specify LIBMAINTDIR = TRUE on a tape that was originally created with LIBMAINTDIR = FALSE or without any LIBMAINTDIR specification, library maintenance uses the directory on the tape to determine what files are on the tape and where the files are located on the tape.
- Specify LIBMAINTDIR = TRUE on a tape that was originally created with LIBMAINTDIR = TRUE, library maintenance uses the LIBMAINTDIR disk file instead of the directory on the tape to determine what files are on the tape and where the files are located on the tape.
- Do not specify LIBMAINTDIR on a tape that was originally created with LIBMAINTDIR = TRUE, library maintenance attempts to use the LIBMAINTDIR disk file. If the LIBMAINTDIR file is missing or if library maintenance encounters an error while reading records from it, library maintenance reverts to using the directory on the tape to determine what files are on the tape and where the files are on the tape.

Library maintenance may access the tape directory disk file for source tapes that library maintenance originally generated with LIBMAINTDIR = TRUE, if the file is resident on the disk family specified by the DL LIBMAINTDIR system command. If you specify TRUE for a source tape that was originally generated with LIBMAINTDIR = TRUE, the tape directory disk file for the tape must be resident on the disk family specified by the DL LIBMAINTDIR system command, or library maintenance will wait with a "NO FILE" RSVP.

Any attempt to purge a library maintenance tape for which there is a tape directory disk file resident on the DL LIBMAINTDIR disk family requires approval by the operator. Whenever a program or library maintenance overwrites a tape for which there is a resident LIBMAINTDIR tape directory disk file the system removes that file from the disk.

LOCATECAPABLE

Indicates that the file requires a tape drive capable of processing the READ POSITION and LOCATE BLOCK ID tape commands for fast tape access.

If the assigned tape drive is locate capable, then library maintenance automatically takes advantage of this feature to do high-speed spacing in the following situations:

- COMPARE Option
Library maintenance uses the LOCATE BLOCK ID tape command to backspace to compare the file. If you receive a RECOPY REQUIRED message and respond "OK," library maintenance uses LOCATE BLOCK ID to backspace to the beginning of that file to recopy the file. If you respond with "OF," the file is erased.
- LIBMAINTDIR for Destination Tapes
For a tape that is locate capable, library maintenance stores the BLOCK ID of the start

of each file it copies in the LIBMAINTDIR directory.

- LIBMAINTDIR for Source Tapes

If the original tape was created on a locate capable tape drive, and a LIBMAINTDIR directory was created for the tape, then library maintenance uses the LOCATE BLOCK ID information found in the LIBMAINTDIR directory to rapidly space up to each of the files to be copied.

MYIPADDRESS

Indicates the numerical designator that uniquely identifies the local host where the source or destination volume is located.

OFFSITE

When you specify OFFSITE for a tape destination, library maintenance updates the onsite/offsite status of that tape in the volume library or in the volume directory.

If library maintenance does not successfully copy a file to a destination tape, it purges the tape and does not update the onsite/offsite status for the tape.

If library maintenance successfully copies the files and OFFSITE is TRUE, then, when library maintenance closes the tape, it updates the entry for the tape in the volume library or in the volume directory to indicate that the volume or volumes are "offsite." If library maintenance successfully copies the files and OFFSITE is false, then, when library maintenance closes the tape, it updates the entry for the tape in the volume library or in the volume directory to indicate that the volume or volumes are "onsite."

Note: Library maintenance performs these actions only when the tape volume is listed in either the Volume Library, at sites that use the OP + CATALOGING option, or the Volume Directory, at sites that use the SECOPT TAPECHECK = AUTOMATIC option.

SCRATCHPOOL

Indicates the scratch pool that the tape is retrieved from when files are copied. The scratch pool name is a 17-character identifier. There is no default value.

SERIALNO

Identifies the specific disk or tape volumes to be used when copying files. To copy a few specific files from a multi-reel library maintenance tape set, you can use this attribute to skip directly to the reel holding the file or files you need. If you know which reel contains the file you want (or the first reel that contains one of the files you want), use the serial number of that tape volume for the value of the SERIALNO attribute. The SERIALNO attribute does not have a default value. For more information, refer to [Serial Number Assignment](#).

SSLMODE

You can use the SSLMODE attribute in FTP file transfer requests only. The SSLMODE can be set to use an explicit SSL transfer, an implicit SSL transfer, or a normal file transfer. Setting SSLMODE in the volume attribute overrides any setting from the FTP configuration file. Explicitly setting the AUTHMODE attribute to a value other than SSL overrides the SSLMODE attribute.

UNITNO

Designates the assigned hardware unit number of the tape volume to be used when copying files. In the case of a multi-volume tape set, UNITNO is applied only to the first volume.

USERCODE

Passes log-on information, which consists of a usercode, password, and charge account, to the remote host specified with the HOSTNAME attribute. The default value is a null string (""). The log-on information has a maximum length of 255 characters, which include all quotation marks ("), single quotation marks or apostrophes ('), and slashes (/). The USERCODE attribute is ignored if the HOSTNAME attribute is not specified, if Host Services File Transfer is used, or if NFT is used.

Note: *if the log-on information includes a password, the password and delimiting slash are erased from the job summary. If a charge account is included, it is retained.*

YOURUSERCODE is an acceptable synonym for USERCODE.

WINDOWSIZE

Indicates the maximum amount of data, in octets, that can be outstanding between the source and the destination processes involved in an NFT transfer. An octet is a unit of data that is 8 bits in length.

Note: *The WINDOWSIZE attribute is applicable only if an NFT transfer is executed. For more information, refer to the Distributed Systems Services Operations Guide.*

YOURNAME

Overrides the default value for the control port number. When you copy from a remote system, specify YOURNAME in the <source volume> attributes. When you copy to a remote system, specify YOURNAME in the <destination volume> attributes. YOURNAME must be an integer in the range 1 to 64999.

The following example copies a file to a remote system with an IP address of 1.1.1.1 and assigns a port number of 12345:

```
COPY [FTP] A_FILE AS 'a.txt' TO DISK
      ( IPADDRESS = "1.1.1.1", YOURNAME = 12345 )
```

Note: *The control port number used by the Batch Client is determined in the following order:*

- *The default value: 21*
- *The value from the Configuration File, if specified*
- *The value of the YOURNAME volume attribute, if specified*

Examples

The following statements illustrate various aspects of the COPY statement with the AUTOUNLOAD and SCRATCHPOOL file attributes.

The following statement will copy file X as A/B from disk PACK to tape T. Since the value of AUTOUNLOAD file attribute is ON, the tape will be unloaded when the operation is completed.

```
COPY X AS A/B FROM PACK TO T(KIND=TAPE, AUTOUNLOAD=ON);
```

The following statement will copy the file X as A/B from disk PACK to tape T in the TEST scratch pool:

```
COPY X AS A/B FROM PACK TO T(KIND=TAPE, SCRATCHPOOL=TEST);
```

Additional File Attributes

You can specify the following additional file attributes for the destination volume attribute list:

- | | |
|------------------------|-----------------|
| • CDCOPIES | • PACKETWRITE |
| • COMPRESSIONCONTROL | • SAVEFACTOR |
| • COMPRESSIONREQUESTED | • SECURITYGUARD |
| • DENSITY | • SECURITYTYPE |
| • ENCRYPT | • SECURITYUSE |
| • ENCRYPTVERSION | • SENSITIVEDATA |
| • FAMILYINDEX | • SINGLEUNIT |
| • LIBMAINTAPPEND | • USECATALOG |
| • LIBMAINTDIR | • VERSION |
| • LOCKEDFILE | |
| • MULTIVOLUME | |

Notes:

- *These file attributes cannot be specified for the source volume attribute list construct.*
- *The tape attributes SECURITYGUARD, SECURITYTYPE, and SECURITYUSE only work if the Security Accountability Facility software is installed and the TAPECHECK option of the SECOPT system command is set to AUTOMATIC. If the TAPECHECK option of the SECOPT command is set to NONE, these three attributes are ignored for disk and tape destination volumes.*

CDCOPIES

You can specify how many copies of a CD-R disc are to be burned with the attribute CDCOPIES. The default value of CDCOPIES is 1.

When data is copied to a CD-R disk, it is first copied from the source media to a temporary disk file. The data in the temporary disk file is formatted exactly as it is to appear on the CD-ROM. The temporary disk file is then copied to a CD-R drive. CDCOPIES causes the disk file to be copied to the CD-R drive as many times as you specify, without having to recopy the data from the source media.

COMPRESSIONCONTROL

Determines whether data compression will occur for a specific tape volume. This attribute must be selected before a tape volume is assigned.

When the value USER is selected, the value of the attribute COMPRESSIONREQUESTED will determine whether compression will occur. When the value SYSTEM is selected, compression will occur based upon the value of the compression flag in the tape label.

For further information, refer to the *File Attributes Programming Reference Manual*.

COMPRESSIONREQUESTED

This attribute only has significance for tape files if the COMPRESSIONCONTROL attribute is set to a value of USER.

If COMPRESSIONCONTROL = USER, COMPRESSIONREQUESTED will determine whether compression will occur for a tape file.

Compression will only occur if COMPRESSIONCONTROL = USER and COMPRESSIONREQUESTED = TRUE.

For further information, refer to the *File Attributes Programming Reference Manual*.

DENSITY

Indicates the recording density of a magnetic tape volume. The default value for output files is the density setting of the tape unit selected.

When the destination volume has kind CD, the DENSITY value identifies the size of the CD image that needs to be created and burnt on to the CD or DVD media. Refer to the section on DENSITY in the *File Attributes Programming Reference Manual* for the allowable values.

ENCRYPT

Specifies that library maintenance encrypt the data it writes to the specified destination CD or tape. The following three encryption algorithms are provided:

- TDES
- AES256
- AESGCM

To use AESGCM, Media Encryption Version 2 is required.

Note: *You cannot specify ENCRYPT for disk destinations.*

If you specify LIBMAINTAPPEND, then library maintenance ignores the ENCRYPT setting and instead uses the original ENCRYPT value for that set of tapes.

The LOADER cannot read encrypted library maintenance tapes.

You cannot specify ENCRYPT for a source CD or tape. Library maintenance automatically decrypts data when reading from an encrypted CD or tape, so you can use a simple COPY statement to copy files from an encrypted CD or tape:

```
COPY ... files ... FROM <tape name>;
```

ENCRYPTVERSION

Specifies the Media Encryption Version to be used. This overrides the value of the SYSOPS LMDEFENCRYPT version.

- V1
- V2

FAMILYINDEX

Designates a specific physical volume within a disk family. If you do not specify the FAMILYINDEX attribute, the FAMILYINDEX value of each source file is used (if that file has had explicit values specified for FAMILYINDEX). By specifying this attribute, you can alter or erase the FAMILYINDEX attribute of the new copies. If you specify a value of 0 or an integer expression that equals 0 for the FAMILYINDEX attribute, the disk areas will be allocated in the normal rotational order of the system.

LIBMAINTAPPEND

You can use this attribute to add files to the tape volume originally created when the LIBMAINTDIR = TRUE file attribute is set.

Notes:

- *Library maintenance does not update the tape directories on any reels already copied with the file names of the files being added. Library maintenance only updates the LIBMAINTDIR tape directory disk files with the names of the new files.*
- *The message BACKUP START ON: data and time appears in the PD display for files that have archive backups. This message refers to the time of the original task that started the tape. It does not refer to the start time of the task with LIBMAINTAPPEND = TOEND specified.*

LIBMAINTAPPEND = NO

If you specify LIBMAINTAPPEND = NO, or do not specify LIBMAINTAPPEND, the copy procedure copies to a new tape.

LIBMAINTAPPEND = TOEND

If you specify LIBMAINTAPPEND = TOEND, then library maintenance searches for an existing library maintenance tape with the name and serial number that you specify.

When using LIBMAINTAPPEND, if you specify a SERIALNO list, then the serial numbers in the list are used as follows:

The first serial number indicates to library maintenance the tape set to which files are appended. You must do one of the following:

- Specify a serial number of one of the tape volumes that is already in the tape set. The best serial number to choose is that of the last tape volume added to the set. You can also specify the serial number of the original tape volume created with LIBMAINTDIR = TRUE, or the serial number of any other tape in the set.
- Omit the first serial number.

Library maintenance uses any additional serial numbers you specify to add new tape volumes to the set, as required. Ensure that these serial numbers do not match the serial numbers of existing tape volumes in the set. If there is a conflict in serial numbers, then library maintenance automatically deletes the conflicting serial numbers from the SERIALNO list.

The tape you specify must be a tape created with the LIBMAINTDIR = TRUE specification. Library maintenance opens the LIBMAINTDIR disk file with the EXCLUSIVE file attribute TRUE. This prevents other archive, library maintenance, and FILEDATA processes from using that LIBMAINTDIR disk file while library maintenance is updating it. Library maintenance checks the LIBMAINTDIR tape directory disk file for that tape to determine the serial number of the last tape in that set of tapes. (Even if the tape contains no tape directory, it does contain a pointer to the LIBMAINTDIR file on disk, which is used to find the location of the end of the last file on the final tape.)

If necessary, library maintenance searches for that tape. Then library maintenance skips to the end of the last file copied to that tape and copies the files that you specified. The copy procedure expands the LIBMAINTDIR tape directory disk file for the tape with the names and status of all files copied to the LIBMAINTDIR file.

You cannot use the LIBMAINTAPPEND = TOEND attribute for a library maintenance tape if the most recent library maintenance that copied files to the tape terminated because of an error. If you try to use LIBMAINTAPPEND = TOEND for such a tape library, library maintenance issues the following error message and does not add any files to that tape:

```
MT<unit number> LOCATION OF THE END OF  
THE LAST FILE ON THE TAPE NOT KNOWN
```

If an error occurs that stops the copy to the tape (for example, if the operator issues a <mix number> DS command to terminate the append task), then all the files successfully copied up to that point are on the tape and are listed in the LIBMAINTDIR file. However, no more files can be appended to the tape. Any attempt to use LIBMAINTAPPEND again to add more files to that tape is rejected with an error message.

The following conditions must be met to use LIBMAINTAPPEND = TOEND:

- The destination tape must have been created by a COPY or ARCHIVE statement that specifies LIBMAINTDIR = TRUE for that tape.
- You must specify & VERIFY for the append operation if the & VERIFY option was specified when the tape was originally created.
- You must not specify & VERIFY for the append operation if the & VERIFY option was not specified when the tape was originally created.

When library maintenance copies files to a tape with LIBMAINTAPPEND = TOEND, it does not add the names of those files to the tape directory for the tape. You cannot use the TDIR system command or the FILEDATA utility TAPEDIR request to view the names of files copied to tape with LIBMAINTAPPEND = TOEND. Instead, use the FILEDATA utility LIBMAINTDIR modifier to view the names of all the files copied to a tape.

If you specify a list of tape serial numbers for the SERIALNO attribute, library maintenance takes special action. Library maintenance uses the first serial number in the list to locate any existing tapes in the set of tapes to which the files are to be added or appended. Library maintenance then reads the LIBMAINTDIR file for that tape to determine the serial number of the last tape volume in the set. Library maintenance immediately opens that tape, if necessary, and then uses the other serial numbers you supplied when it reaches the end of that tape.

Example

The original tapes have the serial number 111111, 222222, and 333333, and the following COPY statement is specified:

```
COPY . . . TO <tape name> (LIBMAINTAPPEND = TOEND,
                           SERIALNO = ( 222222 , "AAAAAA" ) );
```

The preceding COPY statement is processed as follows:

- The tape with serial number 222222 opens and the LIBMAINTDIR file is read. It is determined that the tape 333333 is the last tape in the set.
- Tape 222222 closes.
- Tape 333333 opens.
- Library maintenance moves to the end of the last file on tape 333333.
- Library maintenance appends three new files to the end of tape 333333.
- If tape 333333 fills before the copy operation completes, tape AAAAAA opens and the additional data is appended to tape AAAAAA.

LIBMAINTAPPEND = TOENDRESTORE

This case works similarly to the LIBMAINTAPPEND = TOEND case except when an error occurs while copying to the tape. If an error occurs that stops the copy to the tape, then the system restores the LIBMAINTDIR file to where it was at the start of the append request. This means none of the files were appended to the tape. However, you can retry the append request.

LIBMAINTDIR

Determines whether library maintenance should create a tape directory disk file on the DL LIBMAINTDIR disk family. The DL LIBMAINTDIR disk family describes the destination tape and the files copied to it. Library maintenance gives these files names of the form, LIBMAINTDIR/<tape name>/<date>/<tape serialno>. It also puts them under the usercode that library maintenance is running under, or * if there is no usercode.

Library maintenance stores the following information in these files:

- Serial numbers of the tapes used
- Names of the files copied to those tapes
- Certain other attributes of those files

You get a report of the information in tape directory disk files with the SYSTEM/FILEDATA utility program.

If the HOSTNAME volume attribute is also specified for the destination volume of the COPY or ADD statement, the LIBMAINTDIR file is created on the host with the destination tape.

LOCKEDFILE

When used in the COPY statement, this file attribute can be set for files copied to tape only. It has no effect on disk files. If this file attribute is set to TRUE for a file, the entire content of the specified destination library tape has locked file protection, and cannot be purged without operator confirmation.

MULTIVOLUME

Causes a multivolume set of CD-ROMs to be written when the data to be copied overflows a single CD-ROM. (If MULTIVOLUME is FALSE when a data overflow occurs, the copy is terminated with an error.)

When a multivolume set is written, data for a given CD-ROM is written to a temporary disk file, the CD-ROM is burned, and the temporary disk file is removed, all before the data for the next CD-ROM is written to a temporary disk file. In this way, the system ascertains whether storage capacity equal to or less than CD-ROM is needed.

In a multivolume copy, the system splits files across CD-ROM boundaries. This capability makes it possible to copy to a set of CD-ROMs files that are much larger than the capacity of a single CD-ROM.

Each CD-ROM in a multivolume set has a complete library maintenance directory listing all files on the entire set. As a result, you can use the TDIR system command on any of the CD-ROMs to list all files on the set.

The directory of a given CD-ROM (volume N) of a multivolume set contains the correct locations for the file data of all files located on previous CD-ROMs in the set (volumes 1 thru N). However, the directory records the location of all subsequent files as volume N+1. For example, if volume 1 of a multivolume set is mounted and you enter a COPY command specifying a file that happens to be on volume 3, the system asks for volume 2. If volume 2 is then mounted, the system asks for volume 3. If volume 3 is mounted, the COPY is then carried out.

PACKETWRITE

Packet write recording is used to back up and restore files and is selected by specifying the PACKETWRITE attribute. When set to TRUE, packet write recording is used when the CD-ROM is written. PACKETWRITE causes the data to be written to the CD-ROM in relatively small packets.

The MCP enables the buffer underrun prevention feature on CD-RW drives that support it. With this feature enabled, these drives resume writing after a buffer empty condition. Thus, buffer underrun does not occur in track-at-once mode, and there is no need to use the packet mode.

Packet write recording has the following advantages and disadvantages compared with track-at-once:

Advantages	Disadvantages
The risk of buffer underrun is eliminated.	The resulting disc can be read only in CD-R drives.
Less save memory is used for output buffers—130,390 words of output buffers are used for packet write versus 651,950 words for track-at-once.	The resulting disc cannot be used as a master to be sent to a CD-ROM factory for replication. Trackat-once output must be used for this purpose.
The burn runs at normal priority.	The capacity of the disc is reduced by 3.6% and throughput is reduced when writing to the disk.

When PACKETWRITE is FALSE (the default), track-at-once recording is used. Trackatonce causes the data to be written to the CD-ROM an entire track at a time. Track-at-once is the default recording mode and is normally used to create CD-ROMs to be read in CD-ROM drives or to create master CD-ROMs for replication.

However, this mode carries a potential risk of “buffer underrun.” Buffer underrun occurs when CD-ROM drive buffers become empty in the middle of a track, subsequently ruining the disc. To reduce this risk, track-at-once recording uses ten output buffers of 391,168 bytes and permits the implementation to run at MCP priority while writing to the CD-ROM drive.

The following events cause buffer underrun:

- Memory dump
- Path failure or path reconfiguration anywhere on the paths to the CD-ROM drive or the CD-ROM image disk file
- Heavy contention for the disk drives containing the CD-ROM image disk file
- Very long stack searches

Due to the limitations of packet write, it is currently used only to backup and restore files.

SAVEFACTOR

Indicates the expiration date of a tape volume in terms of the number of days past the creation date. The default value when a COPY statement creates a tape is 30 days. (When a library maintenance tape expires, the system does not automatically mark the tape as a scratch tape. The system uses the SAVEFACTOR value of the library maintenance tapes for reporting purposes such as in the reports generated by the LISTVOLUME utility program.)

SECURITYGUARD

Identifies the guard file to be invoked for the tape volume if the SECURITYTYPE attribute is assigned GUARDED or CONTROLLED. The default value is a null string (""). For more information about guard files, refer to the Security Features Guide.

SECURITYTYPE

Specifies how a usercode, other than that of the owner of a file can access a tape volume. The SECURITYTYPE attribute can have a value of PRIVATE (default), PUBLIC, GUARDED, or CONTROLLED. PRIVATE tape volumes can be accessed or overwritten only by their owners and privileged users. PUBLIC tape volumes can be accessed by tasks with any usercode, as limited by the setting of the SECURITYUSE attribute. The security of GUARDED and CONTROLLED tape volumes is determined by the guard file referenced by the SECURITYGUARD attribute.

SECURITYUSE

Specifies how a tape volume that is to be protected by the SECURITYTYPE attribute can be accessed by nonprivileged users using nonprivileged programs. This attribute can have a value of IO (default), IN, or OUT.

When a PUBLIC tape volume is accessed by a task with a usercode that differs from the FAMILYOWNER attribute value, the SECURITYUSE attribute can be used for the following actions based on its value:

- A value of IO permits reading, writing, overwriting, and purging.
- A value of IN permits reading but not writing, overwriting, or purging.
- A value of OUT permits writing, overwriting, or purging, but not reading.

SENSITIVEDATA

When the file is removed, causes the disk or pack areas assigned for a file to be overwritten with an arbitrary pattern before the disk space is returned to the system for reallocation.

SINGLEUNIT

Indicates whether areas for a disk file are to be allocated from a single family member. The default value is FALSE.

USECATALOG

If true, indicates that a tape listed in the volume library should be used. Library maintenance uses this attribute at cataloging installations only.

VERSION

Designates the successive iteration of the same generation of a tape volume. This file attribute is used in conjunction with the CYCLE attribute. The default value is 0.

Copying Files from Tape or CD-ROM

No single file on a tape or CD-ROM can be copied more than once from the same tape or CD-ROM in the same <copy from group>.

Examples

The following COPY statement causes the library maintenance to issue an "MT<unit number>X NOT ON T" error message for the second request unless there are two files named X on T:

```
COPY X, X AS Y FROM T
```

However, the following COPY statement copies X twice:

```
COPY X FROM T, X AS Y FROM T
```

For a COPY statement that includes a directory name and a file name that belongs under that directory name, library maintenance issues a "<unit> <file name> NOT ON <volume name>" error message, depending on whether the matching directory name or file name appears first in the COPY statement.

In the following example, the COPY statement causes library maintenance to issue an "MT<unit number> A/B NOT ON T" error message:

```
COPY A/=, A/B FROM T
```

However, the following statement ,might cause library maintenance to issue an "MT<unit number> A NOT ON T" error message, depending on whether there are files other than A/B under the A directory on tape T:

```
COPY A/B, A/= FROM T
```

When copying directories from tape or CD-ROM, if you specify two directories with the same name in the same FROM clause, library maintenance issues a "<unit> <directory name> NOT ON < volume name>" error message for the specified second directory.

For example, the following statement copies once only the files under X:

```
COPY X/=, X/= FROM T
```

However, the following statement copies the files under X twice:

```
COPY X/= FROM T, X/= FROM T
```

When copying directories from tape or CD-ROM, if you specify two directories in the same FROM clause and one directory subsumes the other, then library maintenance issues a "<unit> <directory name> NOT ON < volume name>" error message, depending on which directory appears first in the list.

For example, the following statement causes library maintenance to issue the error message "CD <unit number> X/Y NOT ON C":

```
COPY X/=, X/Y/= FROM C(CD)
```

However, the following statement might cause library maintenance to issue an "CD <unit number> X NOT ON C" error message, depending on whether there are any files on the tape under the X directory that are not under the X/Y directory:

```
COPY X/Y/=, X/= FROM C(CD)
```

Copying Files with or without Usercodes

Library maintenance selects usercoded and unusercode files differently when you copy files from tape, depending on whether the task is running with a usercode.

Running without a Usercode

The following list shows how library maintenance determines how to select files when the task is unusercoded. Files without usercodes are indicated by an asterisk (*).

- COPY = FROM TAPE
COPY = FROM C(CD)

Conditions: Task running without a usercode.

Result: Selects all files on source volume.

- COPY *= FROM TAPE
COPY *=FROM C(CD)

Conditions :Task running without a usercode.

Result: Selects all files on source volume.

- COPY D/=FROM TAPE
COPY D/=FROM C(CD)

Conditions: Task running without a usercode and copying files from a directory without specifying a usercode.

Result: Selects all files that match the specified directory.

- `COPY *D/=FROM TAPE`
`COPY *D/= FROM C(CD)`

Conditions: Task running without a usercode and copying files from a directory with an asterisk (*).

Result: Selects all files from the directory under the asterisk (*) directory.

Running under a Privileged Usercode

The following list shows how library maintenance determines how to select files when the task is running under a privileged usercode. Files without usercodes are indicated by an asterisk (*).

- `COPY * = FROM TAPE`
`COPY * = FROM C(CD)`

Conditions: Privileged task running with a usercode and copying files from an asterisk (*) directory.

Result: Selects all files on source volume.

- `COPY = FROM TAPE`
`COPY = FROM C(CD)`

Conditions: Privileged task running with a specific usercode and copying files without indicating a usercode.

Result: Selects files either from the asterisk (*) directory or from the task usercode, but not from both. The first file the system finds on the source volume that matches either the asterisk (*) directory or the usercode determines the location from which files are copied.

Example: If the first matching file is an asterisk (*) directory file, only matching asterisk (*) directory files are copied. If the first matching file is a file under the task usercode, the system copies only files that match that usercode.

- `COPY D/= FROM TAPE`
`COPY D/= FROM C(CD)`

Conditions: Privileged task running with a usercode and copying files from a directory without an asterisk (*) directory or specific usercode.

Result: Selects files either from the asterisk (*) directory or from the task usercode, but not from both. The first file the system finds on the source volume that matches either the asterisk (*) directory or the usercode determines the location from which files are copied.

Example: If the first matching file is an asterisk (*) directory file, the system copies only matching asterisk (*) directory files. If the first matching file is a file under the task usercode, the system copies only files that match that usercode.

Running Under a Nonprivileged Usercode

The following list shows how library maintenance determines how to select files when the task is running under a nonprivileged usercode. Files without usercodes are indicated by an asterisk (*).

- `COPY *=FROM TAPE`
`COPY *=FROM C(CD)`

Conditions: Nonprivileged task running with a specific usercode and copying files from an asterisk (*) directory.

Result: Does not select any files for copying.

- `COPY D/= FROM TAPE`
`COPY D/= FROM C(CD)`

Conditions: Nonprivileged task running with a specific usercode and copying files from a directory that does not indicate an asterisk (*) directory or specific usercode.

Result: Selects only those files under the task usercode that match the directory specification.

- `COPY = AS ... FROM TAPE`
`COPY = AS ... FROM C(CD)`
`COPY = ONTO...FROM TAPE`
`COPY = ONTO...FROM C(CD)`
`COPY D/= AS...FROM TAPE`
`COPY D/= AS...FROM C(CD)`
`COPY D/= ONTO...FROM TAPE`
`COPY D/= ONTO...FROM C(CD)`

Conditions: Using the AS or ONTO option with a nonprivileged task running with a specific usercode and copying files from a directory that does not indicate an asterisk (*) or specific usercode.

Result: Selects files either from the asterisk (*) directory or from the task usercode, but not from both. The first file the system finds that matches either the asterisk (*) directory or the usercode determines which files are copied.

Example: If the first matching file is an asterisk (*) directory file, the system copies only matching asterisk (*) directory files. If the first matching file is a file under the task usercode, the system copies only files that match that usercode.

Copying Files to CD-ROMs

`COPY` to `KIND=CD` writes an image of a library maintenance format CD-ROM to a temporary disk file. The image is then transferred to a blank CD-R disc that is mounted on a CD-R drive connected directly to a SCSI channel adapter.

Note: *CD-R drives function as CD-ROM drives when reading CD-ROMs; however, CD-R drives can also write to blank write-once CD-R media, creating a CD-ROM.*

When you write to a library maintenance CD-ROM, you can specify the `SERIALNO` attribute for the destination volume. The specified serial number is assigned to the CDROM as it is written. However, you can specify only a single serial number. If you create a multiple-volume set of CD-ROMs, all are assigned the same serial number.

When you read a library maintenance CD-ROM, if you do not specify the SERIALNO attribute for the source volume, a source CD-ROM is used whether has a serial number or not. However, if you do specify the SERIALNO attribute for the source volume, a source CD-ROM is used only if it has a matching serial number.

Track-at-once is the default recording mode. If you specify the attribute PACKETWRITE, "packet write" recording is used when the CD-ROM is written. Additionally, you can specify how many copies of the disc are to be burned with the attribute CDCOPIES. Refer to the explanations for each of these settings in the following paragraphs.

Before the CD-ROM disc is finalized with the CLOSE SESSION SCSI command, the entire data track is read and compared with the image file to ensure that no media defects affected the burn.

Note: *Except during this compare operation, the MCP refuses to read a CD-ROM disc that is not finalized. Discs that fail the compare operation cannot be read by mistake thereafter.*

The family used for the temporary disk file must have room for the entire image. Also, for track-at-once recording, you cannot use a family that is heavily used by other programs. The temporary disk file is created on the default family of the task executing the COPY statement, unless you set the TASKSTRING attribute of that task to

FAMILYNAME = <family name>

Restrictions

- The capabilities of KIND=CD apply only to the COPY statement and are not valid with the ADD statement.
- The only "& options" are COMPARE, DSONERROR, WAITONERROR, and REPORT.
- There can be only one destination volume and the destination volume cannot require quotes.
- The source and destination volumes must both be on the local host. NFT is not supported.
- Except when using the MULTIVOLUME file attribute, the data must fit on a single CD-ROM.
- For track-at-once output, a single CD-ROM holds 681984000 bytes or 333000 CD-ROM sectors of 2048 bytes each.
- For packet write output, a single CD-ROM holds 657553408 bytes or 321071 CDROM sectors of 2048 bytes each.
- All errors except skipped files are fatal and cause the task executing the COPY statement to terminate with a failure.
- The HOSTNAME attribute cannot be specified when KIND = CD.

Example 1

```
COPY TEST1, TEST2 TO GGG(CD);
```

A library maintenance CD-ROM named GGG is created. The disk contains files TEST1 and TEST2.

Example 2

```
BEGIN JOB J;  
  TASK T;  
    T(TASKSTRING= "FAMILYNAME = BLUE");  
    COPY *SYSTEM/PRINT/= TO SYSTEM_PRINT(CD) [T];  
END JOB
```

A library maintenance CD-ROM named SYSTEM_PRINT is created. The disc contains all the files from *SYSTEM/PRINT/=.

The temporary disk file used to hold the CD-ROM image is placed on family BLUE.

Example 3

```
COPY (USER9)= FROM GREEN(PACK),  
      =FROM TEST4 (SERIALNO= "123321"),  
      *SYSTEM/PRINT/= FROM SYSTEM_45123(CD) TO  
      SYSTEM_012(CD);
```

A library maintenance CD-ROM named SYSTEM_012 is created. The disc contains all the files from (USER9)= on family GREEN, from tape TEST4 (with serial number 123321), and from *SYSTEM/PRINT/= on CD SYSTEM_45123.

COPY = AS and COPY *= AS

The COPY = AS . . . and COPY *= AS . . . statements initiate one of the following actions:

- Select all files to be copied from the source volume.
- Select all files under the usercode of the task, and copy the files from the input volume.
- Select all files under the global *directory, but will not select any files under usercodes to be copied from the input volume.
- Select no files for copying.

When either statement is used to select input files with usercodes in their file names, the output file name is formatted in one of the following ways:

- The output file names contain the same usercode as the input file names.
- The output file name does not contain the usercode. The output files are copied to the global *directory.
- The usercode of the output file name is changed to an ordinary node name.

The outcome of specifying a simple directory such as = or *= for the input files, followed by AS and an output directory specification, varies depending on whether

- You specify COPY = AS . . . or COPY *= AS
- The task is run with or without a usercode, and the task is a privileged or a nonprivileged task.

- The source is a disk or a tape.

The following table is a description of using the COPY statements COPY = AS . . . and COPY *= AS . . . to copy files running without a usercode, with a usercode, or with a privileged usercode.

COPY Statement	Conditions	Result
COPY = AS... COPY *= AS...	Task running without a usercode with a disk or tape input volume	Selects all files on the input disk or tape volume and changes the usercode node of any input file to an ordinary node in the output file name.
COPY = AS...	Task running under a privileged or a nonprivileged usercode with a disk input	Selects only the files under the usercode of the task.
COPY = AS...	Task running under a privileged or a nonprivileged usercode with a tape input volume	Selects files under the usercode of the task or files without usercodes . The file order on the tape determines which files are copied. If a file with the usercode of the task is found on the tape before a file without a usercode, only files under the usercode of the task are copied. If a file without a usercode is found on the tape before a file with the usercode, of the task, only files without usercodes are copied. All output file names correspond to the name specified after the word AS in the COPY statement.
COPY *= AS...	Task running under a privileged usercode	Selects all files on the input volume for copying and changes the usercode node of any input file name to an ordinary node in the output file name. Note: A task running with a nonprivileged usercode is not permitted to use this construct and will receive a run time security error.

Examples

Tasks running without a usercode:

- This example copies the input file *X as *ABC/X and copies the input file (UC)Y as *ABC/UC/Y:

```
COPY = AS ABC/=
```

- These examples copy the input file *X as *X and copies the input file (UC) Y as *UC/Y:

```
COPY = AS =  
COPY *= AS =  
COPY *= AS *=
```

Tasks running under a privileged or a nonprivileged usercode with a disk input volume:

- This example copies all the files under the usercode of the task UC without changing their names. This statement is equivalent to the following statement:

```
USER=UC  
COPY = AS = FROM DISK...
```

- This example copies the input file (UC)X as (UC)X. Files such as *Y or (XUSER)FILE are not copied.

```
COPY = FROM DISK...
```

- This example copies the input file (UC)X as *X.

```
USER=UC  
COPY = AS *= FROM DISK...
```

(Files such as *Y or (XUSER)FILE are not copied.)

Tasks running under a privileged or a nonprivileged usercode with a tape input volume:

- This example copies the file (UC)X as (UC)P/X.

```
USER=UC  
COPY = AS P/= FROM TAPE...
```

- This example copies the file (UC)X as *Q/X.

```
USER=UC  
COPY = AS *Q/= FROM TAPE...
```

Note: If this statement is used to copy to a disk, a nonprivileged user gets a security error and no files are copied.

Tasks running under a privileged usercode:

- This example copies the input file *X as *ABC/X and copies (UC)Y as *ABC/UC/Y.

```
USER=PRIV  
COPY *= AS *ABC/=
```

- This example copies the input file *X as *X and copies (UC)Y as *UC/Y.

```
USER=PRIV  
COPY *= AS *=
```

- This example copies the input file *X as (PRIV)X and copies (UC)Y as (PRIV)UC/Y.

```
USER=PRIV  
COPY *= AS =
```

COPY and ADD Statement Examples

COPY Statement Examples

The following examples illustrate various aspects of using COPY to copy files locally using library maintenance.

For information regarding file transfer services for copying to remote hosts, refer to the end of this section that discusses the particular type of file transfer service you are interested in.

- This statement copies the files DATA/HOLD and PROG/SUMMARY from the family DISK to tape. The tape will be named XFER.

```
COPY DATA/HOLD, PROG/SUMMARY TO XFER;
```

- This statement copies the file PROG/SUMMARY from the tape XFER to the family PACK. The COMPARE option double-checks the copy.

```
COPY & COMPARE PROG/SUMMARY FROM XFER TO PACK;
```

- These statements copy the files (UC)OBJECT/C/PROG and (UC)C/PROG from the disk family USERPACK to the disk family SYSPACK.

```
USER = UC;
FAMILY DISK = USERPACK OTHERWISE DISK;
COPY OBJECT/C/PROG, C/PROG TO SYSPACK (PACK);
```

- These statements simultaneously copy all files under the usercode UC on the disk family USERPACK to one backup tape (or set of tapes if the files fill up more than one tape volume), which will be named UCUSERPACK, and all files under the usercode UC from the disk family PACK to another backup tape (or set of tapes), which will be named UCPACK. The VERIFY option double-checks the copies.

```
USER = UC;
FAMILY DISK = USERPACK OTHERWISE DISK;
PROCESS COPY & VERIFY = TO UCUSERPACK;
COPY & VERIFY = FROM PACK TO UCPACK;
```

- If these statements run under a privileged usercode or without a usercode in a job started from an ODT, they copy all the files from the family USERPACK to a backup tape (or set of backup tapes if all the files do not fit on one tape volume). The tape or tapes will be named USERPACK030891. The VERIFY option double-checks the copies. A task variable, T, is used to check the success of the copy.

```
TASK T;
RECOPY:
COPY & VERIFY *= FROM USERPACK (PACK) TO USERPACK030891 [T];
IF T(VALUE) NEQ 0 THEN
BEGIN
  DISPLAY "RETRYING BACKUP OF USERPACK.";
  GO RECOPY;
END;
```

- This statement copies and compares all files under the usercode UC from the tape USERPACK030891 to another backup tape. The new tape will be named UCSAVE. Immediately after each file is copied to tape UCSAVE, it is compared with the original file on tape USERPACK030891.

```
COPY & COMPARE (UC)= FROM USERSPACK030891 TO UCSAVE;
```

- This statement, if run under a privileged usercode or run without a usercode in a job started from an ODT, makes two backup copies of every file on the family DISK onto two sets of backup tapes. One tape (or set of tapes if all the files will not fit on one tape volume) will be named ADISK, and the other will be named BDISK.

```
COPY & COMPARE *= FROM DISK TO ADISK, TO BDISK;
```

- These statements copy the file A/B, name the copy X/Y, and copy all of the files under the directory Z/= from DISK to tape T1:

```
S1:="A/B";  
S2:="X/Y";  
S3:="Z/=";  
S4:="T1";  
COPY #S1 AS #S2, #S3 FROM DISK TO #S4(KIND=TAPE);
```

- These statements copy all files under the (UC)OBJECT/= directory on the pack USERPACK, and put the new copies under the (UC)SAVE/OBJECT= directory on USERPACK.

```
USER = UC;  
COPY OBJECT/= AS SAVE/OBJECT= FROM USERPACK (PACK)  
TO USERPACK (PACK);
```

- These statements copy the file *SYSTEM/FILEDATA on the pack PACK and the new copy of the file is named (UC)SYSTEM/FILEDATA on the pack USERPACK.

```
USER = UC;  
COPY *SYSTEM/FILEDATA AS SYSTEM/FILEDATA FROM PACK  
TO USERPACK (PACK);
```

- This statement copies all files that are under the SYSTEM/= directories on the disk families DISK, PACK, and SYSPACK to a single tape set that will be named SYS. The SERIALNO attribute specifies which tape or tapes will be used. Library maintenance will first request the tape with the serial number X66778. If that tape fills up, library maintenance will make an additional request for the tape 553322. If that tape fills up, library maintenance will request any available scratch tape.

```
COPY SYSTEM/= FROM DISK, SYSTEM/= FROM PACK,  
SYSTEM/= FROM SYSPACK (PACK)  
TO SYS (SERIALNO = ("X66778", 553322));
```

- This statement copies the file SYSTEM/MCP from the disk family HLPACK to the disk family named PACK. The FAMILYINDEX=0 specification erases the value of the FAMILYINDEX attribute in the new copy of the file.

```
COPY SYSTEM/MCP FROM HLPACK (PACK) TO PACK (FAMILYINDEX=0);
```

- This statement copies all the files in the directory SYMBOL/= from the tape REL to the disk family called DISK and all the files in the directory SYSTEM/= also from the tape REL, to the disk family PACK. Each new copy will be marked as a cataloged file, and a backup entry will be added to the catalog for each file. The backup entry will point to the source tape REL. (This example assumes that a VOLUME ADD has been done for the tape REL, and for the disk families DISK and PACK.)

```
COPY & COMPARE & CATALOG SYMBOL/= FROM REL (TAPE) TO DISK,  
SYSTEM/= FROM REL (TAPE) TO PACK;
```

- If this statement runs under a privileged usercode or without a usercode in a job

started from an ODT, it copies all files from the disk family USERPACK to a backup tape that will be named USERPACK. For each cataloged file that is copied, library maintenance adds a backup entry to the catalog record that points to the tape USERPACK. (This example assumes that a VOLUME ADD has been done for the tape USERPACK and the disk family USERPACK.)

```
COPY & VERIFY & BACKUP *= FROM USERPACK (PACK) TO USERPACK;
```

- At an installation that uses the TAPECHECK security feature, these statements copy all the files in the (UC)OBJECT/= directory from the pack USERPACK to a tape that will be named UCOBJECT. This tape will have PUBLIC IN security, so that nonprivileged users can copy files from it.

```
USER = UC;
COPY OBJECT/= FROM USERPACK (PACK)
TO UCOBJECT (SECURITYTYPE=PUBLIC, SECURITYUSE=IN);
```

- This statement, if run under a privileged usercode or run without a usercode in a job started from an ODT, copies all files from the disk family DISK to a tape that will be called DISKBACKUP and all files from the disk family PACK to a tape that will be called PACKBACKUP. This statement is equivalent to two separate COPY statements. By using this one statement, if the "first" copy is terminated abnormally or discontinued (such as with the DS system command), the "second" copy will not proceed. If you used two separate statements, and if the "first" copy was terminated abnormally or discontinued, the "second" copy will proceed.

```
COPY & COMPARE *= FROM DISK TO DISKBACKUP,
                *= FROM PACK TO PACKBACKUP;
```

ADD Statement Examples

- The following statement copies the file X/Y from tape T to DISK, only if no file named X/Y already resides on DISK.

```
ADD X/Y FROM T(KIND=TAPE);
```

- The following statement copies files under the directory Z/= from tape T to disk R and to DISK. All files already resident on the destination volumes are not copied. Different files might be copied to R and to DISK, depending on what is already resident on each destination volume before the ADD statement is executed.

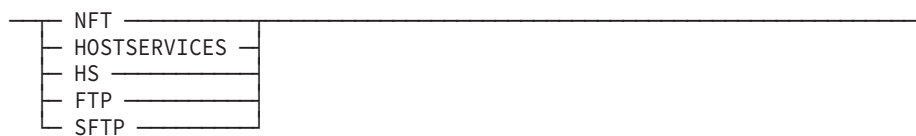
```
ADD Z/= FROM T(KIND=TAPE) TO R(KIND=DISK), TO DISK;
```

- The following statements restore all the "missing" files for the usercode UC from the tape UCUSERPACK to the disk family USERPACK. The VERIFY option double checks the copy.

```
USER = UC;
ADD & VERIFY = FROM UCUSERPACK TO USERPACK (PACK);
```

COPY File Transfer Services

<transfer service>



Explanation

The process of copying files between hosts is called a file transfer, and the software used to copy the files between hosts is called a transfer service. When you initiate a file transfer, distributed systems services determines which transfer service can be used for the hosts involved in the transfer. Since distributed systems services selects the most suitable transfer service, you do not have to specify the transfer service in the COPY statement. You should specify the transfer service only when you want to override the automatically selected transfer service.

Note: File transfer services and the transfer service construct do not apply when you transfer files on a single host. In this case, library maintenance is used to transfer the files.

Available File Transfer Services

The file transfer services that distributed systems services selects or that you specify in the COPY statement include the following:

- Native File Transfer (NFT)

NFT enables you to transfer files between MCP BNA hosts. You can use NFT to copy files from disk families to disk families and tape volumes, from library maintenance tape volumes to a disk family or a tape volume, and from CD-ROM volumes to disk families and tape volumes.
- Host Services File Transfer

Host Services File Transfer enables you to copy disk files between MCP environment systems, B 1000, V Series, and CP9500 hosts in a BNA network and between MCP environment systems in an Open Systems Interconnection (OSI) network.
- File Transfer Protocol (FTP)

FTP enables you to copy disk files between hosts connected to a Transmission Control Protocol/Internet Protocol (TCP/IP) network.
- SSH File Transfer Protocol (SFTP)

SFTP enables you to copy disk files between hosts connected to a Transmission Control Protocol/Internet Protocol (TCP/IP) network over the SSH secure shell protocol.

Principles of Selection

The system selects the file transfer service based on the following conditions:

- Whether you specify a particular file transfer service in the COPY statement
- Whether you specify options or attributes that require a particular file transfer service
- Whether the involved source and destination hosts support the transfer service
- Which network or networks (BNA, OSI, or TCP/IP) connect the hosts involved in the transfer

Order of Selection

The selection proceeds as follows:

- If you specify a particular file transfer service in the COPY statement, the system tries to use it.
- If you specify DOMAINNAME or an IPADDRESS, the system selects FTP.
- If any of the following conditions is met, the system selects NFT:
 - The file names and directory names in the COPY, ADD, or REPLACE statement exceed 60 to 80 kilobytes of text.
 - You use the REPLACE statement instead of COPY or ADD.
 - You specify the SKIPEXCLUSIVE option.
 - You specify a file copy from tape by file number with an “# <file number>” clause option.
 - You specify a CD-ROM source volume.
- The system selects the transfer service based on whether the involved hosts can support the transfer service.
- NFT is selected when both local and remote hosts support NFT.

The system tries NFT first, whenever possible, because NFT provides all of the features that are provided by Host Services File Transfer plus additional features. These additional features include the resumption of file transfer from the point of failure, the simultaneous transfer of files to several destinations, and the transfer of directory copies to and from remote hosts. All features of NFT are explained in the *Distributed Systems Services Operations Guide*.

- If both source and destination hosts are connected by an OSI network, distributed systems services selects Host Services File Transfer.
- If both source and destination hosts are connected by a TCP/IP network, the system tries FTP.

Additional Considerations

If the COPY statement contains a copy request with multiple hosts, the appropriate service for each pair of source and destination hosts is selected. Therefore, it is possible for more than one transfer service to be used to transfer the files specified in a COPY statement. If a host is offline or does not exist, distributed systems services displays a message and disregards the copy request to that host. However, copy requests between other hosts, if valid, are still performed.

If a file is transferred between two remote hosts through the initiating host (rather than directly between the two remote hosts), all three hosts must use the same transfer service.

Caution

For file transfers involving files with node names exceeding 17 characters, the SYSOPS LONGFILENAMES option for both sending and destination hosts must be set. (For details on the SYSOPS command, see the *System Commands Reference*.)

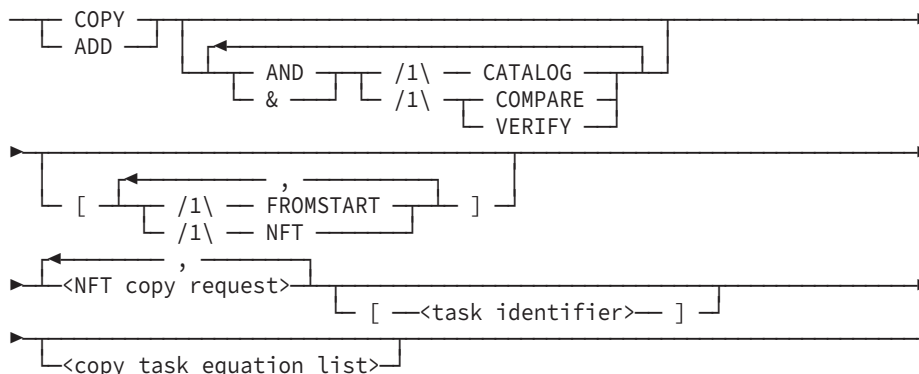
More About the File Transfer Services

Each of the file transfer services has certain features that you should know about and certain restrictions that you must observe. These features and restrictions are explained under "NFT File Transfers," "Host Services File Transfers," and "FTP File Transfers" found later in this section.

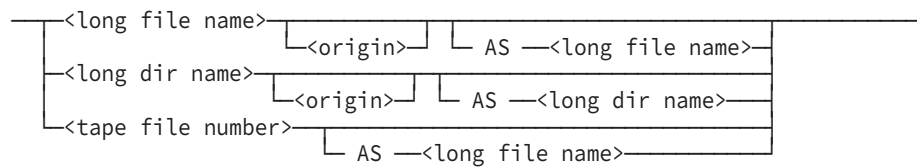
For more extensive information about transferring files between hosts on BNA or OSI networks, or about using NFT or Host Services File Transfer, refer to the *Distributed Systems Services Operations Guide*. For information about using FTP to transfer files between hosts on TCP/IP networks, refer to the *TCP/IP Distributed Systems Services Operations Guide*.

NFT File Transfers

<NFT copy or add statement>



<NFT file>



The following section describes NFT file transfers to disk.

Transferring Files to Disk

When you transfer a file to disk by using NFT, the FILENAME and FILEKIND attributes of the destination file are set aside temporarily while the file is being transferred. The FILENAME attribute is set to NFTTEMP/<file name>, and the FILEKIND attribute is set to FTAUDIT. These attributes are set to their original values following the successful transfer of the entire file.

When a file transfer fails and you reissue the original COPY statement, NFT restarts the interrupted file transfer. The resumption point depends on the kind of destination volume and the characteristics of the files being transferred. For more information on file transfer resumption, refer to the *Distributed Systems Services Operations Guide*.

Constraints

The following list describes the constraints for NFT file transfers:

- If you specify the CATALOG option and the destination host is a cataloging system, the destination host marks the copied files as cataloged. However, the destination host does not place a backup entry that references the source file in its system catalog.
- Do not use tape or CD-ROM volumes as sources, unless you specify only one destination volume.
- Do not specify the USERCODE or GENERATION attributes in the source volume attribute list construct or the destination volume attribute list construct.
- You cannot specify KIND=CD for a destination volume when using NFT.

Examples

The following examples illustrate the COPY statement syntax when NFT is used to transfer files between host systems.

This statement copies the file A from the family DISK on the local host to a pack named PACK on host HOSTB. For this example, assume that the COPY statement is part of a WFL job that is restarted if the file transfer fails. If the WFL job restarts, file A is completely re-transferred because the FROMSTART option is specified in the COPY statement.

```
COPY [NFT, FROMSTART] A TO PACK (HOSTNAME=HOSTB) ;
```

This statement copies files from the SYMBOL/= directory on the source CD-ROM volume SYMCD at host CENTER. It copies only those files under SYMBOL/= for which there is already a resident version on the SYM disk family:

```

REPLACE [NFT] SYMBOL/=
  FROM SYMCD (CD, HOSTNAME=CENTER)
  TO SYM (PACK);

```

This WFL job copies the file A from the family DISK on the local host to a pack named PACK on the host HOSTB, taking advantage of the file transfer resumption feature of NFT:

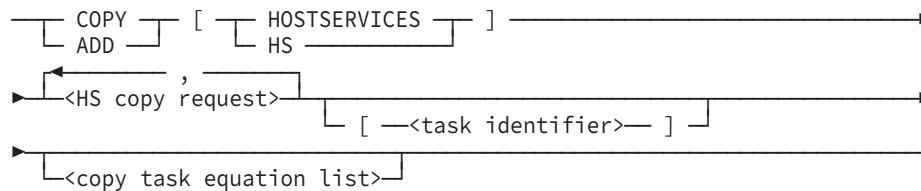
```

BEGIN JOB NFT/RECOVERY;
  TASK T;
  RESUME:
  COPY A TO PACK(HOSTNAME=HOSTB)[T];
  IF T(VALUE) NEQ 0 THEN
    BEGIN
      WAIT(30);
      GO RESUME;
    END;
  END JOB.

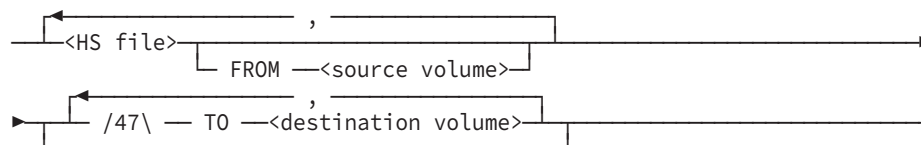
```

Host Services File Transfer

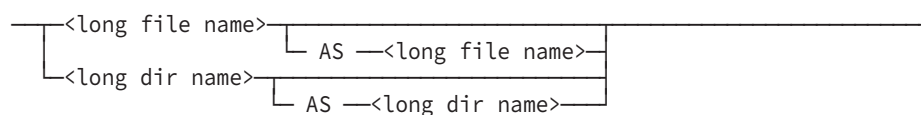
<HS copy or add statement>



<HS copy request>



<HS file>



When you transfer a file by using Host Services File Transfer, you must observe the following restrictions:

- Do not specify a directory in the copy file construct unless it resides on the local initiating host.
- Copy only from disk to disk.
- Specify only the attributes KIND and HOSTNAME in the source volume attribute list and destination volume attribute list constructs.

Examples

The following examples illustrate the COPY statement syntax when Host Services File Transfer is used to transfer files.

This statement copies the file X from DISK (at the local host) to DISK at host HOSTB, and names the new file Y. The HOSTSERVICES option is used to designate that Host Services File Transfer should be used to transfer the file.

```
COPY [HOSTSERVICES] X AS Y FROM DISK TO DISK(HOSTNAME=HOSTB) ;
```

This statement copies the file WEEKLY_SUMMARY (from DISK at the local host) to DISK at host HOSTB, and renames the file as WEEKLY/SUMMARY. In this example, HOSTB does not support NFT, so Host Services File Transfer is selected automatically.

```
COPY WEEKLY_SUMMARY AS WEEKLY/SUMMARY TO DISK(HOSTNAME=HOSTB) ;
```

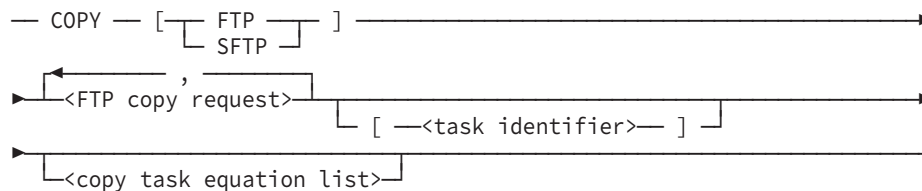
This statement copies the file TIME/SUM from the disk ENGDATA at the host HOSTE as the file ENG/TIME/SUM on the disk ACCTDATA at the host HOSTA. In this example, HOSTE supports NFT but HOSTA does not. Therefore, Host Services File Transfer is automatically selected to copy the files.

```
COPY TIME/SUM AS ENG/TIME/SUM FROM ENGDATA
(KIND=DISK,HOSTNAME=HOSTE)
TO ACCTDATA(KIND=DISK,HOSTNAME=HOSTA) ;
```

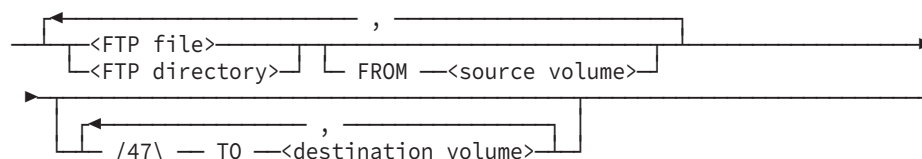
FTP File Transfers

SFTP File Transfers

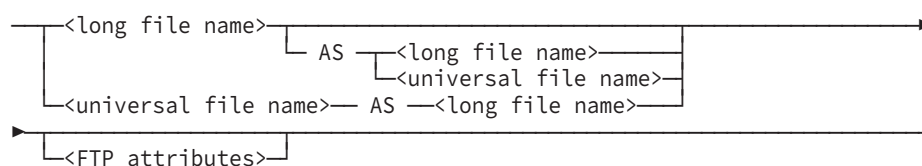
<FTP copy>



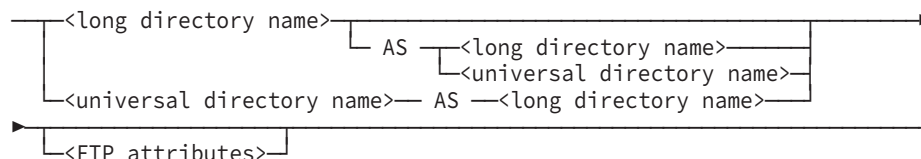
<FTP copy request>



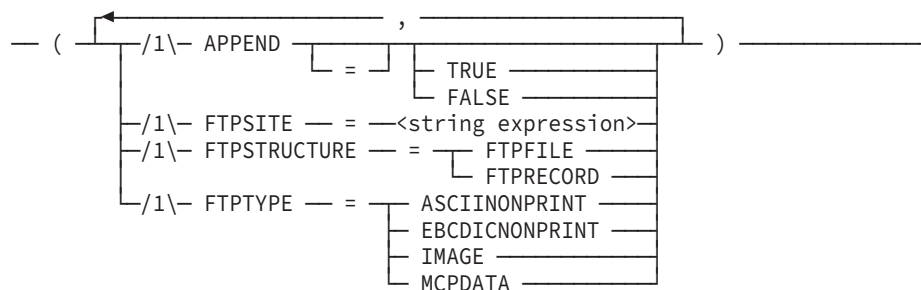
<FTP file>



<FTP directory>



<FTP attributes>



Explanation

When you transfer a file by using FTP, you should note that FTP does not distinguish between the ADD and COPY statements. A file existing under the name specified in the ADD statement is overwritten on a receiving host.

Any ASCII nonprint and EBCDIC nonprint files copied by using FTP reside on disk as FTPDATA files that are specially formatted. An FTPDATA file is not a standard MCP environment file kind and must be converted before it can be processed, printed, or viewed by MCP environment software. The FTP utility program can convert an FTPDATA file to a conventional MCP environment format with a title and certain other attribute values that you designate. For more information on converting FTPDATA files with the FTP utility, refer to the *TCP/IP Distributed Systems Services Operations Guide*.

When you transfer a file by using FTP, you must observe the following restrictions:

- Copy only from disk to disk.
- Specify only the IPADDRESS, DOMAINNAME, KIND, HOSTNAME, and USERCODE attributes in the source volume attribute list and destination volume attribute list constructs.
- Specify only the volume name DISK in the source volume and destination volume constructs.
- Depending on the FTP implementation of a receiving non-MCP host, FTPSTRUCTURE=FTPFILE and FTPTYPE=ASCIINONPRINT might be the only file structure and character code type recognized.
- Other options for the FTPSTRUCTURE and FTPTYPE transform attributes might result in an error message from the non MCP host, indicating that the specified FTPSTRUCTURE or FTPTYPE is not recognized at the remote host.

FTP Transform Attributes for Copying Files

The following FTP transform attributes are used to assign file characteristics to files that are transferred to a remote host through the FTP file transfer service.

APPEND

Appends the contents of the source file to the destination file. The APPEND attribute applies only if the destination is not on an MCP host.

FTPSITE

Specifies any of several options that are described in detail in the *TCP/IP Distributed Systems Services Operations Guide*.

FTPSTRUCTURE

Identifies the structure of the file that is being transferred.

FTPFILE represents the file structure where there is no internal structure. The file is considered to be a continuous sequence of data bytes. This is the default structure used when transferring a file if FTPSTRUCTURE is not specified.

FTPRECORD represents a record structure where the file is made up of sequential records.

FTPTYPE

Identifies the type of code format (either ASCII, EBCDIC, or image) in which the characters of the file are represented.

ASCII NONPRINT represents the file in ASCII format without vertical format information. Vertical format control characters provide printer instructions such as carriage return (CR), line feed (LF), new line (NL), form feed (FF), and so on. The ASCII format is the default code format type.

EBCDIC NONPRINT represents the file in EBCDIC format without vertical format control information.

IMAGE represents the file in 8-bit transfer bytes. Data is sent as contiguous bits and must be stored as contiguous bits by the receiving host. If the receiving host must store the data in a manner such that the file (or record for a record-structured file) necessitates the padding of the file with 0 (zeroes) to some convenient boundary such as byte, word, or block, then the zeroes must be placed at the end of the file or record and the padding bits must be identifiable.

MCPDATA represents the file in EBCDIC format with added MCP attribute information. Embedded MCP attributes allow the file to be stored at the destination with the same attributes as the source file.

Examples

The following examples illustrate the COPY syntax when FTP is used to transfer files across a TCP/IP network.

This statement copies the file PAYROLL/TIMECARDS/061490 from DISK at the local host as the file [PAY735]TC061490.DAT on DISK at the remote host HQSYS1. Log-on information is passed to the remote host HQSYS1 with the USERCODE attribute.

```
COPY PAYROLL/TIMECARDS/061490
  AS '[PAY735]TC061490.DAT'
  TO DISK(KIND=PACK, HOSTNAME=HQSYS1,
    USERCODE=RMPAYR/MONEY/735);
```

This statement copies the file [PAY735]90AWARDS.DAT from DISK at the remote host HQSYS1 as the file PAYROLL/AWARDS/1990 on DISK at the local host.

```
COPY '[PAY735]90AWARDS.DAT'
  AS PAYROLL/AWARDS/1990
  FROM DISK(KIND=PACK, HOSTNAME=HQSYS1,
    USERCODE=RMPAYR/MONEY/735) TO DISK;
```

The following examples show some uses of the FTP transform attributes.

Transfer by IP Address, with FTPTYPE = IMAGE

The following example transfers a file FILE1 as file 1 from a local MCP host to a remote host 125.32.1.1. The usercode is INV, the password is 4367, and the account number is 88315.

```
COPY FILE AS 'file1' (FTPTYPE = IMAGE) TO DISK(IPADDRESS =
  "125.32.1.1", USERCODE = INV/4367/88315)
```

Transfer by Domain Name, with FTPSTRUCTURE = FTPRECORD

The following example transfers a file PARTS.LIST as PARTS/LIST from a remote host NIC.DDN.MIL to a local MCP host. The FTPSTRUCTURE is set to FTPRECORD.

```
COPY "PARTS.LIST" AS PARTS/LIST (FTPTYPE = EBCDICNONPRINT,
  FTPSTRUCTURE = FTPRECORD) FROM DISK(DOMAINNAME = "NIC.DDN.MIL",
  USERCODE = FDR/'/'/'') TO TCP(PACK)
```

Transfer Using Secure Socket Layer (SSL) Protocol

The following example transfers FILEX from the MCP host MP156 to the local MCP host using SSL data encryption.

```
COPY [FTP] FILEX (FTPTYPE=IMAGE)
  FROM DISK (SSLMODE=IMPLICIT, USERCODE=PTF/PTF, HOSTNAME=MP156);
```

Transfer Using Secure Shell (SSH) Protocol

The following example transfers FILEX from the MCP environment to host MVHOST using SSH data encryption.

```
COPY [FTP] FILEX (FTPTYPE=ASCIINONPRINT) FROM DISK
  TO DISK(USERCODE=ABC/ABC, HOSTNAME=MVHOST, AUTHMODE=SSH)
```

Transfer a Case-sensitive Directory

The following example transfers all files in the Test directory from a remote host to a local MCP host. The interchange name is used to preserve the case of the directory name sent to the remote host.

```
COPY 'Test' /= AS TEST /= FROM DISK
  (DOMAINNAME = "rhel5", USERCODE = UC/PW)
```

Restarting COPY Interrupted File Transfers

When a host or network failure interrupts a COPY statement, the WFL job containing the COPY statement automatically restarts if the failure was caused by a halt/load process or if the job was written to restart after an unsuccessful file transfer.

The results of restarting the COPY statement depend on the type of transfer, as follows:

- If the file transfer is local, all files named in the COPY statement are transferred again when the job restarts.
- If the file transfer is remote and the transfer service is Host Service File Transfer or FTP, then all files transferred by Host Services File Transfer or FTP are transferred again when the job restarts.

If some of the files were transferred with NFT, it might not be necessary for all files named in the COPY statement to be retransferred. Those files transferred with NFT may be transferred again according to the rules for the FROMSTART option, listed in the following item.

- If the file transfer is remote and the transfer service used is NFT, the FROMSTART option determines how the copying process or processes will be restarted.
- If the FROMSTART option is not specified, NFT automatically resumes the COPY statement so that data already transferred is not transferred again.

The resumption point depends on the kind of destination volume and the characteristics of the files being transferred. For more information about resuming a file transfer in NFT, refer to the *Distributed Systems Services Operations Guide*.

- If the FROMSTART option is specified, all files named in the COPY statement are completely transferred again, whether or not the file transfer in NFT can be resumed.

Note: When the FROMSTART option is specified, all previously created NFTTEMP recovery files that match the COPY request are removed.

CREATE LIBMAINTDIR Statement

<create libmaintdir statement>

```
— CREATE LIBMAINTDIR —<options>— FROM —<tape name>—
└ ( —<tape attributes>— ) ┘
```

<options>

```
└ & — /1\— REPORT —
  └ AND — /1\— DSONERROR —
    └ WAITONERROR —
```

<tape attributes>

```
└ /1\— 'TAPE' —
  └ /1\— KIND — = —
  └ /1\— SERIALNO — = —<serial number list>—
  └ /1\— FAMILYOWNER — = —
    └ * —
      └ <usercode>—
  └ /1\— AUTOUNLOAD — = —
    └ ON —
    └ OFF —
    └ DONTCARE —
  └ /1\— CYCLE — = —<integer expression>—
  └ /1\— VERSION — = —<integer expression>—
```

Explanation

The CREATE LIBMAINTDIR statement is used to recreate the LIBMAINTDIR tape directory disk file for a tape or set of tapes when the original LIBMAINTDIR file for those tapes has been accidentally removed or damaged or is not located at the site. When library maintenance executes the CREATE LIBMAINTDIR statement, it places the new copy of the LIBMAINTDIR tape directory disk file on the disk family assigned with the DL LIBMAINTDIR system command.

Note: You cannot use CREATE LIBMAINTDIR to build a LIBMAINTDIR tape directory disk file for a library maintenance tape that was not originally copied with the tape attribute LIBMAINTDIR = TRUE.

Caution

For tape volumes containing large numbers of files, the CREATE LIBMAINTDIR operation can be slow to complete and can cause unwanted CPM consumption. Unisys strongly recommends that you maintain a backup of the DL LIBMAINTDIR family in order to avoid unwanted system resource usage.

When the tape name includes a <string primary>, embedded periods will terminate the tape name unless the entire name is enclosed in quotes.

The following table describes the options available in the CREATE LIBMAINTDIR statement. No statement options are required. You can specify REPORT with either DSONERROR or with WAITONERROR, or by itself. You can specify either DSONERROR or WAITONERROR alone; you cannot specify them together.

Option	Description
DSONERROR	The DSONERROR option causes library maintenance to terminate with a DS response if any errors occur. In this case, library maintenance removes the new LIBMAINTDIR file from the disk.
REPORT	The REPORT option causes library maintenance to print a report of any errors encountered.
WAITONERROR	<p>The WAITONERROR option causes library maintenance to issue an RSVP message whenever an error occurs. Examples of possible errors include:</p> <ul style="list-style-type: none"> • I/O errors reading from the tape • Errors writing to a LIBMAINTDIR disk file <p>The RSVP message halts library maintenance until the operator or programmer responds with OK or DS. A response of OK causes library maintenance to continue building the LIBMAINTDIR disk file. A response of DS causes library maintenance to terminate building the LIBMAINTDIR disk file.</p>

If you specify any file attribute not listed here for a tape used as input to a CREATE LIBMAINTDIR process, then either the WFL compiler issues a syntax error for that attribute or library maintenance ignores the value you specified for that attribute.

The following table describes the CREATE LIBMAINTDIR tape attributes you can use:

Option	Description
SERIALNO	Identifies the specific tapes to be read and the order they should be read. The serial numbers specified and the order in which they are specified, should be the same as the set of tapes originally created.

Statements

Option	Description
FAMILYOWNER	<p>Indicates the usercode of the owner of a tape volume. If you specify a usercode with the FAMILYOWNER attribute, then library maintenance searches for the named tape owned by that usercode. If you do not specify the FAMILYOWNER attribute or you specify the null string (" "), then library maintenance searches for the named tape owned by the usercode of the library maintenance process.</p> <p>If you specify an asterisk (*) for the FAMILYOWNER attribute, then library maintenance searches for the named tape owned by the * usercode. Library maintenance ignores the FAMILYOWNER attribute if the Security Accountability Facility software is not installed, or the TAPECHECK option of the SECOPT system command is not set to AUTOMATIC.</p>
AUTOUNLOAD	<p>Determines whether a tape is unloaded when it is released by library maintenance during a reel switch or a file close operation. If the value is ON, the tape is rewound and unloaded. If the value is OFF, the tape is not unloaded. If the value is DONTCARE or not specified, the tape behavior is controlled by the setting of the AUTOUNLOAD option of the MODE (Unit Mode) system command.</p> <p>Refer to the <i>System Operations Guide</i> for further information. If a reel is switched during a COPY AND COMPARE operation, intermediate reels are not unloaded until the COMPARE phase is finished regardless of the AUTOUNLOAD value.</p>
CYCLE	<p>Designates the specific generation of a tape volume family. This file attribute is used in conjunction with the VERSION attribute. The default value is 1.</p>
VERSION	<p>Designates the successive iteration of the same generation of a tape volume. This file attribute is used in conjunction with the CYCLE attribute. The default value is 0.</p>

Examples

In the following example, a LIBMAINTDIR directory was originally created for the set of tapes in the following COPY statement:

```
COPY & COMPARE SYSTEM/=FROM PACK TO TPACK  
(SERIALNO=(6623, 6624), LIBMAINTDIR);
```

The following statement will read those tapes and create a new copy of the original LIBMAINTDIR tape directory disk file:

```
CREATE LIBMAINTDIR FROM TPACK (SERIALNO=(6623, 6624));
```

CRUNCH Statement

<crunch statement>

```
— CRUNCH — ( —<file identifier>— ) —————|
```

Explanation

The CRUNCH statement closes and crunches a file. Crunching causes the unused portion of the last row (beyond the end-of-file indicator) of disk space to be returned to the system. The file to be crunches must be a disk file. Once a file has been crunches, the file is released and can no longer be expanded.

Examples

The following are examples of the CRUNCH statement:

```
CRUNCH (LONGFILE)
```

```
CRUNCH (BITSFILE)
```

DISPLAY Statement

<display statement>

```
— DISPLAY — <string expression> —————|
```

Explanation

The DISPLAY statement displays and logs the value of the string expression (up to a maximum of 430 characters). The DISPLAY statement accepts up to 1799 characters. If the job was initiated from CANDE, the message is also displayed at the user's terminal.

Examples

The following examples illustrate the DISPLAY statement:

```
DISPLAY "HI THERE"
```

```
DISPLAY PROGNAME & "DID NOT COMPILE"
```

The values of integer and real variables can be displayed by first converting them to string values using the STRING function, as in the following example:

```
DISPLAY STRING (X, *);
```

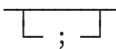
In the preceding example, X can be an integer or real variable. However, if X is a real

variable, any fractional value following the decimal point is rounded off. Refer to [String Expressions](#) for a description of the STRING function.

The DISPLAY statement does not display quotation marks (") around the string value it displays. It also adds a period (.) at the end of the message.

DO Statement

<do statement>

— DO — <statement>  UNTIL — <Boolean expression> —————|

Explanation

The DO statement enables a job to execute a statement until a condition is TRUE.

The statement is executed and the Boolean expression is evaluated. If the expression is TRUE, control passes to the next executable statement; otherwise, the statement is reexecuted.

Notes:

- If the Boolean expression never evaluates to TRUE, the statement is repeated forever (or until the task is discontinued by an operator action or discontinued because it has exceeded queue limits).
- If the same PROCESS statement is executed repeatedly by a DO statement, a run-time error might result. A run-time error can occur if an asynchronous task initiated by an earlier pass through the DO statement is still running. A given task variable can only be used by one task at a time.


Example

Following is a partial job that uses the DO statement:

```
DO BEGIN
    A:=A+1;
    RUN X[T];
    TASKVALUE=A;
END
UNTIL T(TASKVALUE) GEQ 4;
```

GO Statement

<go statement>

— GO —  <label identifier> —————|

Explanation

The GO statement causes job execution to pass directly to the point in the job where the label identifier appears. The label identifier occurs earlier or later in the job than the GO statement. One or more label identifiers can appear in front of any statement in the job. [Statement List](#) for the syntax of label identifier placement.

Notes:

- A GO statement that returns control to an earlier point in the job can result in an infinite loop, unless a conditional statement that exits the loop is also included.
- If the same PROCESS statement is executed repeatedly by a GO loop, a run-time error might result. This can occur because the asynchronous task initiated by an earlier pass through the GO loop might still be running. A given task variable can only be used by one task at a time.

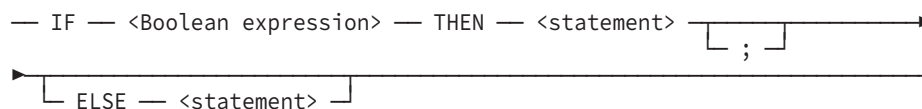
Example

Following is a partial job that uses the GO statement:

```
GO TO L;
.
.
.
L: RUN X;
```

IF Statement

<if statement>



Explanation

The IF statement enables a conditional decision to be made based on the evaluation of a Boolean expression. The statement following THEN is executed if the condition is TRUE. If the condition is FALSE, control passes to the next executable statement. When the ELSE option is specified and the condition is FALSE, the statement following ELSE is executed.

The optional semicolon (;) after the THEN clause does not terminate the IF statement, and thus does not affect the logic of nested IF statements.

For nested IF statements, the WFL compiler matches the first ELSE clause it encounters with the innermost IF statement. This matching can result in a logic error when an IF statement without an ELSE clause is nested within another IF statement that does have an ELSE clause. To guarantee the correct logic, you can either add an ELSE clause containing a null statement, or use a compound statement to enclose the inner IF statement within the reserved words BEGIN and END.

Examples

This first example shows simple IF statements.

```
IF T(TASKVALUE) = 5 THEN
  RUN X;
IF FILE X/Y ISNT RESIDENT THEN
  DISPLAY "NO FILE X/Y"
ELSE
  RUN X/Y;
```

In this example, the first ELSE clause that contains a null statement guarantees the correct logic for the nested IF statements.

```
?BEGIN JOB NESTEDIFS/LOGIC (BOOLEAN B1, BOOLEAN B2);
  IF B1 THEN
    IF B2 THEN
      DISPLAY "Both expressions are TRUE";
    ELSE
      ; % Null statement to guarantee correct logic.
  ELSE
    DISPLAY "Expression 1 is FALSE";
?END JOB.
```

In this example, the compound statement guarantees the correct logic for the nested IF statements.

```
?BEGIN JOB NESTEDIFS/TESTER;
  TASK TASKVAR1, TASKVAR2;
  START NESTEDIFS/LOGIC [TASKVAR1] FOR SYNTAX;
  IF TASKVAR1 IS COMPILEDOK THEN
    BEGIN
      START NESTEDIFS/LOGIC (B1 := FALSE, B2 := TRUE) [TASKVAR2];
      IF TASKVAR2 IS COMPLETEDOK THEN
        DISPLAY "NESTEDIFS/LOGIC compiled and completed";
    END;
  ELSE
    ABORT [TASKVAR1] "NESTEDIFS/LOGIC did NOT compile";
?END JOB.
```

INITIALIZE Statement

<initialize statement>

— INITIALIZE — (— <task identifier> —) —————|

Explanation

The INITIALIZE statement causes the STATUS attribute of the specified task variable to be assigned a value of NEVERUSED and causes all other task attributes and file equations associated with the task variable to be returned to their default values.

This statement should be used in cases where successive task initiation statements in a job make use of the same task variable.

Examples

The following example illustrates such a situation:

```
?BEGIN JOB EXAMPLE;
  TASK T (PRIORITY=80); % Declares task variable with PRIORITY=80
  RUN X [T];           % Runs program X with PRIORITY=80
  INITIALIZE (T);      % Reinitializes task variable
  RUN Y [T];           % Runs program Y with default priority
?END JOB.
```

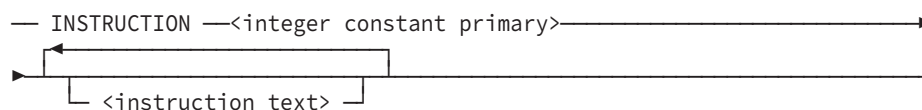
The INITIALIZE statement has the same effect as using a task assignment statement to set the STATUS attribute of a task variable to NEVERUSED, as in the following example:

```
T(STATUS=NEVERUSED); % T is a task variable
```

Refer to [Using Task Variables](#) for further information about task variables.

INSTRUCTION Statement

<instruction statement>



Explanation

The INSTRUCTION statement supplies job instructions to operators. The integer constant primary must be in the range from 1 through 63. If the statement does not include an integer constant primary, a syntax error occurs.

An INSTRUCTION statement must be terminated by a semicolon (;) or the <i> construct. The list of string characters cannot include any semicolons. Once an INSTRUCTION statement has been declared with a number and a string message, you can reuse that instruction (make it the current instruction) by using the "INSTRUCTION <number>;" form of the statement.

As a WFL job executes, any INSTRUCTION statements encountered are stored in a table. When an INSTRUCTION statement is executed, the system marks it as the most current instruction until another INSTRUCTION statement is executed. At any point during execution of the job, any individual instruction can be displayed by number through the IB (Instruction Block) system command. If an instruction number is not specified, the system displays the most current instruction.

For further information, refer to the *System Commands Reference*.

<instruction text>

From 1 to 1500 EBCDIC characters. The characters can be any combination of letters, numbers, blanks, and punctuation except for the semicolon (;) character.

Examples

This first example shows simple INSTRUCTION statements.

```
INSTRUCTION 5 DS MY JOB IF NO FILE A;  
INSTRUCTION 2 MOUNT TAPE TEST3;  
INSTRUCTION 5;
```

The second "INSTRUCTION 5;" statement makes the "INSTRUCTION 5 DS MY JOB IF NO FILE A" the current instruction.

In the next example, the system needs tape TESTTAPE during execution of the COPY statement. If the operator asks for the most recent instruction, instruction 1 is displayed to indicate where TESTTAPE can be found. Subsequently, the job needs files T17 and T17A; an instruction request at that point causes instruction 2 to be displayed with instructions regarding the action to be taken if T17 and T17A are not present.

```
?BEGIN JOB COMPILE/TESTS;  
  FAMILY DISK = USERS OTHERWISE DISK;  
  INSTRUCTION 1 TESTTAPE IS IN TAPE RACK 3.;  
  COPY = FROM TESTTAPE TO USERS(DISK);  
  INSTRUCTION 2 IF T17 OR T17A WERE NOT COPIED FROM TESTTAPE TO  
    USERS, DS THIS JOB AND LEAVE JK A NOTE;  
  COMPILE TEST/17 WITH ALGOL LIBRARY;  
    ALGOL FILE CARD(TITLE=T17, KIND=DISK);  
    FILE F(TITLE=T17A);  
?END JOB.
```

LOCK Statement

<lock statement>

— LOCK — (— <file identifier> —) —————|

Explanation

The LOCK statement closes and locks the specified file and the file is released. The file buffer areas are returned to the system. The logical file is no longer assigned to the physical file. If the file is a tape file, the tape is rewound and unloaded. If the file is not a disk file, the unit is made inaccessible to the system and must be readied again manually. If the file is a disk file, it is kept as a permanent file on disk.

Example

The following is an example of the LOCK statement:

```
LOCK (F1)
```

```

— LOG — [ <LOGANALYZER options> ] [ [ — <task identifier> — ] ]
— [ <task equation list> ]

```

The WFL compiler does not analyze the LOGANALYZER options. This analysis is done by the LOGANALYZER utility.

The optional task equation list can be used to assign attributes to the LOGANALYZER task. For details, refer to [Task Equation](#).

```
LOG
LOG "SYSTEM/SUMLOG" 2359 07/26/90 ALL [I]
LOG /123 2200 07/27/90 TO 0800 07/28/90 HL [T];
PRINTLIMIT = 50
```

```
— MKDIR — <directory name> ┌ ON — <family name> ┐
```

Notes:

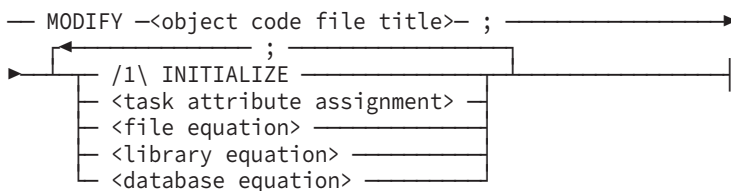
- Only privileged users can create a *DIR directory.
- Users with write and traverse permission to an existing permanent directory can create subdirectories within the existing permanent directory using the MKDIR statement.
- A file and a permanent directory cannot both have the same name.
- After a permanent directory has been created, it cannot be overwritten with a file of the same name that is not a permanent directory.

The OWNER attribute of the directory is set to the usercode of the task. The SECURITYMODE attribute is set to OWNERRWX=RWX, GROUPEWX=X, and OTHERRWX=X by default. The NONUSERFILES option has no effect on the security of newly created permanent directories; however, this option does affect other files created in the permanent directory name space.

To set SECURITYMODE and other security-related attributes to values other than the default, use the PROPAGATESECURITYTODIRS file attribute. Setting this attribute on *DIR or any other permanent directory causes newly created subdirectories to inherit the security attributes of their parent directory.

MODIFY Statement

<modify statement>



Explanation

The MODIFY statement permanently changes the attributes within an object code file. See the *File Attributes Programming Reference Manual* for more information on file attributes.

The MODIFY statement permits task attributes, file equations, library equations, and database equations that are compiled into an object code file to be added to or changed, without recompiling the source file. The attributes listed in the MODIFY statement are permanently stored in the object code file after they have been merged with the previous attributes. If an attribute specified in a MODIFY statement had previously been assigned for that object code file, the new value given in the MODIFY statement is used.

When the MODIFY statement is used with the INITIALIZE keyword, the attributes previously assigned for that object code file are removed. If the INITIALIZE keyword is used, it must precede any task attribute assignments, file equations, library equations, and database equations.

The PRINTPARTIAL file attribute and the REQUESTNAME print modifier cannot be modified.

The file to be modified must be an executable object code file. If you attempt to modify a nonexecutable object code file, the job or processed subroutine containing the MODIFY statement is discontinued.

An object code file that is a compiler must be designated as such after being modified, even if it had previously been designated as a compiler, with the MC (Make Compiler) system command.

The identity specified in the MP (Mark Program) system command is not preserved after the code file is modified. The MP system command needs to be re-applied to a code file that has been altered with the MODIFY statement.

An object code file for which the APL file attribute has the value TRUE cannot be modified. To modify an APL object code file, you must first disable APL access. Change the value of APL to FALSE with the ALTER statement and then modify the code file with the MODIFY statement.

Note: *An object code file associated with a data base (for example, ACCESSROUTINES, DMSUPPORT, and so on) should not be modified with the MODIFY statement. A data base open error may occur when a program accessing the data base is executed.*

When the MODIFY statement creates a new copy of an object code file, the value of the FAMILYINDEX file attribute is not preserved. If you modify a file that should reside on a particular pack, and MODIFY creates a new copy, you might need to recopy the file and specify the appropriate FAMILYINDEX in the COPY statement.

The MODIFY statement can be used to assign initial AX entries or to replace an existing AX entry in a code file. The code file can be assigned multiple initial AX entries by the AX attribute assignments specified in the MODIFY command. If the MODIFY statement is used to replace an existing AX entry and the code file contains multiple AX entries, then the AX entry specified in the MODIFY statement becomes the first in the code file, and the remaining entries are unaffected. If the MODIFY statement specifies more than one AX entry for replacement, only the last is used.

Family substitution is used if the job or task has an active family specification. Only the primary family name is used. Refer to [FAMILY Assignment](#) and [Interrogating Complex Task Attributes](#).

Note: *Using the NAME task attribute as part of the MODIFY statement overrides the file name specified in <object code file title>.*

The following example modifies OBJECT/X:

```
MODIFY OBJECT/X; NAME = OBJECT/Y
```

Examples

This example illustrates how the MODIFY statement changes attributes of the object code file OBJECT/TEST. The attributes are changed so that subsequent runs of OBJECT/TEST have a PRIORITY of 55 and have the substitute family USERS when family substitution is invoked:

```
MODIFY (JONES)OBJECT/TEST;
  PRIORITY = 55;
  FAMILY DISK = USERS OTHERWISE DISK;
```

This example changes the attributes of OBJECT/PROG2 so that subsequent runs of OBJECT/PROG2 use the disk file X for the file INPUT, have a PRIORITY of 45, and have the Boolean attribute SW1 set to TRUE:

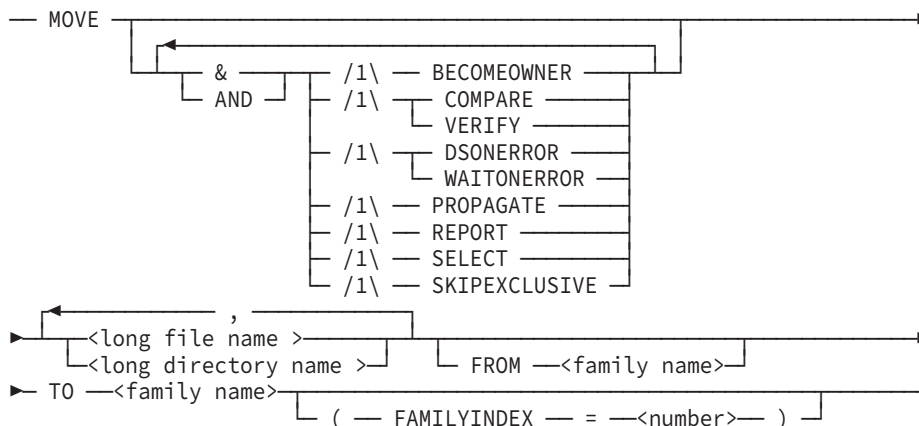
```
MODIFY OBJECT/PROG2;
  FILE INPUT (TITLE = X, KIND = DISK);
  PRIORITY = 45;
  SW1;
```

This example illustrates the MODIFY statement used in conjunction with the INITIALIZE statement:

```
MODIFY PAYROLL/TRIALBALANCE ON ACCOUNTING;
  INITIALIZE; TARGET = 1000; SW8
```

MOVE Statement

<move statement>



Explanation

The MOVE statement causes library maintenance to copy a disk file or files from one disk family to another disk family, and then remove the original file.

When library maintenance successfully copies a file from a source disk to the destination disk, it will remove the original file from the source disk. In addition, it will move any archive backup information for each file copied from the source disk archive directory to

the destination disk directory. On cataloging systems, library maintenance will also move the catalog backup information for each file successfully from the source family catalog directory to the destination file catalog directory.

If there already is a file, catalog, or archive information for the file you are copying on the destination disk, library maintenance will remove the old file from the destination disk including any archive or catalog information for that file.

The MOVE statement will cause the catalog and/or archive backup information to be moved even if there is not a resident version of a requested file on the source family.

The MOVE statement cannot be used for a host/file transfer.

In the following cases, when the file and its backup information are copied to the destination volume, the backup information is purged from the source volume, but the source file is not removed:

- When LOCKEDFILE = TRUE is set
- When a system file is moved

Family substitution is used if the job or task has an active family specification. Only the primary family name is used. Refer to [FAMILY Assignment](#) and [Interrogating Complex Task Attributes](#).

You can use the HI (Cause Exception Event) system command to check the progress of a MOVE statement. A command of the form <mix number> HI displays the number of files already copied and other information.

To avoid conflict with the MOVE system command, a question mark must precede the WFL MOVE statement entered at an ODT. The following options are available in the MOVE statement.

BECOMEOWNER

Controls the ownership of the destination directories and files moved within the permanent directory namespace. This option causes the OWNER attribute to be set to the usercode of the task performing the operation rather than having the value copied from the source.

All other attribute values, including those for GROUP and SECURITYMODE, are copied from the source. If a nonprivileged user issues a COPY or ADD statement without the BECOMEOWNER option, only the source directories and files already owned by that user are included.

COMPARE

Ensures the new copies of the files are written correctly. Ensures the new copies of the files are readable and the data in the copied files matches the data in the source files. This option compares the copied file and the original file bit by bit, immediately after the file is copied.

DSONERROR

Causes library maintenance to terminate with a DS response whenever

- A file or directory to be copied is missing, and library maintenance issues a “file name FILE NOT ON source volume” message.
- An error prevents a file from being copied to the destination.
- Library maintenance issues a “RECOPY REQUIRED” RSVP, and the operator replies with DS, FR, or OF.

FAMILYINDEX

Designates a specific physical volume within a disk family. If you do not specify the FAMILYINDEX attribute, the FAMILYINDEX attribute (if any) of each file being moved is used. For further information, see the [COPY or ADD Statement](#).

PROPAGATE

Enables moved files and permanent directories to inherit the security of the destination directory. This option applies only when the destination PROPAGATESECURITYTOFILES or PROPAGATESECURITYTODIRS attribute of the directory so specifies.

REPORT

Causes library maintenance to print a report of the copied files, including errors. When & REPORT is specified, library maintenance does not write “file copied” messages in the job log or in the system sumlog.

SELECT

Causes library maintenance to call the LIBMAINTSELECTOR procedure in the ARCHIVESUPPORT library for each file to be copied. You or your site can code a special version of the LIBMAINTSELECTOR procedure that indicates if the file should be copied or not.

For information on how to code a custom LIBMAINTSELECTOR procedure, refer to the SYMBOL/ARCHIVESUPPORT DCALGOLSYMBOL file and the *MCP System Interfaces Programming Reference Manual*.

SKIPEXCLUSIVE

Causes the system to not move those files from disk that are opened with EXCLUSIVE=TRUE or that are KEYEDIOII files marked as being updated.

VERIFY

Ensures the new copies of the file were written correctly. Ensures the new copies of the files are readable and that the data was copied accurately. For further details, see [COPY or ADD Statement](#).

WAITONERROR

Causes library maintenance to issue an RSVP message whenever an error occurs. Examples of possible errors include:

- Requesting a file or directory that is missing.
- An error prevents a file from being copied to one or more destinations.
- The RSVP message halts library maintenance until the operator or programmer responds with OK or DS. A response of OK causes library maintenance to continue the copy with other files or tapes. A response of DS will terminate the library maintenance program. After investigating the error that created the RSVP message, you can restart library maintenance.

Null Statement**Explanation**

The null statement is sometimes used in flow-of-control statements when no action is desired. A null statement can be generated within certain flow-of-control statements without using a statement separator, as shown in some of the following examples.

A null statement can also be generated by a statement separator that is preceded only by blank characters or a statement label identifier. The usual statement separator is a semicolon (;) appearing at the end of a statement. However, an invalid character at the start of a line also acts as a statement separator. For details, refer to [Statement List](#).

Examples

In this first example, a CASE statement is used to select which update program (if any) should be run that day. Although no update program should be run on the weekends, the syntax of the CASE statement requires a statement after the selection expression. This requirement is satisfied by the null statement (labeled WEEKEND:).

```
?BEGIN JOB CASE/EXAMPLE(STRING DAY);
  CLASS=2;
  CASE DAY OF
    BEGIN
      ("MONDAY",
       "TUESDAY",
       "WEDNESDAY",
       "THURSDAY"):
        RUN OBJECT/DAILY/UPDATE;
      ("FRIDAY"):
        RUN OBJECT/WEEKLY/UPDATE;
      ("SATURDAY",
       "SUNDAY"):
        WEEKEND:           ;      % No run needed.
      ELSE:
        ABORT "INVALID INPUT STRING:" & DAY;
```

```
END;  
?END JOB.
```

In this example, a null statement (consisting of blank characters after the reserved word THEN) appears within an IF statement. A semicolon can be inserted after the THEN clause, but its presence does not terminate the statement and does not affect the logic of nested IF statements.

```
IF FILE (SAM)DAILY/TOTALS/DATA IS RESIDENT THEN % Continue job.  
ELSE ABORT "Job terminated due to missing data file";
```

In this example, a null statement is unintentionally generated (after the predefined word DO) by the invalid character at the beginning of the next line.

```
?WHILE PRINTCOUNTER LEQ REQCOPIES DO  
? BEGIN  
? PRINT (VIP)ANNUAL/REPORT;  
? PRINTCOUNTER := PRINTCOUNTER + 1;  
? END;
```

ON Statement

<on statement>

```
— ON [ TASKFAULT  
      RESTART ] , — <statement> —
```

Explanation

The ON statement causes the job to execute the specified statement when a task is abnormally terminated (ON TASKFAULT), or when the job is restarted after a halt/load (ON RESTART). The statement specified can be a compound statement if multiple actions are desired.

Only one ON TASKFAULT statement and one ON RESTART statement can be enabled at a time. Thus, any ON TASKFAULT statement disables any previous ON TASKFAULT statement, and any ON RESTART statement disables any previous ON RESTART statement.

If a subroutine executes an ON statement, the ON statement disables any previous ON statement of the same type until the subroutine is finished or the subroutine ON condition is disabled.

The ON TASKFAULT, statement; form enables the condition TASKFAULT. If any task is subsequently terminated abnormally or if a compilation is terminated for syntax errors, the statement is executed. These termination conditions include operator or programmatic discontinuation as well as program faults. If any asynchronous tasks are active when the end of a subroutine or job is reached, the WFL job will automatically wait for them to complete before continuing. If one of the asynchronous tasks terminates abnormally while the WFL job is waiting for all of them to complete, the statement specified with the TASKFAULT option will not be executed until all of the asynchronous tasks are completed.

If a GO TO out of the ON TASKFAULT statement is not performed, the following actions occur:

- Any task fault interrupt that occurs during the execution of the ON TASKFAULT statement is queued.
- When execution of the ON TASKFAULT statement is completed, action is taken on any task faults that were queued. A task fault is selected, and a new invocation of the ON TASKFAULT statement is initiated.
- The ON TASKFAULT statement is invoked for each task fault that becomes queued. Therefore, the ON TASKFAULT statement continues execution until no task faults remain in the queue and either a GO TO out of the ON TASKFAULT statement is executed or the end of the ON TASKFAULT statement is reached.

The statement ON TASKFAULT; disables the condition TASKFAULT. While this statement is in effect, an abnormal task termination has no effect on the job.

Similarly, the RESTART condition can be enabled and disabled by using the following statements:

```
ON RESTART, <statement>; ON RESTART;
```

Exiting the subroutine returns the RESTART condition to the condition before the subroutine was called.

If the enabled ON statement does not execute a GO TO statement, the job resumes execution at the point where it would have continued if the statement had been disabled. The enabled ON statement can only be disabled within the same procedure.

If the system is halt/loaded during the execution of a job, the system first checks to determine if there is a RESTART condition that is enabled. If one is enabled, the statement specified in the condition is executed when the job is restarted. If one is not enabled, the job restarts at the most recent point at which no tasks were running. If there are no asynchronous tasks, this point is just prior to the last task initiation.

One typical use of the ON RESTART statement is to reassign values to the task and file variables in the job. These variables do not retain their values after a halt/load. Refer to [Job Restart After a Halt/Load](#) for further details.

Examples

This first example illustrates the use of the ON TASKFAULT statement:

```
?BEGIN JOB X;
  SUBROUTINE SUB1;
    BEGIN ON TASKFAULT,
      BEGIN
        DISPLAY "SUB TASKFAULT TAKEN";
        GO ERR;
      END; % End of ON statement
  RUN X; % If X aborts, "SUB TASKFAULT TAKEN" is
        % displayed by the ON TASKFAULT.
ON TASKFAULT;
```

```
        RUN Y; % If Y aborts, "JOB TASKFAULT" is displayed.
    END SUB1; % End of subroutine SUB1
        RUN A; % If A aborts, no TASKFAULT is executed.
    ON TASKFAULT, DISPLAY "JOB TASKFAULT";
    RUN B; % If B aborts, "JOB TASKFAULT" is displayed.
    SUB1;
    ERR:
?END JOB.
```

In this example, program P2 is assumed to update a global file (G) created by P1. If a halt/load occurs while P2 is running, the job does not normally rerun P1 but reruns P2. In that case, P2 double updates any records that P2 had updated prior to the halt/load.

Therefore, an ON RESTART statement is executed to ensure that P1 is rerun and that the file is re-created.

```
?BEGIN JOB X;
    FILE G;
    ON RESTART, GO TO L;
L:  RUN P1; FILE F1:=G;
    RUN P2; FILE F2:=G;
?END JOB.
```

In this example, if a halt/load occurs while program P1 is running, the RESTART condition that contains the statement RUN R1 is invoked. If a halt/load occurs while program P2 is running, the RESTART in the outer block that contains the start statement RUN R2 is invoked. This is because at this point the RESTART condition in the subroutine is disabled, returning the RESTART condition to the condition before the subroutine was called.

```
?BEGIN JOB X;
    SUBROUTINE SUB1;
    BEGIN
        ON RESTART,
            RUN R1;
        RUN P1;
        ON RESTART;
        RUN P2;
    END;
    ON RESTART,
        RUN R2;
    SUB1;
?END JOB.
```

In this example, if a halt/load occurs while program P1 or program P2 is running, the RESTART condition that contains the statement RUN R1 is invoked. In the latter case (halt/load while program P2 is running), the disabled ON statement within subroutine SUB1 does not disable the RESTART condition outside of subroutine SUB1.

```
?BEGIN JOB X;
    SUBROUTINE SUB1;
    BEGIN
        RUN P1;
        ON RESTART;
        RUN P2;
    END;
    ON RESTART,
```

```

      RUN R1;
      SUB1;
      ?END JOB.

```

OPEN Statement

<open statement>

```
— OPEN — ( — <file identifier> — ) —————|
```

Explanation

The OPEN statement is used to explicitly open a file according to the value of the file attributes. For more information, refer to [File Handling](#).

Example

The following is an example of the OPEN statement:

```
OPEN (FILEOUT)
```

PASSWORD Statement

<password statement>

```
— PASSWORD — = — <old password> — / — <new password> —|
```

<old password>

<new password>

```
— <name constant> —————|
```

Explanation

The PASSWORD statement changes the password for the current usercode of the job. The password is changed in the USERDATAFILE. The PASSWORD statement cannot be used to change a password on a password-generating system.

Refer to the discussion of MAKEUSER in the *System Software Utilities Operations Reference Manual* and the discussion of the USERDATAFILE in the *Security SDK* for information about the USERDATAFILE and password generation.

Example

The following are examples of the PASSWORD statement:

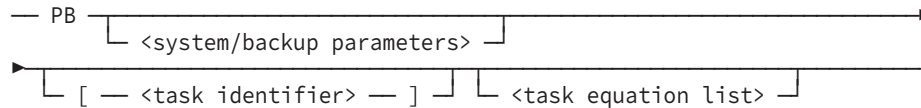
```

PASSWORD = XYZ/ABC
PASSWORD = Old@#$Pass1/New^*Pass23

```

PB Statement

<PB statement>



Explanation

The PB statement initiates the SYSTEM/BACKUP utility, which can be used to print backup files. The PB statement passes the system/backup parameters to SYSTEM/BACKUP; no analysis is done by the WFL compiler. The syntax of the <system/backup parameters> is described in the *Printing Utilities Operations Guide*.

When this statement is entered from the ODT, it must be preceded by a question mark (?) to differentiate it from the PB (Print Backup) system command.

The optional task equation list can be used to assign attributes to the SYSTEM/BACKUP task. For details, refer to [Task Equation](#).

Examples

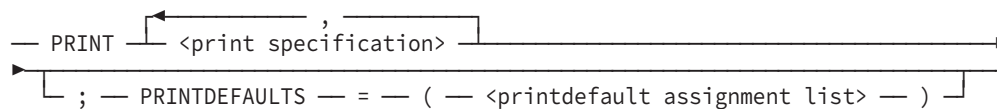
The following are examples of the PB statement:

```
PB MT113 SAVE [T]
```

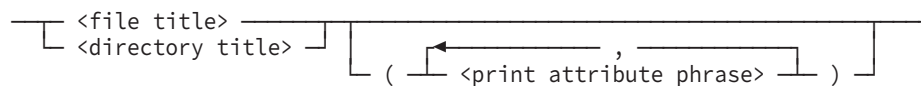
```
PB D 5723 LP11 COPIES 3 SAVE [T]; PRINTLIMIT=500
```

PRINT Statement

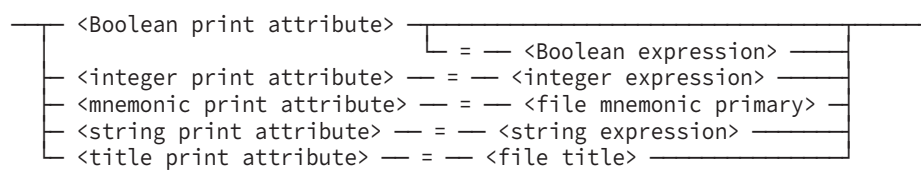
<print statement>

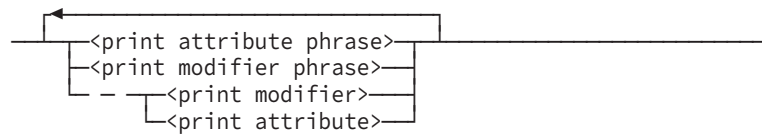
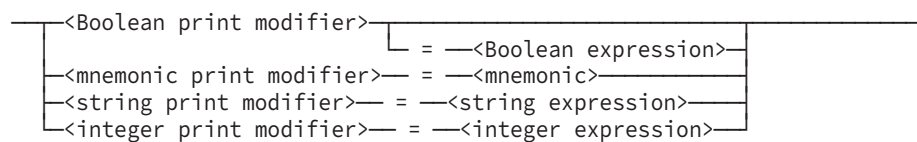


<print specification>



<print attribute phrase>



<printdefaults assignment list>**<print modifier phrase>****Explanation**

The PRINT statement passes a print request to the Print System for processing.

The print specifications in the PRINT statement specify the files to be printed or punched. Each print specification can include its own print attribute phrases, which affect only the specified file or directory.

Various print attribute phrases can be included in the PRINT statement to specify print-related file attributes that control the creation, routing, and formatting of backup files. Refer to print-related task and file attributes in the *Print System User's Guide*, and to general file attributes in the *File Attributes Programming Reference Manual* for more information.

The PRINT statement also enables you to print nonbackup files, such as symbol and data files. Nonbackup files are formatted for printing by the CANDEWRITER transform function supplied in the PRINTSUPPORT library, unless the PRINTPARTIAL file attribute includes a column range specification. However, you can use other transform functions for nonbackup files by specifying them through the TRANSFORM file attribute. The CANDEWRITER transform function formats the lines from a backup file in a format similar to that produced by the CANDE WRITE command. The actual printed format depends on the FILEKIND of the file.

The following tables list the print-related file attributes and shows the correspondence between the types discussed in the *File Attributes Programming Reference Manual* and those discussed in this manual.

File Creation

Print Attribute	File Attribute Type	WFL Type
PRINTCHARGE	Pointer	<string print attribute>
PRINTDISPOSITION	Mnemonic	<mnemonic print attribute>
SAVEPRINTFILE	Boolean	<Boolean print attribute>

Routing

Print Attribute	File Attribute Type	WFL Type
AFTER	Pointer	<string print attribute>
DESTINATION	Pointer	<string print attribute>
PRINTDISPOSITION	Mnemonic	<mnemonic print attribute>
PRINTERKIND	Mnemonic	<mnemonic print attribute>
TRAINID	Mnemonic	<mnemonic print attribute>

AFTER requires a string in the form of a <starttime spec>.

Formatting

Print Attribute	File Attribute Type	WFL Type
ALIGNFILE	Pointer	<title print attribute>
ALIGNMENT	Boolean	<Boolean print attribute>
BANNER	Boolean	<Boolean print attribute>
CHECKPOINT	Boolean	<Boolean print attribute>
FORMID	Pointer	<string print attribute>
NOTE	Pointer	<string print attribute>
PAGECOMP	Pointer	<string print attribute>
PRINTCOPIES	Integer	<integer print attribute>
PRINTPARTIAL	Pointer	<string print attribute>
TRANSFORM	Pointer	<string print attribute>

The maximum length of the FORMID string is 100 characters. The maximum number of PRINTCOPIES is 1000.

Print modifier phrases can be included in a PRINT statement to specify additional requirements for the processing of a print request. Print modifiers can be used with a PRINT statement only through the PRINTDEFAULTS task attribute. For more information, see print modifiers in the *Print System User's Guide*.

The print modifiers and their WFL types are listed in the following table.

Print Modifier	WFL Type
DOUBLESPACE	<Boolean print modifier>

Print Modifier	WFL Type
HEADER	<mnemonic print modifier>
PRINTPRIORITY	<integer print modifier>
REQUESTNAME	<string print modifier>
REQUESTNOTE	<string print modifier>
SUPPRESS	<Boolean print modifier>
TRAILER	<mnemonic print modifier>

A PRINTDEFAULTS specification can be included at the end of a PRINT statement to provide a new set of default values for some print-related file attributes and print modifiers. These values replace defaults that were inherited from the job or from the system.

The print attribute phrases and print modifier phrases that appear in the printdefaults assignment list are merged into the current print defaults. The print modifier and print attribute forms reestablish the system default value for that print modifier or print-related file attribute.

If a print specification and the PRINTDEFAULTS specification both assign values to the same print-related file attribute, then the value assigned in the print specification takes precedence. For details, see the precedence of file attributes and print modifiers in the *Print System User's Guide*.

The PRINTDEFAULTS task attribute can be used to establish default values for print modifiers and some print-related file attributes used in a job or task. This eliminates the need to specify values in each PRINT statement. This attribute is described under [PRINTDEFAULTS Assignment](#), and in the *Task Attributes Programming Reference Manual*.

Printing Portions of a File

It is possible to print selected portions of a file by assigning an appropriate string value to the PRINTPARTIAL file attribute. You can select portions of a file by lines, records, or sequence numbers. LINES is the default clause, since printing lines of a file is usually the desired method. Records are zero-relative, and lines are 1-relative, so that record 0 is equivalent to line 1.

The keyword END corresponds to the last line or record of the file. If only one number is specified for a LINE, RECORD, or SEQUENCE clause, only that record or print line is printed. If two numbers are specified, all records or print lines in that range are printed. Multiple ranges are permitted in ascending order only.

The SEQUENCE clause is valid only for file types that have real sequence numbers; for example, nonbackup files such as symbol files and sequential files. The SEQUENCE clause is not valid for backup files because they do not have sequence numbers. If the SEQUENCE clause is used with a backup file (or a data file without sequence numbers), the sequence numbers are assumed to start at 100 and increment by 100, as CANDE currently does.

You can also select portions of a file by comparing a user-specified text value to a field in each record. If the comparison is true, the record will print.

You can also print a specified range of columns in a file. The start column and end column numbers are integers between 1 and the value of the MAXRECSIZE file attribute. The start column number must be less than or equal to the end column number.

Examples

The following are simple PRINT statements that cause the specified files to be printed:

```
PRINT DRONE/CLONE;  
PRINT (JOHNS)ADD/BACK ON THREEPACK, (CAY)INVENTORY/LIST;
```

The following PRINT statements contain a print attribute phrase as part of the print specification:

```
PRINT (WENDY)FREE/CODE (BANNER=TRUE,  
    NOTE="Review printout for John Smith" );  
PRINT (JAKE)SCRAG/EXTRAS (SAVEPRINTFILE=FALSE,  
    PRINTERKIND=LINEPRINTER);
```

The following PRINT statement includes individual print specifications and common print modifiers as part of the PRINTDEFAULTS specification:

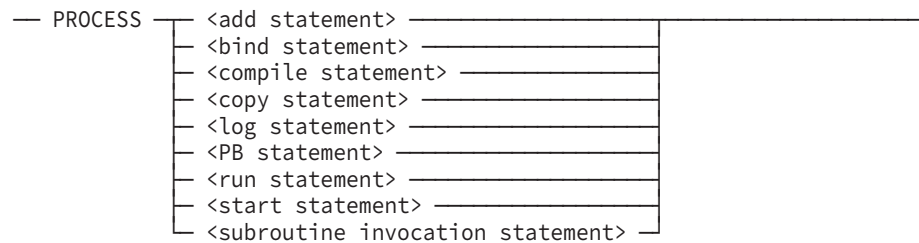
```
PRINT (LIZA)CAVE/DEPTHS (DESTINATION="LP5",PRINTCHARGE=4328),  
    (GEORGE)WATER/COMP (DESTINATION="IP7",PRINTCOPIES=3);  
    PRINTDEFAULTS=(HEADER=SUPPRESSED,TRAILER=UNCONDITIONAL,  
        PRINTPRIORITY=45);
```

The following PRINT statements show the use of the PRINTPARTIAL attribute to print portions of a file:

```
PRINT F1 (PRINTPARTIAL="1, 25-40, 80-END");  
PRINT F1 (PRINTPARTIAL="LINES 1, 25-40, 80-100");  
PRINT F1 (PRINTPARTIAL="RECORD 0, 24-39, 79-END");  
PRINT F1 (PRINTPARTIAL="COLUMN 1-72 SEQUENCE 100-900");  
PRINT F1 (PRINTPARTIAL="SEQ 100-900 @ 1-72");  
PRINT F1 (PRINTPARTIAL="WHERE COL 1-5 = 'NAME:'");
```

PROCESS Statement

<process statement>



Explanation

The PROCESS statement initiates tasks asynchronously. The job does not terminate until all asynchronous tasks have terminated. For more information about task variables, see [Using Task Variables](#).

A task equation list and a task variable can usually be specified when using PROCESS with the statements listed in the syntax diagram. Restrictions are summarized in [Task Equation](#).

If a task variable is attached, a WAIT statement can be used to cause the job to wait for the task to terminate, to reach a particular task state, or to have a particular task attribute value. Refer to [WAIT Statement](#) in this section for details. Many attributes of a task can be altered while the task is active by using the task assignment statement. Refer to [Assignment Statements](#) earlier in this section for the syntax of the task assignment statement.

The name of a subroutine task initiated by a PROCESS statement is set by WFL when the subroutine is invoked. Prior settings of the NAME task attribute are overridden, and the default name of the subroutine task is the same as the name of the subroutine. The name of the subroutine task can be changed by specifying a different value for the NAME task attribute when the subroutine is invoked.

Care should be taken when using global variables in an asynchronous subroutine, or in a program initiated by a PROCESS RUN with passed-by-reference parameters. This is because the subroutine or program executes in parallel with the rest of the job, and all of the parallel processes that are executing can access the variable at the same time. If one process changes the value of the variable, other processes that interrogate the variable will get the new value.

Note: A run-time error will result if the PROCESS statement is executed in either of the following ways:

- Repeatedly with the same task variable and the previously processed task is still active.
- In a loop (one that uses the DO or WHILE statement, for example) with no task variable and the previously processed task is still active.

Examples

The following are simple examples of the PROCESS statement:

```
PROCESS RUN X/Y (3,I) [T]
PROCESS SUB; CORE =100
PROCESS COPY A TO B [T]
```

This example initiates two copy tasks asynchronously and causes the job to wait for both of them to finish before continuing:

```
PROCESS COPY (JAS)= FROM PARTS1 (TAPE) TO DATAPK (DISK) [T1];
PROCESS COPY (JAS)= FROM PARTS2 (TAPE) TO DATAPK (DISK) [T2];
DO
  WAIT
UNTIL T1 IS COMPLETED AND T2 IS COMPLETED;
```

This example runs two programs asynchronously and waits for the first one to complete. According to the TASKVALUE of the first program, the attributes of the second program are changed or else it is aborted:

```
PROCESS RUN PROG/1 [T1];
PROCESS RUN PROG/2 [T2];
WAIT (T1 IS COMPLETED);
IF T2 ISNT COMPLETED THEN
  BEGIN
    IF T1 (TASKVALUE) = 1 THEN
      T2 (SW1=TRUE, OPTION=(ARRAY, DSED))
    ELSE
      ABORT [T2] "T2 ABORTED";
  END;
```

In this example, the prior value of TESTSUB for the NAME task attribute (set with the task variable T) is overridden when the subroutine SUB1 is invoked by a PROCESS statement. The name of the subroutine task is changed from its default value of SUB1 to the value of TEST1:

```
SUBROUTINE SUB1;
  BEGIN
    RUN TEST/1;
  END;
T(NAME=TESTSUB);
PROCESS SUB1[T];
  NAME=TEST1;
```

PURGE Statement

<purge statement>

— PURGE — (— <file identifier> —) —————|

Explanation

The PURGE statement closes, purges, and releases the specified file to the system. If the file is a permanent disk file, it is removed from the disk directory, and the disk space is returned to the system.

The file to be purged must first be explicitly opened by the OPEN statement.

Examples

The following are examples of the PURGE statement:

```
PURGE (ALLFILES)
```

```
PURGE (ONEFILE)
```

RELEASE Statement

<release statement>

```
— RELEASE — ( — <file identifier> — ) —————|
```

Explanation

The RELEASE statement closes and releases the specified file. The file buffer areas are returned to the system. The logical file is no longer assigned to the physical file. If the file is a temporary disk file, the disk space is deallocated. If the file is a tape file, it is rewound.

The tape is unloaded if the AUTOUNLOAD file attribute has the value ON, or, if the AUTOUNLOAD file attribute has the value DONTCARE and the AUTOUNLOAD mode of the tape unit is ON.

Examples

The following are examples of the RELEASE statement:

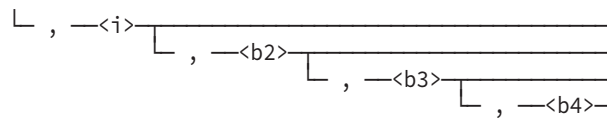
```
RELEASE (GLOBAL);
```

```
RELEASE (FILEA);
```

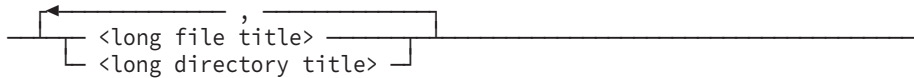
REMOVE Statement

<remove statement>

```
— REMOVE — [ — DESTROY — ] —————→
|
| — <remove list> —————→
|   |
|   | — <remove from group> —————→
|   |   |
|   |   | — , — <remove list> —————→
|   |   |
|   | — [ — <b1> ————— ] —————→
|   |
|   |
```



<remove list>



<remove from group>



Explanation

The REMOVE statement removes files from disk and marks the archive backup records (if there are any) indicating that the files are “removed.” The removed files cannot be restored by the archive AUTORESTORE feature, but they can be restored by using the WFL “ARCHIVE RESTORE” and “ARCHIVE ADDRESTORE” statements. For information regarding how to remove and restore files using the archive AUTORESTORE feature, refer to [ARCHIVE RELEASE Statement](#) earlier this section.

The DESTROY option causes all catalog and archive directory records for the file(s) to be purged. In addition, the catalog and archive records will be purged for named files and directories even if the files are not resident. The REMOVE DESTROY statement is equivalent to a REMOVE statement followed by a CATALOG PURGE statement followed by an ARCHIVE PURGE statement. Files that are removed with the DESTROY option cannot be restored with the WFL statement ARCHIVE RESTORE.

If a directory is specified, all files in that directory are removed. The directories *= and = cannot be used in the REMOVE statement.

If the REMOVE command is used to remove a file whose LOCKEDFILE file attribute is set to TRUE, that file is not deleted. The system displays the following message to indicate that the file was not removed:

```
<file name> NOT REMOVED (LOCKEDFILE).
```

See the ALTER statement in this section for more information about changing the LOCKEDFILE file attribute. For more information about the LOCKEDFILE file attribute, refer to the *File Attributes Programming Reference Manual*.

The first name in a remove list can be a file title or directory title; that is, it can contain an ON family name part. Subsequent names in the remove list can only contain an ON family name part if they contain a string primary as well.

In the remove from group, the FROM clause applies to all the file names and directory names in that remove from group.

Family substitution is used if the job or task has an active family specification. Only the

primary family name is used. Refer to [FAMILY Assignment](#) and [Interrogating Complex Task Attributes](#).

Path substitution is used if the job has an active DATAPATH attribute specification. Only the first path element is used. For more information on active DATAPATH attribute specification, refer to [DATAPATH Assignment](#) and [Interrogating Complex Task Attributes](#).

Remove Results

The REMOVE statement can be followed by an optional results structure: [b1, i, b2, b3, b4]. Parameters are optional. For example [b1, i] is syntactically correct. However, if a parameter is specified, all preceding parameters must also be specified. The parameters return the following information:

Variable	Type	Description
b1	Boolean	TRUE if a failure occurred. FALSE if all files and directories were removed.
i	Integer	Total number of removed files
b2	Boolean	TRUE if at least one file or directory was not removed because it was not present.
b3	Boolean	TRUE if at least one file was not removed because the file was a locked file.
b4	Boolean	TRUE if at least one file was not removed because of a security error.

Examples

The following examples illustrate the REMOVE statement syntax.

This statement removes the file X from DISK:

```
REMOVE X;
```

This statement removes the file A/B from USERS:

```
REMOVE A/B ON USERS;
```

These statements remove all the files under the directory A/= from USERS, and remove all the files under the directory B/= from PACK:

```
S1:="A/= ON USERS";
S2:="B/= ON PACK";
REMOVE #S1, #S2;
```

These statements remove the file X from DISK, and remove all the files under the directory Y/= from PACK:

Statements

```
S:="Y/=";  
REMOVE X, #S ON PACK;
```

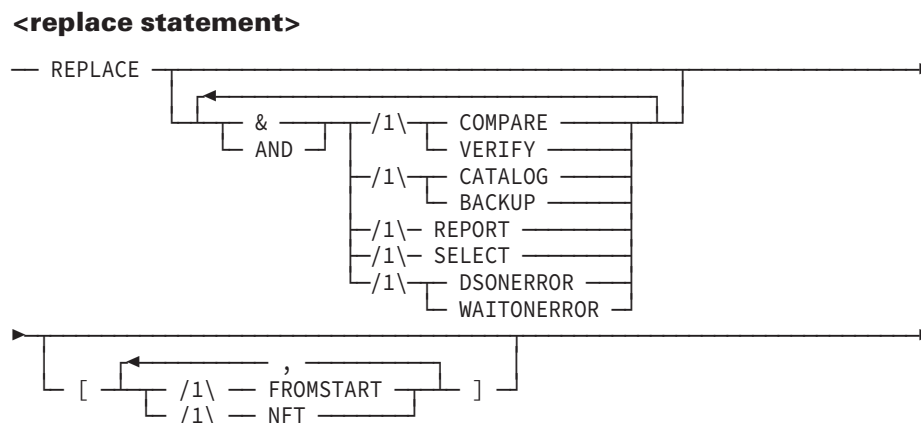
These statements remove the file X from DISK, the file Y from MYPACK, the file X/Y from PACK, and the file Z from DISK:

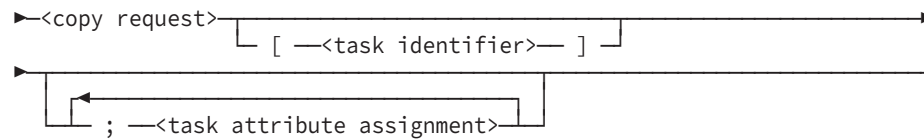
```
S:="Y";  
REMOVE X, #S FROM MYPACK, X/Y FROM PACK, Z;
```

This job demonstrates the use of the optional results:

```
BEGIN JOB DEMONSTRATE/REMOVE;  
BOOLEAN BFAILED, BMISSING, BLOCKED, BSEC;  
INTEGER COUNT;  
  
REMOVE CHANGETEST3/=,  
        CHANGETEST2/=,  
        CHANGETEST7  
        [BFAILED, COUNT, BMISSING, BLOCKED, BSEC];  
  
DISPLAY "Remove test: " & STRING (COUNT, *) & " files removed";  
IF NOT BFAILED THEN  
    DISPLAY "All files removed";  
IF BMISSING THEN  
    DISPLAY "Missing files";  
IF BLOCKED THEN  
    DISPLAY "Locked files seen";  
IF BSEC THEN  
    DISPLAY "Security error seen";  
REMOVE CHANGETEST3/=,  
        CHANGETEST2/=,  
        CHANGETEST7  
        [BFAILED, COUNT]; % Not all variables are needed  
DISPLAY "Remove test: " & STRING (COUNT, *) & " files removed";  
IF NOT BFAILED THEN  
    DISPLAY "All files removed";  
END JOB
```

REPLACE Statement





Explanation

The REPLACE statement enables you to replace existing copies of files on disk with new copies of those files. The REPLACE statement is similar to the COPY statement, except that REPLACE copies only those files that you specify for which there are already existing copies on the destination disk or disks.

You cannot use the REPLACE statement to copy files to tape or CD-ROM.

For a detailed explanation of the <copy request> syntax, refer to the [COPY or ADD Statement](#).

Example

The following example copies files under the directory SYMBOL/= from the tape SYMTAPE to disk SYMBOLS and to PACK. For each destination, library maintenance copies only those SYMBOL files from the tape that already have existing versions on the destination. These existing versions are replaced with new copies from the SYMTAPE.

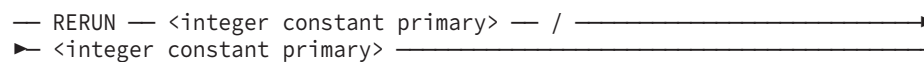
```

REPLACE & VERIFY SYMBOL/= FROM SYMTAPE TO PACK,
                                     TO SYMBOLS (PACK);

```

RERUN Statement

<rerun statement>



Explanation

The RERUN statement restarts a program at a specified checkpoint. The first integer constant primary is the job number. The second integer constant primary indicates which checkpoint files are used for the restart.

Example

The following is an example of the RERUN statement:

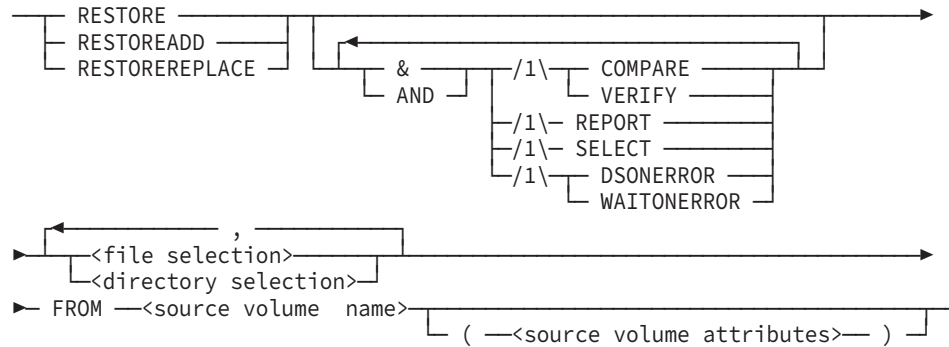
```

RERUN 3555/3

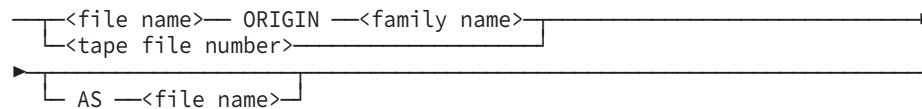
```

RESTORE Statement

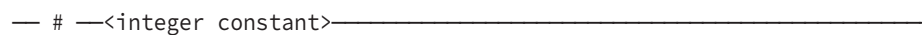
<restore statement>



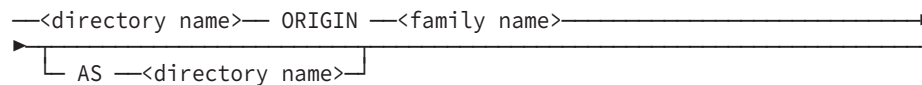
<file selection>



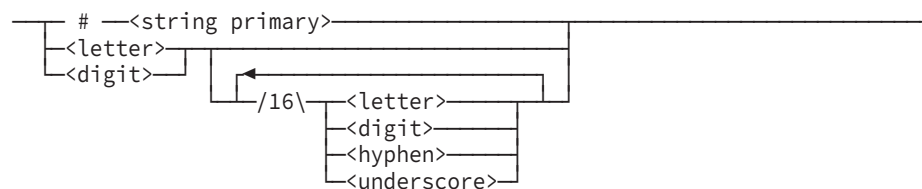
<tape file number>



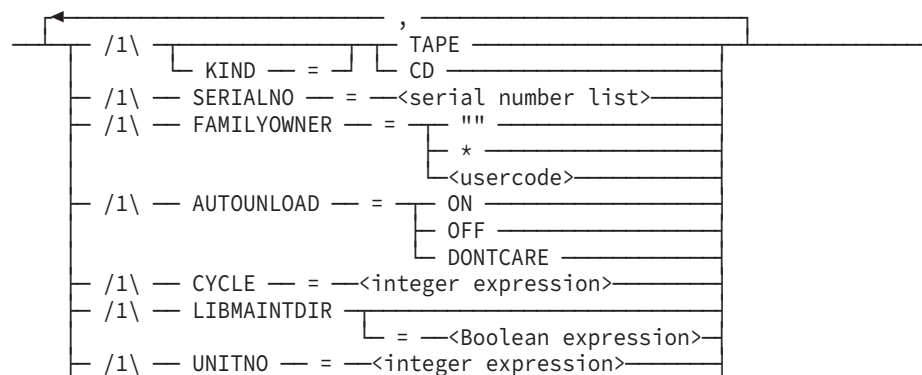
<directory selection>



<source volume name>



<source volume attributes>



└─ /1\ ─ VERSION ─ = ─<integer expression>──┐

Explanation

The RESTORE statement is used to copy files from a library maintenance tape or CDROM volume to the disk families from which they were originally copied.

Note: WFL includes another statement called ARCHIVE RESTORE, rather than simply RESTORE. The two statements serve different purposes. The ARCHIVE RESTORE statement reloads files that have archive backup directory entries. The RESTORE statement reloads files from any library maintenance tape or CD-ROM volume, regardless of whether or not the files have archive backup directory entries.

The family name you specify after the key word ORIGIN for a given file or directory name indicates the name of the disk family from which that file was copied when library maintenance created the input tape or CD-ROM. The input tape or CD-ROM may contain files which were copied from several different disk families. The family names listed in the library maintenance tape or CD-ROM directories are the actual disk family names from which those files were copied. In the case of a library maintenance copy from disk to tape or CD-ROM that is run under a family substitution statement, the disk family names stored in the tape or CD-ROM directory are those used after applying the family substitution.

The RESTORE statement enables you to copy files from tape or CD-ROM to disk when the tape or CD-ROM contains files copied from several different disk families. You should use the RESTORE statement if you want to copy all of the specified files and directories.

The RESTORE statement invokes the *LIBRARY/MAINTENANCE process. If an error occurs in the restore process, library maintenance sets the TASKVALUE task variable to a non-zero value; if no errors occur, library maintenance sets the TASKVALUE task variable to zero.

The RESTOREADD variation enables you to copy specified files that do not have existing copies on the disk. The RESTOREREPLACE variation enables you to copy and replace specified files that do already have existing copies on the disk.

If a restore request specifies a file name or directory name that selects one of the files that does not have an original disk family name listed in the tape or CD-ROM directory, you will receive the following error message:

```
MT <unit number> NO ORIGINAL FAMILY NAME FOR <file name>,
FILE WILL NOT BE RESTORED.
```

The restore process does not copy the named file, but does continue to restore other files. Family substitution can be used in the job which contains the RESTORE statement to redirect the copy for one origin family to a different family. The tape or CD-ROM name you specify indicates the tape or CD-ROM or set of tapes or CD-ROMs that contain the files you want to copy back to disk. You can only specify the name of one input in a RESTORE statement.

You can use the HI (Cause Exception Event) system command to check the progress of a RESTORE statement. A command of the form <mix number> HI displays the number of files already copied and other information.

The <tape file number> option specifies the position number of the file on the source tape or source CD-ROM. The position number of the first file on a tape is 1, and so forth. The <tape file number> must be a positive integer that is greater than zero (0) and has less than 12 digits (counting leading zeros). You cannot use <tape file number> in file transfer copy statements (copy statements with <transfer service> or HOSTNAME clauses). You cannot specify ORIGIN <family name> for a <tape file number>.

When the tape name includes a <string primary>, embedded periods will terminate the tape name unless the entire name is enclosed in quotes.

Library Maintenance

When you use library maintenance to copy files from one library maintenance tape to a new library maintenance tape, the disk family names in the directory for the new tape are the same as those in the original tape directory. The tape directory of a library maintenance tape created by an NFT file transfer from one host to a tape at another host will contain the original disk family names, but not the original host name. A restore from such a tape will attempt to copy the files to the disk families with the proper names at the host where the restore statement is executed.

If you attempt to use a RESTORE statement with one of these tapes, you will receive the following error message and the RESTORE process will terminate:

```
MT <unit number> TAPE DIRECTORY DOES NOT CONTAIN DISK  
FAMILY NAMES, RESTORE TERMINATED
```

RESTORE Statement Options

The following options are available in the RESTORE statement.

COMPARE

The COMPARE option compares the copied files and the resident files bit by bit immediately after the files are copied. You can use the COMPARE option to ensure that the new copies of the files are readable, and that the data in the new files matches the data in the source files.

DSONERROR

The DSONERROR option causes the system to discontinue the library maintenance program if an error is detected. This option will cause library maintenance to terminate with a DS response whenever:

- A file or directory to be copied is missing and library maintenance issues a "filename FILE NOT ON <source volume>" message.
- An error prevents a file from being copied to its destination.
- Library maintenance issues a "RECOPY REQUIRED" RSVP and the operator replies with DS, FR, or OF.

REPORT

The REPORT option causes library maintenance to print a report of the files it copied and any errors encountered. When & REPORT is specified library maintenance does not write “file copied” or “file not copied” messages in the job log or the system sumlog.

SELECT

Causes library maintenance to call the LIBMAINTSELECTOR procedure in the ARCHIVESUPPORT library for each file to be copied. You or your site can code a special version of the LIBMAINTSELECTOR procedure that indicates if the file should be copied or not.

For information on how to code a custom LIBMAINTSELECTOR procedure, refer to the SYMBOL/ARCHIVESUPPORT DCALGOLSYMBOL file and the *MCP System Interfaces Programming Reference Manual*.

VERIFY

The VERIFY option is similar to the COMPARE option. However, instead of comparing the copied files bit by bit, the files are read again, and the overall checksum is compared.

Both the COMPARE and the VERIFY options ensure that the new copy of the files is written correctly.

UNITNO

Designates the assigned hardware unit number of the tape volume to be used.

WAITONERROR

The WAITONERROR option causes library maintenance to issue an RSVP message whenever an error occurs. Examples of possible errors include:

- Requesting a file or directory that is missing.
- Attempting to open a failed tape. The RSVP message halts library maintenance until the operator or programmer responds with OK or DS. A response of OK causes library maintenance to continue restoring other files or tapes. A response of DS will terminate the library maintenance program. After investigating the error which created the RSVP message, you can restart library maintenance.

RESTORE Tape and CD-ROM Attributes

If you specify any attribute not listed here for a tape or CD-ROM volume used as input to a RESTORE process, then either the WFL compiler issues a syntax error for that attribute or library maintenance ignores the value you specified for that attribute.

The following tape and CD-ROM attributes are available for the RESTORE statement.

AUTOUNLOAD

Determines whether or not a tape is unloaded when it is released by library maintenance during a reel switch or a file close operation. If the value is ON, the tape is rewound and unloaded. If the value is OFF, the tape is not unloaded.

If the value is DONTCARE, or if this value is not specified, the tape behavior is controlled by the setting of the AUTOUNLOAD option of the MODE (Unit Mode) system command. Refer to the *System Operations Guide* for further information.

If a reel is switched during a COPY AND COMPARE operation, intermediate reels are not unloaded, regardless of the AUTOUNLOAD value, until the COMPARE phase is finished.

CYCLE

Designates the specific generation of a tape volume family. This file attribute is used in conjunction with the VERSION attribute. The default value is 1.

FAMILYOWNER

Indicates the usercode of the owner of a tape volume. If you specify a usercode with the FAMILYOWNER attribute, then library maintenance searches for the named tape owned by that usercode.

If you do not specify the FAMILYOWNER attribute or if you specify the null string (" "), then library maintenance searches for the named tape owned by the usercode of the library maintenance process itself.

If you specify an asterisk (*) for the FAMILYOWNER attribute then library maintenance searches for the named tape owned by the * usercode. Library maintenance ignores the FAMILYOWNER attribute if the Security Accountability Facility software is not installed, or if the TAPECHECK option of the SECOPT system command is not set to AUTOMATIC.

LIBMAINTDIR

Determines if library maintenance should use the information in the tape directory disk file of source tapes to speed up the search for files.

LOCATECAPABLE

Set the LOCATECAPABLE attribute to ON to indicate that the file requires a tape drive capable of processing the READ POSITION and LOCATE BLOCK ID tape commands for fast tape access.

If the assigned tape drive is locate capable, then library maintenance automatically takes advantage of this feature to do high-speed spacing in the following situations:

- COMPARE Option

Library maintenance uses the LOCATE BLOCK ID tape command to backspace to compare the file. If you receive a RECOPY REQUIRED message and respond "OK," library maintenance uses LOCATE BLOCK ID to backspace to the beginning of that file to recopy the file.

- LIBMAINTDIR for Source Tapes

If the original tape was created on a locate capable tape drive, and a LIBMAINTDIR directory was created for the tape, then library maintenance uses the LOCATE BLOCK ID information found in the LIBMAINTDIR directory to rapidly space up to each of the files to be copied.

SERIALNO

Identifies the specific tape or CD-ROM volumes to be used when copying files.

To copy a few specific files from a multi-reel library maintenance tape set, you can use this attribute to skip directly to the reel holding the file(s) you need. If you know which reel contains the file you want (or the first reel that contains one of the files you want), use the serial number of that tape volume for the value of the SERIALNO attribute.

The SERIALNO attribute does not have a default value. For more information, refer to [Serial Number Assignment](#).

VERSION

Designates the successive iteration of the same generation of a tape volume. This file attribute is used in conjunction with the CYCLE attribute. The default value is 0.

Examples

The following example will restore SYSTEM/ALGOL and SYSTEM/COBOL85 to the disk family CODE if those files are not currently resident on the family CODE and if the files were previously copied from the family CODE to a tape name CODETAPE:

```
RESTOREADD SYSTEM/ALGOL ORIGIN CODE, SYSTEM/COBOL85 ORIGIN
CODE FROM CODETAPE (SERIALNO= (6622, "6622A") , AUTOUNLOAD=ON);
```

You can use the following form of the RESTORE statement to copy all files from a tape to their original families. To use the *= construct, the statement must be started from an ODT, or it must run under a privileged usercode.

```
RESTORE & COMPARE *= ORIGIN DISK, *= ORIGIN PACK FROM SAVETAPE;
```

In the preceding example, any files copied to SAVETAPE from the family DISK are copied back to DISK, and any files copied to SAVETAPE from the family PACK are copied back to PACK.

You can use the following WFL statements to copy certain files from CD-ROM to a new disk family:

```
FAMILY OLDFAM = NEWFAM ONLY;  
RESTORE DATA/= ORIGIN OLDFAM,  
PROG/=ORIGIN OLDFAM FROM XPORT(CD);
```

These statements select files under the directories DATA/= and PROG/= that were originally copied from the family OLDFAM to the CD-ROM XPORT. The selected files are copied from the CD-ROM XPORT to the disk family NEWFAM.

The following example replaces all the resident SYMBOL files with the backup copies of those files:

```
RESTOREREPLACE SYMBOL/= ORIGIN SYSPACK FROM SYSTAPE;
```

RETURN Statement

<return statement>

— RETURN —————|

Explanation

The RETURN statement makes an early return from a subroutine or makes an early return from the invocation of an ON condition. If a RETURN statement is executed in a subroutine that was called synchronously, execution passes to the statement following the subroutine invocation statement. If a RETURN statement is executed in an asynchronous subroutine, the subroutine terminates normally.

Example

The following is an example job that uses the RETURN statement:

```
SUBROUTINE COMPSUB;  
BEGIN  
  TASK T;  
  COMPILE OBJECT/X WITH COBOL74 [T] LIBRARY;  
  COMPILER FILE CARD(KIND=DISK,TITLE=X);  
  IF T ISNT COMPILEDOK THEN  
    RETURN;  
  RUN OBJECT/X;  
END COMPSUB;
```

REWIND Statement

<rewind statement>

— REWIND — (— <file identifier> —) —————|

Explanation

The REWIND statement closes the specified file. The logical file remains assigned to the physical file. If the file is a tape file, it is rewound. For disk files, the record pointer is reset to the first record of the file. The file buffer areas are returned to the system. The I/O unit remains under program control.

Examples

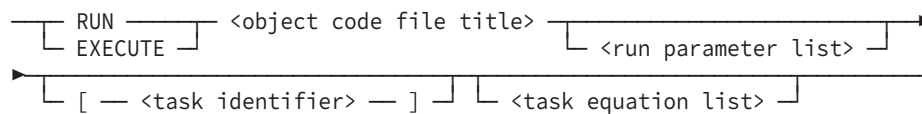
The following are examples of the REWIND statement:

```
REWIND (GLOBAL)
```

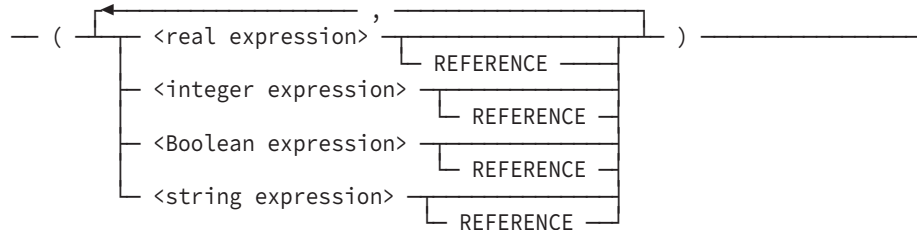
```
REWIND (FILEA)
```

RUN Statement

<run statement>



<run parameter list>



Explanation

The RUN statement initiates a previously created object code file. If the object code file title specified is not an executable object code file, the job or processed subroutine is discontinued.

Parameter checking is done against the object code file, and any mismatches cause the job to be discontinued. By default, all parameters are passed by value. The keyword REFERENCE indicates that the parameter is passed by reference rather than by value.

Note: REFERENCE is valid only if the preceding expression is an identifier.

When a parameter has been passed by reference, the actual parameter is updated whenever the formal parameter is changed. If the WFL job that initiated the program inquires on the value of a passed-by-reference parameter while the program is running, the latest value is returned. The receiving program of a passed-by-reference string parameter should not resize the parameter smaller than 300 words (1800 characters.)

Strings consist of EBCDIC characters and are passed as arrays. If a string expression is passed by value, an extra NUL (48"00") character is added to the end of the string to enable the object program to determine the length of the string. This string can be up to 1800 characters long. If a string identifier is passed by reference, a real array large enough to hold the maximum size of the string is passed with NUL characters padded to the end of the array. The contents of the array can be changed in the object program. If the task is initiated by a RUN statement, WFL updates the length of the string by locating the first NUL character in the array.

If the task is initiated by a PROCESS RUN, the length of the string cannot be changed by the processed task.

A task equation list can be given to override any task attribute assignments, file equations, and database equations set when the object code file was created. Refer to [Task Equation](#) and "Run-Time Overriding of Compiler Task Equation" under [Local Data Specifications](#).

Note: Using the NAME task attribute as part of the RUN statement overrides the file name specified in <object code file title>.

The following example runs OBJECT/Y:

```
RUN OBJECT/X; NAME = OBJECT/Y
```

Examples

The following are examples of simple RUN statements:

```
RUN X;
RUN A/B [T];
RUN A/C (1,I,FALSE,3.14,OCTAL ("123"), "HI THERE") [T1];
    FILE F(BLOCKSIZE=10, KIND=TAPE);
    FILE G(KIND=DISK);
RUN A/D (COUNT REFERENCE, TRUE, STR1 REFERENCE);
RUN A/E; TASKVALUE=I;
```

The following examples depict different methods of specifying early AXs in a RUN statement. The ability to supply an early AX command with a RUN statement is always available. Multiple AXs, however, will be passed to the program by the MCP only if the QUEUEDAX system option is set (SYSOPS QUEUEDAX SET).

If multiple early AXs are entered for a program while QUEUEDAX is RESET, then only the final AX is actually passed to the program.

The following job queues a 1-byte AX message, specify a priority of 70, and queues a 13byte AX message behind the first AX message. All three task assignments are for task T. When PROGA encounters an ACCEPT statement, it will accept "1" first, and then accept "MESSAGE THREE" when it encounters the next ACCEPT statement.

```
?BEGIN JOB EXAMPLE/EARLYAX;
TASK T (AX="1", PRIORITY=70, AX="MESSAGE THREE");
RUN PROGA[T];
?END JOB.
```

The following job queues a 1-byte AX message, specifies a priority of 70, and queues an 11byte AX message behind the first AX message. All three task assignments are for task PROGB. When PROGB encounters an ACCEPT statement, it will accept the "1" first, and then accept the "MESSAGE TWO" when it encounters the next ACCEPT statement.

```
?BEGIN JOB EARLYAX;
  STRING S := "MESSAGE TWO";
  RUN PROGB;AX="1";PRIORITY=70;AX=S;
?END JOB.
```

The following statement is the same as the first example, except that it is a PROCESS RUN statement:

```
PROCESS RUN PROGA[T];
```

Note: AX input relates to the next ACCEPT encountered in the stack, either by library code or the program's code. This can cause AX messages to be unintentionally routed to a library when they were intended for the client task.

Be careful when using the AX attribute with a task that will attach an interrupt to the ACCEPTEVENT. The ACCEPTEVENT will already be in the happened state before the task can possibly enable the interrupt. For further information, refer to the *Task Management Programming Guide*.

The following example shows the use of WFL-provided parameters for a program written in ALGOL:

```
?BEGIN JOB EXAMPLE/ALGOL;
  INTEGER I;
  STRING STR := "WFL STRING";
  COMPILE ALG WITH ALGOL LIBRARY;
ALGOL DATA CARD
$ SET LEVEL 2
PROCEDURE D(WFLINTEGER, WFLSTRING, WFLBOOLEAN);
  INTEGER WFLINTEGER;
  ARRAY WFLSTRING[*];
  BOOLEAN WFLBOOLEAN;
BEGIN
  ARRAY A[0:49];
  WFLINTEGER := 20;
  REPLACE POINTER(A) BY "WFLSTRING = ",
                        POINTER(WFLSTRING) FOR 256 UNTIL=48"00",
                        48"00";

  DISPLAY (POINTER(A));
  REPLACE POINTER(WFLSTRING) BY "WFL STRING HAS BEEN CHANGED",
                                48"00";

END.
? % END OF COMPILER DATA
  RUN ALG (I REFERENCE, STR REFERENCE, TRUE);
  DISPLAY "I = "& STRING(I, *); % I = 20
  DISPLAY "STR = "& STR; % STR = "WFL STRING HAS BEEN CHANGED"
?END JOB.
```

The following example shows the use of WFL-provided parameters for a program written in COBOL74. Programs written in COBOL74 cannot specify Boolean parameters.

```
?BEGIN JOB EXAMPLE/COBOL74;
STRING STR := "WFL STRING";
COMPILE COBOL74/EXAMPLE WITH COBOL74 LIBRARY;
COBOL74 DATA CARD
100100IDENTIFICATION DIVISION.
100200ENVIRONMENT DIVISION.
100300DATA DIVISION.
100400WORKING-STORAGE SECTION.
10050077 WFLINTEGER BINARY PIC 9(11) RECEIVED BY REFERENCE.
10060001 WFLSTRING RECEIVED BY REFERENCE.
100700    03 MSG          PIC X(30).
10080001 CHARMESSAGE     PIC X(30).
100900PROCEDURE DIVISION USING WFLINTEGER, WFLSTRING.
101000P1.
101100    STRING WFLSTRING DELIMITED BY LOW-VALUE
101200          INTO CHARMESSAGE.
101300    DISPLAY "WFLSTRING = ", CHARMESSAGE.
101400    MOVE "WFL STRING HAS BEEN CHANGED" TO WFLSTRING.
101500    STOP RUN.
? % END OF COBOL74 DATA CARD
  RUN COBOL74/EXAMPLE (1234, STR REFERENCE);
  DISPLAY "STR = "& STR;
              % STR = "WFL STRING HAS BEEN CHANGED "
?END JOB.
```

The following example shows the use of WFL-provided parameters for a program written in Pascal:

```
?BEGIN JOB EXAMPLE/PASCAL;
STRING STR := "WFL STRING";
BOOLEAN BOOL := TRUE;
COMPILE PASC WITH PASCAL LIBRARY;
PASCAL DATA CARD
PROGRAM p((VAR WFLInteger:INTEGER; VAR WFLString:WFLStringType;
           VAR WFLBoolean: BOOLEAN));
TYPE WFLStringType = RECORD str:PACKED ARRAY [1..30] OF CHAR;
                        END;

BEGIN
DISPLAY (CONCAT('WFLString = ', WFLString.str));
IF WFLBoolean THEN
  DISPLAY ('WFLBoolean = TRUE')
ELSE
  DISPLAY ('WFLBoolean = FALSE');
WFLString.str := CONCAT('WFL string has been changed', CHR(0));
WFLBoolean := FALSE;
END.
? % End of Pascal Data Card
  RUN PASC (23, STR REFERENCE, BOOL REFERENCE);
  DISPLAY "STR = " & STR;
              % STR = "WFL STRING HAS BEEN CHANGED"
  IF BOOL THEN      % BOOL = FALSE
    DISPLAY "BOOL = TRUE"
  ELSE
    DISPLAY "BOOL = FALSE";
?END JOB.
```

The following example shows the use of WFL-provided parameters for a program written in NEWP:

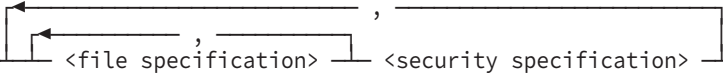
```
?BEGIN JOB EXAMPLE/NEWP;
INTEGER I := 10;
BOOLEAN BOOL := FALSE;
COMPILE NEWP/EXAMPLE WITH NEWP LIBRARY;
NEWP DATA CARD
  $ SET LEVEL 2
PROCEDURE P(WFLINTEGER, WFLSTRING, WFLBOOLEAN);
  INTEGER WFLINTEGER;
  ARRAY WFLSTRING[*];
  BOOLEAN WFLBOOLEAN;

BEGIN
  ARRAY A[0:49];
  REPLACE POINTER(A) BY "WFLINTEGER = ",
                        WFLINTEGER FOR 10 DIGITS,
                        48"00";

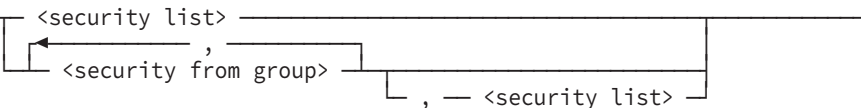
  DISPLAY (POINTER(A));
  WFLINTEGER := 20;
END.
? % END OF COMPILER DATA
  RUN NEWP/EXAMPLE (I REFERENCE, "ABCD", BOOL);
  DISPLAY "I = " & STRING(I, *); % I = 20
?END JOB.
```

SECURITY Statement

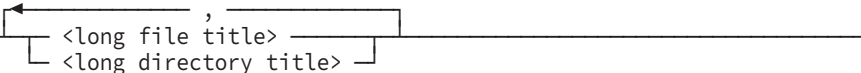
<security statement>

— SECURITY — 

<file specification>



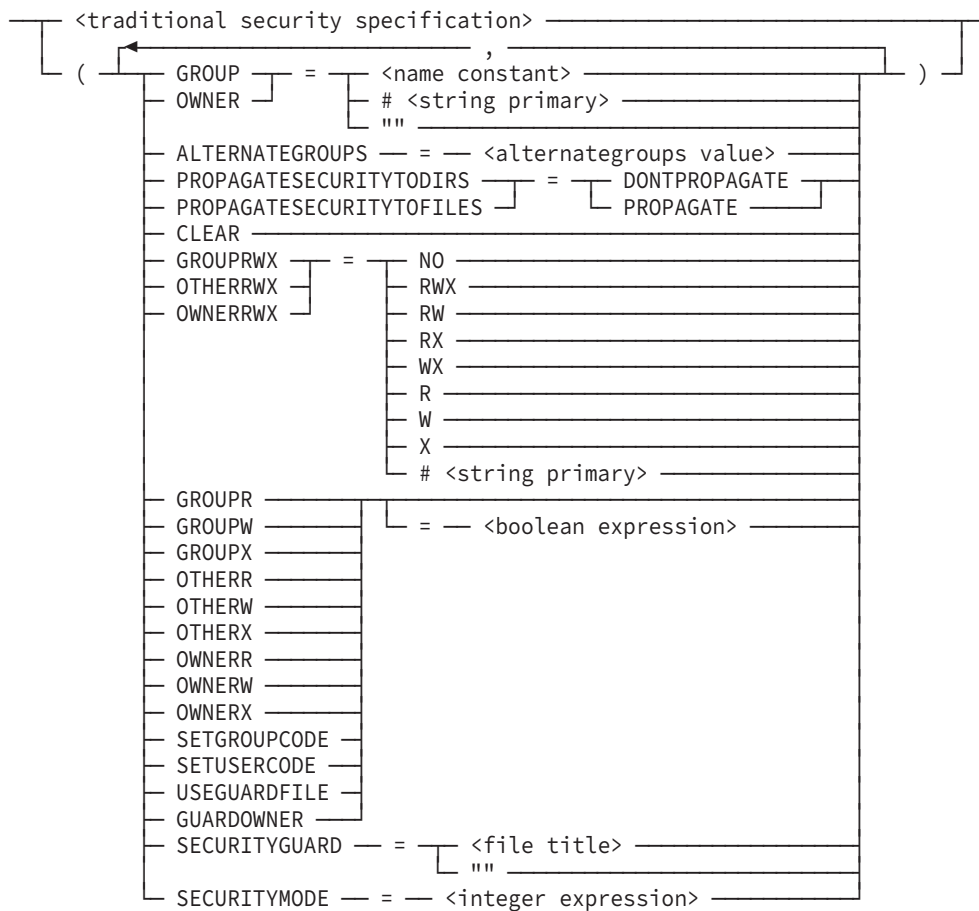
<security list>



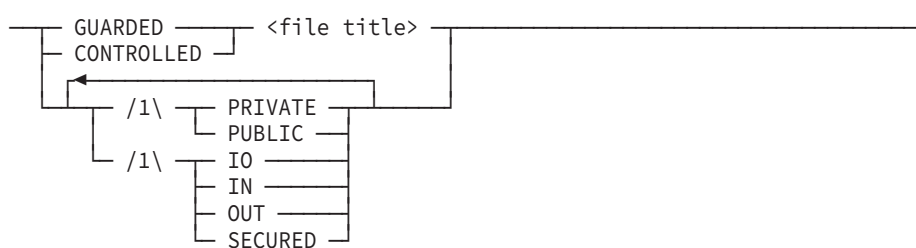
<security from group>

 FROM <family name> —

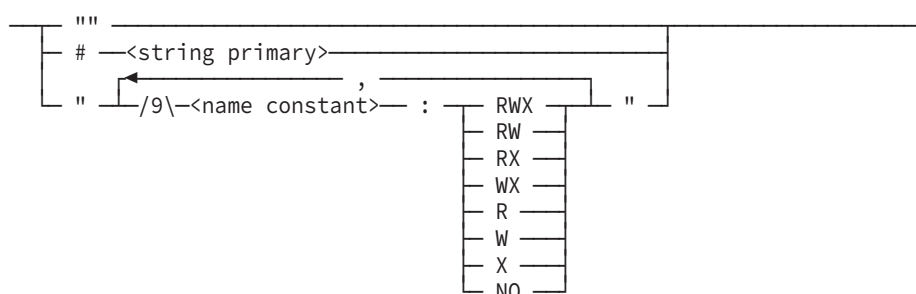
<security specification>



<traditional security specification>



<alternategroups value>



Explanation

The SECURITY statement changes the security of files on disk. In the <security from group>, the family name applies to all the file names in that <security from group>.

With the exception of the CLEAR attribute, refer to [ALTER Statement](#) earlier in this section for a description of the security attributes listed under <security specification>.

The CLEAR attribute causes all security mode flags to be reset.

Attributes are applied in the order in which they are specified. Specifying a Boolean security attribute without specifying a value implies a value of TRUE. Specifying a null string for SECURITYGUARD causes the current SECURITYGUARD value, if any, to be discarded.

Notes:

- The *OWNER*, *PROPAGATESECURITYTODIRS*, and *PROPAGATESECURITYTOFILES* attributes only apply to files in the permanent directory namespace.
- Many security attributes are interrelated; therefore changes to one attribute might affect another attribute.

For descriptions of PRIVATE, PUBLIC, GUARDED, and CONTROLLED, refer to the description of the SECURITYTYPE file attribute in the *File Attributes Programming Reference Manual*.

For descriptions of IO, IN, OUT, and SECURED, refer to the description of the SECURITYUSE file attribute in the *File Attributes Programming Reference Manual*.

Family substitution is used if the job or task has an active family specification. Only the primary family name is used. Refer to [FAMILY Assignment](#) and [Interrogating Complex Task Attributes](#).

Examples

The following examples illustrate the SECURITY statement syntax.

This statement changes the security of file AB/XY on DISK to PRIVATE for input and output:

```
SECURITY AB/XY PRIVATE IO;
```

This statement changes the security of file Z on PACK to PUBLIC for input only:

```
SECURITY Z ON PACK PUBLIC IN;
```

This statement changes the security of file A/B on MYPACK to GUARDED. File XYZ is the guard file.

```
SECURITY A/B ON MYPACK GUARDED XYZ;
```

These statements change the security of the files A/B on DISK and C/D on PACK to PUBLIC for input and output:

Statements

```
S1:="A/B";  
S2:="C/D ON PACK";  
SECURITY #S1, #S2 PUBLIC IO;
```

This statement sets the other R and W permission flags for A/B without affecting any other permission flags:

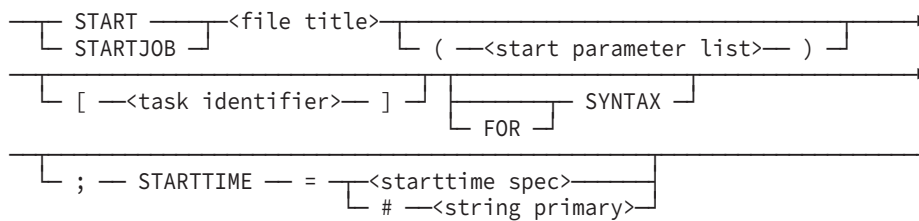
```
SECURITY A/B (OTHERRWX = RW);
```

This statement clears all permission flags, then sets the owner R and W permission flags and the other user R flag for A/B and C/D:

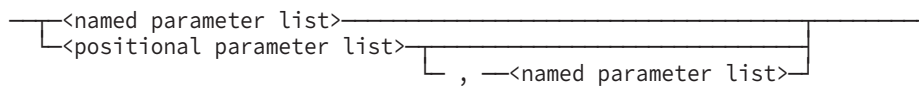
```
SECURITY A/B FROM MYPACK, C/D (CLEAR, OWNERRWX = RW, OTHERR);
```

START Statement

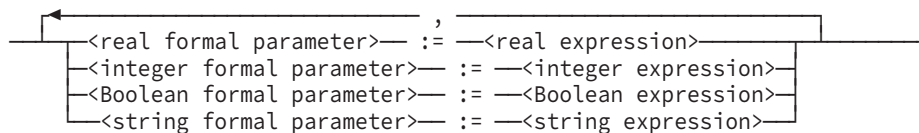
<start statement>



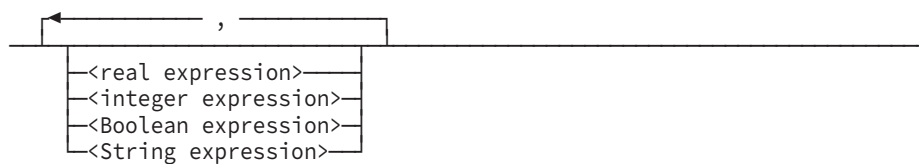
<start parameter list>



<named parameter list>



<positional parameter list>



Explanation

The START statement initiates a WFL job stored in a disk file. The file title specified is the title of the disk file containing the job that is to be initiated.

The started job inherits many of the attributes of the original job, including the values of the USERCODE and FAMILY task attributes.

The START statement first initiates a synchronous task to compile the job. The job is then inserted in a job queue and executes independently of the job that initiated it; it is not considered a task of the originating job.

When FOR SYNTAX is specified, it indicates that the job is to be compiled for syntax checking only and is not to be executed. In this case, it is not necessary to include a start parameter list in the START statement, even if the job being initiated includes a job parameter list. However, WFL does check that the parameters are of the correct type (real, Boolean, and so on) if they are supplied.

The START statement cannot include a STARTTIME specification if the job is to be started at another host through Host Services. Including a STARTTIME specification in the START statement results in an error for the started job and is detected before the job is transferred to the other host.

The STARTTIME = starttime spec form delays execution of the job until the specified time and date. However, job compilation occurs immediately, and the originating job then resumes execution at the next statement without waiting for the started job to begin.

The STARTTIME = starttime spec form in the START statement overrides any STARTTIME specification in the job attribute list of the job being started. Refer to [STARTTIME Specification](#) for the syntax of a STARTTIME specification.

The STARTTIME specification is the only job attribute that can be specified in the START statement. However, it is possible to pass job attribute values to the started job through the start parameter list if the started job was designed to accept such values.

Parameters

A start parameter list can be included in the START statement if the job being initiated includes a job parameter list. Actual parameters can be passed as positional parameters by listing them in positional order, or as named parameters by explicitly naming the corresponding formal parameters in the job parameter list. Named parameters can be given in any order.

An actual parameter need not be passed if the corresponding formal parameter specifies the keyword OPTIONAL. If an actual parameter is not passed, the default value will be assigned as follows:

Type	Default Value
Boolean	FALSE
Integer	0
Real	0
String	" " (two double quotation marks)

A default value can also be assigned by using the DEFAULT clause for the corresponding formal parameter in the job being started.

When a positional parameter list is used, optional parameters can be omitted by specifying consecutive commas (,) or by specifying fewer actual parameters than formal parameters. When fewer parameters are passed than are expected by the job, all the remaining parameters in the job parameter list must be specified as optional.

Named parameters can be used in combination with positional parameters. The positional parameters are listed first, in their normal position, followed by the named parameters. Once a named parameter is used, the rest of the parameters must be named parameters.

Examples

The following is an example of a simple START statement:

```
START (SINGA)OBJECT/FIZZCOMP ON GCPACK;
```

The following examples show various START statements that can be used to start a job with a parameter list. The job being started has the following job heading:

```
?BEGIN JOB JOB/1 (STRING CODEFILE,  
                  INTEGER JOBQUEUE OPTIONAL DEFAULT=3,  
                  BOOLEAN BIND OPTIONAL);
```

Using only positional parameters:

```
START JOB/1 ("OBJECT/P", 4, TRUE)  
START JOB/1 ("OBJECT/P")  
START JOB/1 ("OBJECT/P", , TRUE)
```

Using only named parameters:

```
START JOB/1 (BIND := TRUE, CODEFILE := "OBJECT/P")  
START JOB/1 (JOBQUEUE := 4, CODEFILE := "OBJECT/P",  
            BIND := FALSE)
```

Using both positional and named parameters:

```
START JOB/1 ("OBJECT/P", BIND := FALSE)  
START JOB/1 ("OBJECT/P", BIND := FALSE, JOBQUEUE := 0)
```

The task identifier associates a task variable with the compilation of the specified job. (This task variable is not associated with the execution of the job.) The task state expression can then be used to inquire whether the compile was successful, as in the following job excerpt:

```
START TEST/WFLJOB [T];  
IF T ISNT COMPILEDOK THEN  
ABORT "TEST/WFLJOB HAS SYNTAX ERRORS";
```

The following job compiles the job EX/1 and then runs the program OBJECT/BLAH. Execution of EX/1 begins 30 minutes after it is compiled, regardless of whether the job SR has already completed.

```
?BEGIN JOB SR;  
  START EX/1; STARTTIME = + 00:30;  
  RUN OBJECT/BLAH;  
?END JOB.
```

In the following example, the value of the <starttime spec> is supplied through a string primary. The START statement schedules the job EX/2 to begin execution after 8:00 a.m. the day after the job EXAMPLE is started.

```
?BEGIN JOB EXAMPLE;
  INTEGER I := 8;
  START EX/2;
    STARTTIME = # (STRING(I,2) & ":00 ON +1");
?END JOB.
```

The following job schedules the job EX/3 to begin execution after 2:00 a.m. each of the next five days after the job EXAMPLE is started:

```
?BEGIN JOB EXAMPLE;
  INTEGER I; I:=1;
  WHILE I LEQ 5 DO
    BEGIN
      START EX/3;
      STARTTIME = # ("2:00 ON +" & STRING(I,*));
      I:=I+1;
    END;
  ?END JOB.
```

The following job uses the first two parameters passed to it to assign values to CLASS and CHARGECODE in the job attribute list. The third parameter is used to pass a file title for use in file equation.

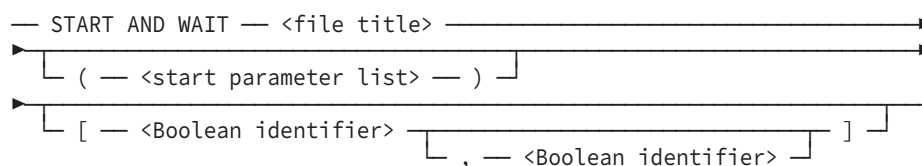
```
?BEGIN JOB EXAMPLE (INTEGER CLASSP, STRING CHARGE, STRING FILEP);
  CLASS = CLASSP;
  CHARGECODE=#CHARGE;
  COMPILE OBJECT/#FILEP WITH ALGOL LIBRARY GO;
  COMPILER FILE CARD(KIND=DISK, TITLE=#FILEP);
?END JOB.
```

The following job starts the previous job six times with different parameter values:

```
?BEGIN JOB STARTERINTEGER I;
  I:=0;
  WHILE I LEQ 5 DO
    BEGIN
      START TEST/EXAMPLE(30,"CHARGE" & STRING(I,*),
        "TEST" & STRING(I,*));
      I:=I+1; END;
  ?END JOB.
```

START AND WAIT Statement

<start and wait statement>



Explanation

The START AND WAIT statement initiates a second job as a dependent task within a WFL job and waits for the dependent task to complete. The first Boolean identifier is set to true if the WFL job compiles and the dependent task initiates successfully. The second Boolean identifier is set to true if the dependent task completes successfully.

Notes:

- The FOR SYNTAX phrase is not allowed.
- A task identifier cannot be associated with the dependent task.
- The STARTTIME specification cannot be used.
- An ABORT statement causes the dependent task and the parent job to terminate. To terminate only the dependent task, use ABORT [MYSELF].
- The START AND WAIT statement cannot be used in a PROCESS statement.
- The restrictions and behavior of the dependent task are similar to those of a bit 38 set in a CONTROLCARD ZIP WITH ARRAY call (request type 4). For more information, refer to the CONTROLCARD function in the DCALGOL Reference Manual. The restrictions include the following:
- References to MYJOB and inheritance of attributes normally inherited from a job might behave differently since they refer to the parent job.
- The dependent task performs no job rollout. Consequently, if the parent job is restarted, the dependent task is reinitiated from the beginning of the job.

STOP Statement

<stop statement>

— STOP — <string expression>

Explanation

The STOP statement is used either to terminate a job or to terminate an asynchronous subroutine. The job or asynchronous subroutine is terminated normally if it has no asynchronous subtasks being executed at the time of the STOP; otherwise, the parent and its in use dependent subtasks are terminated abnormally. The in use dependent subtasks include tasks that are SCHEDULED, ACTIVE, or STOPPED. Refer to “Task State” under [Boolean Expressions](#).

If a string expression is specified in the STOP statement, the value of the string expression (up to a maximum of 430 characters) is displayed prior to the STOP. The STOP statement accepts up to 1799 characters.

Example

The following is an example job that uses the STOP statement:

Subroutine invocation statements can be included in subroutines. The rules defining which subroutines can be invoked from a particular subroutine are defined under [Declaration Syntax](#).

Note: *A subroutine invocation statement can cause the job to enter an infinite loop if the subroutine invokes itself or a subroutine that is nested within a subroutine. In these cases, a test should normally be included to cause one of the subroutines involved in the loop to be exited if some specified condition is met.*

When a task is initiated in a subroutine, that subroutine becomes the critical block for the task. Whether the task variable associated with the task is declared locally or globally does not affect which block is the critical block for the task.

A subroutine is not exited until all asynchronous tasks initiated within that subroutine have gone to end of task. An explicit WAIT is not necessary; the subroutine automatically waits.

A subroutine invocation statement accepts string expressions of up to 1800 characters as parameters.

Examples

The following is an example of a subroutine:

```
SUBROUTINE COMPANDGO(FILE FNAME, STRING PARAM1, INTEGER PARAM2);
BEGIN
  COMPILE XYZ WITH ALGOL LIBRARY;
  COMPILER FILE CARD (TITLE=#FNAME(TITLE),KIND=DISK);
  RUN XYZ (PARAM1,PARAM2);
  REMOVE XYZ;
  PARAM2 := PARAM2+1;
END COMPANDGO;
INTEGER I;
FILE F1(TITLE=TEST1);
FILE F2(TITLE=FLOP2);
I:=1;
COMPANDGO(F1, "TEST" & STRING(I,2), I);
COMPANDGO(F2, "FLOP" & STRING(I,2), I);
```

This example illustrates that a subroutine waits for its asynchronous tasks to complete before returning. The subroutine SUB waits for the completion of OBJECT/PROG before returning to the outer block.

```
?BEGIN JOB PROCESS/TEST;
  TASK TSK;
  SUBROUTINE SUB;
  BEGIN
    PROCESS RUN OBJECT/PROG [TSK];
    DISPLAY "SUB is waiting for OBJECT/PROG";
  END SUB;
  SUB;
?END JOB.
```

This example runs the subroutine SUB1 and the program OBJECT/TEST2 asynchronously and waits for the subroutine SUB1 to finish before continuing.


```

SUBROUTINE SUB1;
  BEGIN
    COPY MONTHLY/SUMMARY AS CURRENT/SUMMARY FROM USERS(DISK);
    RUN OBJECT/ALGOL/1;
  END SUB1;
PROCESS SUB1 [TSK];
RUN OBJECT/TEST2;
WAIT (TSK IS COMPLETED);

```

In this example, the prior value of TESTSUB for the NAME task attribute (set with the task variable T) is overridden when the subroutine SUB1 is invoked by a PROCESS statement. The name of the subroutine task is changed from its default value of SUB1 to the value of TEST1.

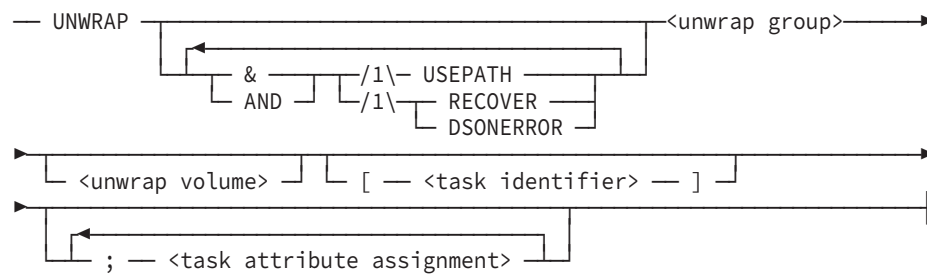
```

SUBROUTINE SUB1;
  BEGIN
    RUN TEST/1;
  END;
T(NAME=TESTSUB);
PROCESS SUB1[T];
NAME=TEST1;

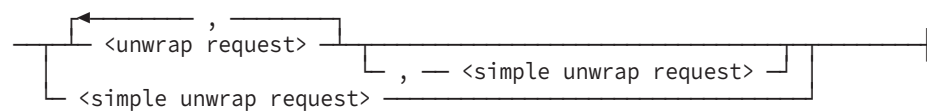
```

UNWRAP Statement

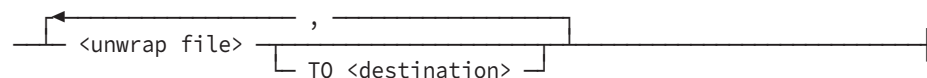
<unwrap statement>



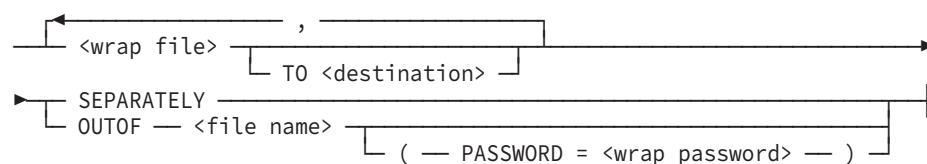
<unwrap group>



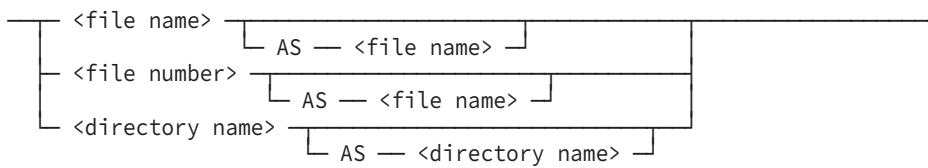
<simple unwrap request>



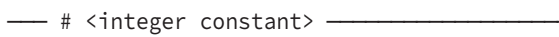
<unwrap request>



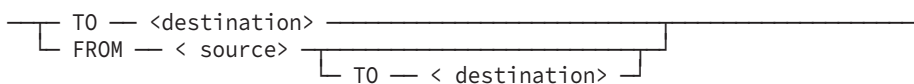
<unwrap file>



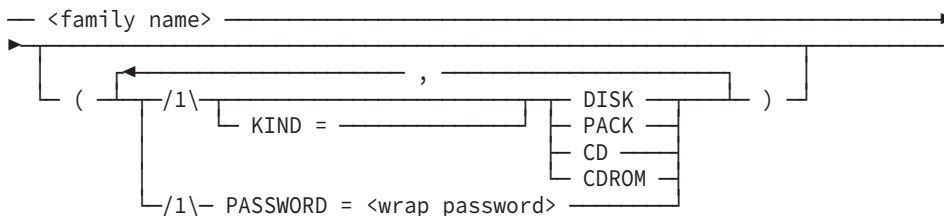
<file number>



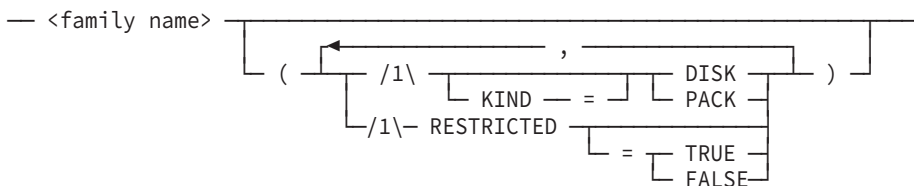
<unwrap volume>



<source>



<destination>



Explanation

The UNWRAP statement enables users to

- Unwrap files wrapped by the WRAP statement.
- Unwrap all or selected wrapped files from a container.

The unwrap process re-creates the original files from the data contained within the wrapped files or containers. By default, UNWRAP also marks system files, compilers, backup files, and code files as restricted. A security administrator on a system running with the security administrator status enabled or a privileged user can override this restriction by setting the RESTRICTED attribute to FALSE. If the file was restricted before being wrapped, it will remain restricted even if the UNWRAP statement specifies RESTRICTED=FALSE.

You can also copy unwrapped data files from a CD-ROM as part of an unwrap operation, by setting the task attribute SW1. The system then tests to see if requested files are wrapped, then it

- Unwraps wrapped files.
- Copies other files as CDATA files.

To unwrap files to multiple destinations, a destination that directly follows an unwrap file list has scope only for that list. If a destination does not follow an unwrap file list, then the list uses a destination in the unwrap volume. DISK is used if there is no destination in the unwrap volume.

A destination can be interpreted as being in a simple unwrap request or an unwrap volume. This ambiguity is resolved to maintain consistency with previous semantics, which had only one destination. If the statement contains a destination that might be in an unwrap volume and it is the only destination specified, then it is applied to all files.

To select a file by position within the directory, you can specify the <file number> in the unwrap file statement. You can use the <file number> attribute to unwrap files when the same file name appears more than once in the container directory. The file number value must be an integer greater than zero, fewer than 12 digits in length (including leading zeros).

Note: To determine the file number of a particular file in a container, use the FILEDATA NAMELIST request:

```
RUN *SYSTEM/FILEDATA ( "NAMELIST: DIR=*=  
                        CONTAINER=<container>" )
```

The DIR parameter must be "*" to select all files in the directory because the file number listed in the report refers to the position in the list rather than the position in the directory. To see the file numbers in the NAMELIST report, you can direct output a printer, or use the DEFINEOUTPUT request to set LINEWIDTH to at least 81. For example:

```
RUN *SYSTEM/FILEDATA ( "DEFINEOUTPUT:LINEWIDTH=96;  
                        NAMELIST: DIR=*=  
                        CONTAINER=<container> SCR" )
```

If the PASSWORD attribute is specified, the file or container is decrypted using the encryption algorithm specified when the file was created and an encryption key derived from the password. The PASSWORD attribute for WRAPPEDDATA files is specified on the source family name and for CONTAINERDATA files on the container file name.

If you specify the RECOVER option in the UNWRAP statement, the unwrap process attempts to recover internal wrapped files whenever it encounters a bad container whose directory is missing, incomplete, or corrupted. The RECOVER option is applicable only to container files and cannot be used with the DSONERROR option. You cannot specify a file number in conjunction with the RECOVER option because the file number cannot be determined if the directory is missing, incomplete, or corrupted.

If you specify the DSONERROR option, the unwrap process aborts whenever it encounters an error.

USEPATH causes the DATAPATH attribute to be used when resolving nonqualified file names. This includes file names without a usercode or system (*) prefix and without a family name or with a family name of DISK. Only the first element in the DATAPATH attribute string is used. Substitution is applied only when the source volume is disk. It is not used if the source is CD-ROM.

The TASKVALUE attribute can be tested to determine the success or failure of the wrap or unwrap of disk files. When the value of the TASKVALUE attribute is 0, all requests were satisfied. When the value of the TASKVALUE attribute is not 0, one or more files were not wrapped or unwrapped. A nonzero value is also returned if the wrap or unwrap operation is discontinued.

To wrap files into a container, use the INTO syntax. To unwrap files out of a container, use the OUTOF syntax. A directory name of *= wraps or unwraps all files. Only a privileged user can use this directory name. A directory name of = wraps or unwraps all files in the user's directory; if no files exist and the caller is privileged, all files in the system directory are wrapped or unwrapped.

Some wrapped files or wrapped containers might have already been digitally signed by a private key. Unwrapping such files requires you to obtain a corresponding public key and specify it through the task attribute TASKSTRING.

To see what files are wrapped in a container, you can invoke the FILEDATA utility from WFL, as follows:

```
RUN *SYSTEM/FILEDATA ( "ATTRIBUTES: DIR = *=  
                        CONTAINER =  
                        <container name>" )
```

You can also use the CANDE LFILES command for the same effect:

```
LFILES *= :CONTAINER = <container name>
```

See the *System Software Utilities Operations Reference Manual* for details.

Examples

The following examples demonstrate the UNWRAP statement:

```
UNWRAP = OUTOF CONTAINER1;  
  
UNWRAP FILEC AS NEWFILEC  
      OUTOF CONTAINER1;  
  
UNWRAP FILEF, FILEB AS NEWFILEB;
```

The following examples show how to override the RESTRICTED option:

```
UNWRAP WRAPPED/FILE AS UNWRAPPED/FILE FROM MYCD(CD)  
      TO MYPACK(RESTRICTED=FALSE) ;  
  
UNWRAP *= OUTOF MYCONTAINER TO MYPACK(RESTRICTED=FALSE) ;
```

The following example demonstrates the RECOVER option:

```
UNWRAP &RECOVER *= AS UWFILES/= OUTOF (THIEN)CONTAINER
                                FROM DISK TO MYPACK
```

The following example demonstrates the DSONERROR option:

```
UNWRAP &DSONERROR MYSTUFFS/= OUTOF MYCONTAINER FROM MYCD(CD)
                                TO MYPACK;
```

The following example shows how to unwrap a digitally signed wrapped file. The public key is obtained from the person who wrapped the file.

```
UNWRAP MY/SIGNED/WAPPED/FILE AS MY/FILE;
      TASKSTRING=<the public key's hex string>;
```

The following example shows how to unwrap a digitally signed wrapped container. The public key is obtained from the person who wrapped the container.

```
UNWRAP *= OUTOF MY/SIGNED/CONTAINER;
      TASKSTRING=<the public key's hex string>;
```

The following example shows the use of the USEPATH modifier to unwrap files from a container named *DIR/SHARED/PROJECTFILES/MYCONTAINER on PROJECTPACK.

```
UNWRAP & USEPATH *= OUTOF MYCONTAINER;
      DATAPATH = *DIR/SHARED/PROJECTFILES ON PROJECTPACK;
```

The following example shows how to unwrap files from multiple volumes in a single UNWRAP statement. Out of container C1, FILEJ is unwrapped to DISK, and FILEK and FILEL are unwrapped to MYPACK.

```
UNWRAP FILEJ TO DISK(RESTRICTED=FALSE), FILEK, FILEL OUTOF C1
                                TO MYPACK;
```

The following example shows how to specify a file number.

```
UNWRAP #3 OUTOF MYCONTAINER
```

USER Statement

<user statement>

```
— USER — = — <usercode name constant> —————→
└ / — <password name constant> ┘
```

Explanation

The USER statement changes the usercode of the job. This usercode is not retained across a halt/load. If this statement appears in a subroutine running asynchronously with the job, it changes the usercode of the subroutine only; the usercode of the job is not changed.

The usercode assignment can be used to assign a usercode to a task, as explained in [Section 5, Task Initiation](#).

The correct password must be given if a password is associated with the usercode. The password associated with the current usercode cannot be changed by the USER statement; the PASSWORD statement must be used instead.

If the job was initiated from CANDE, and this statement changes the usercode from the usercode of the originating CANDE session, then job messages cease being sent to the originating terminal. (Job messages can be output from DISPLAY statements, ACCEPT functions, or BOT, EOT, and EOJ messages.) The messages resume if a later USER statement restores the original usercode.

The USER statement does not affect the accesscode of the job, even if the current accesscode is one that is not normally permitted for the new usercode. If you want the accesscode to be affected, use usercode assignment with MYSELF or MYJOB; this will set the accesscode to a null string if it is not permitted for the new usercode.

Examples

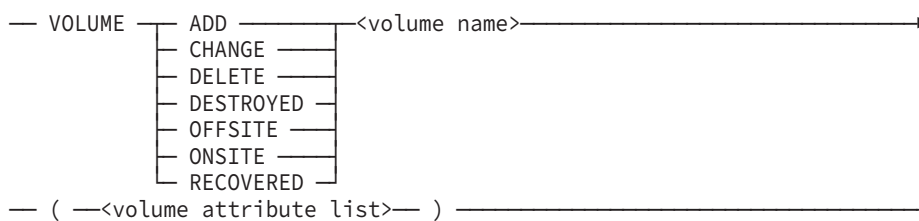
The following are examples of the USER statement:

```
USER = MYCODE
```

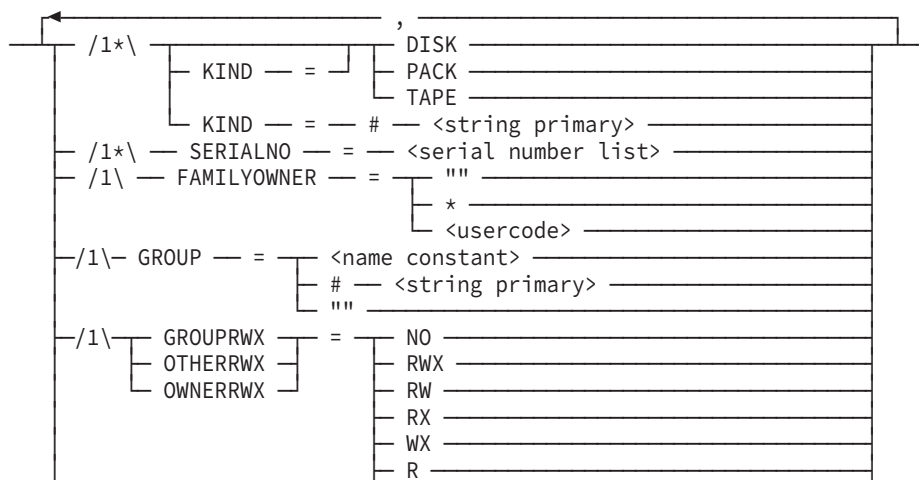
```
USER = A/B
```

VOLUME Statement

<volume statement>



<volume attribute list>



		W	
		X	
		#	<string primary>
/1\	GROUPR		
	GROUPW	=	<Boolean expression>
	GROUPX		
	OTHERR		
	OTHERW		
	OTHERX		
	OWNERR		
	OWNERW		
	OWNERX		
	SETGROUPCODE		
	SETUSERCODE		
	USEGUARDFILE		
	GUARDOWNER		
/1\	MATCHONLYSERIALNO	=	<Boolean expression>
/1\	PERMANENTLYOWNED	=	<Boolean expression>
/1\	SECURITYGUARD	=	<file title>
			""
/1\	SECURITYLABELS	=	<Boolean expression>
/1\	SECURITYMODE	=	<integer expression>
/1\	SECURITYTYPE	=	<file mnemonic primary>
/1\	SECURITYUSE	=	<file mnemonic primary>

Explanation

The VOLUME statement applies to cataloging systems and systems with the tape security subsystem activated. It is used to add, modify, or delete a family description in the cataloging volume library or the tape volume directory.

A cataloging volume library or a tape volume directory is a database containing information on volumes, with one entry per volume family. The cataloging feature provides a method for locating where backup files from disk and tape are stored.

Note: Only privileged jobs and jobs started from an ODT can issue a VOLUME command. (The VOLUME CHANGE command is the only exception to this rule.)

The volume name and the file attributes KIND and SERIALNO are required to identify the family. If the volume name is DISK or PACK, the KIND attribute defaults to DISK and can be omitted. When the volume name includes a <string primary>, embedded periods will terminate the volume name unless the entire name is enclosed in quotes.

For tapes with two-level names (for example, LMTAPE/FILE000), the tape name is the first name.

Options

The following options are available in the VOLUME statement.

VOLUME ADD

Creates an entry in the cataloging volume library and/or the tape volume directory. When an entry for a tape volume is created, the system automatically tracks the changing status of the tape volume. It is not necessary to issue a new VOLUME ADD each time the status of the tape volume changes. The order of the serial numbers in the statement coincides with the order of the volumes in the family (reel number).

The volume family does not need to be online when the VOLUME ADD is executed. However, if the volume family is online, the information in the WFL statement is supplemented by other information stored in the entry, such as the volume creation date.

Serial numbers added to the volume library must be unique for tapes and for disks. The VOLUME ADD request is rejected if an entry in the volume library contains a serial number for a volume (disk or tape) that matches a serial number in the list to be added in the VOLUME ADD request. Therefore, the entry with the conflicting serial number must be deleted from the volume library before the VOLUME ADD request can be executed.

When SCRATCH is specified as the volume name in a VOLUME ADD statement for a tape, it indicates that the volume is a scratch tape (a purged tape) or that SCRATCH is the name of the tape volume.

VOLUME CHANGE

Changes the tape volume directory entry (but not the cataloging volume library entry) for a tape family. Only the attributes SECURITYGUARD, SECURITYTYPE, SECURITYUSE, SECURITYMODE, and the sub attributes of SECURITYMODE can be changed.

The VOLUME CHANGE statement for tape files is similar to the SECURITY statement for disk files.

VOLUME CHANGE can be used by privileged jobs and for non-privileged jobs with tapes owned by the job.

Note: If the VOLUME CHANGE statement is used to alter the SECURITYMODE sub attributes of a tape volume (for example, OWNERRWX, GROUPEWX, and so on), then all of the desired attributes must be specified in the CHANGE request similar to what is required on a VOLUME ADD statement. All SECURITYMODE sub attributes that are omitted from the VOLUME CHANGE request revert back to their default values.

VOLUME DELETE

Deletes entries from the cataloging volume library and/or the tape volume directory. The order that serial numbers appear in the statement does not need to be the order of volumes in the family.

The family must not be in use when the VOLUME DELETE is executed.

VOLUME DESTROYED

Changes the entry for one or more tape volumes in the cataloging volume library and/or the tape volume directory, to show that the volumes are no longer usable. The entry for a volume remains in the cataloging volume library and/or the tape volume directory and can be subsequently deleted.

The archive system in the MCP cannot request tape volumes that are marked as destroyed, unless there are no other backup copies of the requested file.

VOLUME DESTROYED does not affect cataloging information. To cancel the destroyed condition, use the VOLUME RECOVERED statement.

VOLUME OFFSITE

Marks the entry for one or more tape volumes in the cataloging volume library and/or the tape volume directory. This will show that the tape, or tapes, are offsite.

The file search mechanism in the MCP will not request tape volumes that are marked as offsite, unless there are no other backup copies of the requested file on tapes that are not marked as offsite or destroyed.

VOLUME ONSITE

Marks the entry for one or more tape volumes in the volume library and/or volume directory that are no longer offsite.

Note: *The default value for a volume when a VOLUME ADD is completed is ONSITE.*

The onsite/offsite status is displayed by the PV and TV system commands and the onsite/offsite status is reported by the SYSTEM/LISTVOLUME and SYSTEM/LISTVOLUMELIB utility programs.

VOLUME RECOVERED

Changes the entries for one or more tape volumes in the catalog volume library or the tape volume directory or both to show that the volumes are no longer considered destroyed.

File Attributes

The following file attributes can be specified in the VOLUME statement:

GROUP

Specifies a group whose members can access the file in the manner defined by the GROUPRWX attribute. Any process executing with a task GROUPCODE or SUPPLEMENTARYGROUPS that matches the GROUP attribute of the file, and that also does not match as the owner of the file, is granted the access permissions defined by the GROUPRWX attribute. If the GROUP attribute is not set, then group access is not granted to any process attempting to access the file.

GROUPR

When set to TRUE, grants group members read-access to the file.

GROUPW

When set to TRUE, grants group members write-access to the file.

GROUPX

Has no effect for tape volumes.

GROUPRWX

Specifies the manner in which members of the group matching the group attribute of the file are permitted to access the physical file.

GUARDOWNER

Used in conjunction with the USEGUARDFILE attribute to cause the guard file to define access permissions for the owner of the file.

Note: The *GUARDOWNER* attribute has no effect if the *USEGUARDFILE* attribute is reset.

OTHERR

When set to TRUE, grants other users (excluding the owner and members of the group) read-access to the file.

OTHERW

When set to TRUE, grants other users (excluding the owner and members of the group) write-access to the file.

OTHERX

Has no effect for tape volumes.

OTHERRWX

Specifies the manner in which all other users (excluding the owner and members of the group) are permitted to access the physical file.

OWNERR

When set to TRUE, grants the owner readaccess to the file.

OWNERW

When set to TRUE, grants the owner writeaccess to the file.

OWNERX

Has no effect for tape volumes.

OWNERRWX

Specifies the manner in which the owner of the file is permitted to access the physical file.

SECURITYLABELS

When set to true, the system stores the security attributes for files in the tape labels on the tape volume.

A VOLUME statement can be used to set the SECURITYLABELS attribute to TRUE or the PERMANENTLYOWNED attribute to TRUE, but not both. The system will reject a VOLUME statement that specifies both SECURITYLABELS=TRUE and PERMANENTLYOWNED = TRUE.

Note: Only one serial number can be specified in the serial number list of the SERIALNO attribute when you use either the SECURITYLABELS or the MATCHONLYSERIALNO attribute.

SECURITYMODE

Specifies the manner in which users are permitted to access the physical file, including the owner of the file.

SERIALNO

The SERIALNO attribute for disks and packs works in a special manner.

The VOLUME ADD statement for a disk or pack family should specify the serial number of the disk with the family index number 1. For multipack families, the serial number of the disk with family index 1 should be specified even if the family has more than one base pack. The serial number of the disk with family index 1 should be specified even if that member no longer has a directory or even if that particular disk has disappeared.

Note: Do not use more than one serial number in a list for a disk and pack family specification.

SETGROUPCODE

Has no effect for tape volumes.

SETUSERCODE

Has no effect for tape volumes.

USEGUARDFILE

When set to TRUE, a guard file in addition to the SECURITYMODE attribute controls access to the physical file. In order for the guard file to control access to the file completely, all of the file's access permission flags OWNERRWX, GROUPEWX, and OTHERWX should be set to TRUE.

Tape Volume Security

The tape volume security feature is available through the Security Accountability Facility software. When the tape security subsystem is activated, the system applies security constraints and checks on tape files. The two main features of the tape security subsystem include: tape volume ownership and tape file security.

For details, refer to the *MCP Security Overview and Implementation Guide*.

VOLUME ADD Statement with Tape Security Subsystem

The following file attributes can be specified for a VOLUME ADD statement when the tape security subsystem is activated.

FAMILYOWNER

Indicates the owner of the tape volume. If FAMILYOWNER is not specified, or is specified as " ", the owner will be the usercode of the task issuing the VOLUME ADD. If FAMILYOWNER is specified as *, the tape will be owned by the jobs and tasks running without a usercode.

GROUP

Specifies a group whose members can access the file in the manner defined by the GROUPEWX attribute. Any process executing with a task GROUPCODE or SUPPLEMENTARYGROUPS that matches the GROUP attribute of the file, and that also does not match as the owner of the file, is granted the access permissions defined by the GROUPEWX attribute. If the GROUP attribute is not set, then group access is not granted to any process attempting to access the file.

GROUPE

When set to TRUE, grants group members read-access to the file.

GROUPEW

When set to TRUE, grants group members write-access to the file.

GROUPEX

Has no effect for tape volumes.

GROUPRWX

Specifies the manner in which members of the group matching the group attribute of the file are permitted to access the physical file.

GUARDOWNER

When used in conjunction with the USEGUARDFILE attribute, causes the guard file to define access permissions for the owner of the file.

Note: *The GUARDOWNER attribute has no effect if the USEGUARDFILE attribute is reset.*

MATCHONLYSERIALNO

When set to TRUE, instructs the system not to check the tape names and creation dates of an entry in the tape volume directory. The volume name is ignored.

OTHERR

When set to TRUE, grants other users (excluding the owner and members of the group) read-access to the file.

OTHERW

When set to TRUE, grants other users (excluding the owner and members of the group) write-access to the file.

OTHERX

Has no effect for tape volumes.

OTHERRWX

Specifies the manner in which all other users (excluding the owner and members of the group) are permitted to access the physical file.

OWNERR

When set to TRUE, grants the owner readaccess to the file.

OWNERW

When set to TRUE, grants the owner writeaccess to the file.

OWNERX

Has no effect for tape volumes.

OWNERRWX

Specifies the manner in which the owner of the file is permitted to access the physical file.

PERMANENTLYOWNED

When set to TRUE, only tape files with a matching FAMILYOWNER can be written on the tape. If it is set to FALSE, when tape files with a different owner are written on the tape, the FAMILYOWNER of the tape volume is automatically changed.

SECURITYLABELS

When set to TRUE, maintains security attributes SECURITYTYPE, SECURITYUSE, SECURITYGUARD, and FAMILYOWNER in the tape volume label. With this information on the tape itself, a tape can be transferred easily among hosts in a multihost environment. At volume creation, the system determines the security attribute values of the first file written to the volume. The tape system then writes those values to the tape label and to the volume directory.

Each volume of a multivolume file must have a SECURITYLABELS value that matches the SECURITYLABEL value of the first volume. When a SECURITYLABELS=TRUE comes online, the system reads the security attribute values from the tape label and updates the volume directory accordingly.

Note: A VOLUME statement can set the SECURITYLABELS attribute to TRUE or the PERMANENTLYOWNED attribute to TRUE but not both. The system will reject a VOLUME statement that specifies both SECURITYLABELS=TRUE and PERMANENTLYOWNED=TRUE.

SECURITYMODE

Specifies the manner in which users are permitted to access the physical file, including the owner of the file.

SECURITYTYPE

Provides access control over users, other than the owner of a file, to a physical file. This attribute can have a value of PRIVATE (default), PUBLIC, GUARDED, or CONTROLLED. PRIVATE files can be accessed or overwritten only by their owners and privileged users. PUBLIC files can be accessed by tasks with any usercode, as limited by the setting of the SECURITYUSE attribute. The security of GUARDED and CONTROLLED tape files is determined by the guard file referenced by the SECURITYGUARD attribute.

SECURITYGUARD

Identifies the guard file to be used if the SECURITYTYPE attribute is set to GUARDED or CONTROLLED.

SECURITYUSE

Has a value of IO (default), IN, or OUT. When a PUBLIC file is accessed by a task with a usercode that differs from the FAMILYOWNER, the SECURITYUSE attribute can be used for the following actions based on its value:

- A value of IO permits reading, writing, overwriting, and purging.
- A value of IN permits reading but not writing, overwriting, or purging.
- A value of OUT permits writing, overwriting, or purging, but not reading.

SETGROUPCODE

Has no effect for tape volumes.

SETUSERCODE

Has no effect for tape volumes.

USEGUARDFILE

When set to TRUE, then a guard file in addition to the SECURITYMODE attribute controls access to the physical file. In order for the guard file to control access to the file completely, all of the file's access permission flags OWNERRWX, GROUPRWX, and OTHERRWX should be set to TRUE.

Note: Many security attributes are interrelated, therefore changes to one attribute might affect another attribute.

For details about these file attributes, refer to the *File Attributes Programming Reference Manual*.

Examples

The following examples illustrate use of the VOLUME statement syntax.

This statement establishes a two-volume tape family with the volume name QFILES; both volumes are permanently owned by the usercode AEDEPT:

```
VOLUME ADD QFILES (TAPE, SERIALNO = (111111, 222222),
                    FAMILYOWNER = AEDEPT, PERMANENTLYOWNED)
```

This statement establishes a volume directory entry for a tape volume named QFILES under the usercode AEDEPT:

```
VOLUME ADD QFILES (TAPE, SERIALNO = (111111),
                    FAMILYOWNER = AEDEPT,
                    SECURITYTYPE = GUARDED,
                    SECURITYGUARD = NEWGUARD)
```

This statement changes the status of the entry for the tape LPROGS so that the tape is GUARDED with GUARD FILE A/B:

This statement deletes the entry for PACK from the cataloging volume library:

This statement changes the status of the entry for tape ABC to damaged:

The following statement establishes a volume directory entry for a scratch tape volume under the usercode AEDEPT. The security attribute values are stored in the tape volume label when a file is written to the tape.

WAIT Statement

```

— WAIT —
└ ( ┌ <string expression> — , — <wait specification> ┐ ) ┘
    └ <string expression> ┘
      └ <wait specification> ┘

```

OK	
<real expression>	
<task identifier>	
<task state>	
<simple task relation>	
<task mnemonic comparision>	
<task identifier> (—<Boolean task attribute>—)	
FILE —<file title>— IS — RESIDENT —	

$$\begin{array}{l} \text{---}\langle\text{task identifier}\rangle\text{---} \left(\begin{array}{l} \text{---}\langle\text{integer task attribute}\rangle\text{---} \\ \text{---}\langle\text{real task attribute}\rangle\text{---} \end{array} \right) \text{---} \\ \text{---}\langle\text{real relation}\rangle\text{---}\langle\text{real expression}\rangle\text{---} \end{array}$$

The WAIT statement enables the job stack to suspend execution until specified conditions are met. The following t options are available in the WAIT statement.

The simple form of the WAIT statement causes the job stack to wait for its own exception event. If a string expression is used, it is displayed (up to a maximum of 430 characters) on the ODT. The WAIT statement accepts up to 1799 characters.

WAIT (OK)

The job stack is suspended until an OK is entered from the ODT.

WAIT (<real expression>)

The job stack waits the specified number of seconds and then resumes execution.

WAIT (<task identifier>)

The job stack waits until the task is completed. This statement has no effect on the task when the value of the STATUS attribute for the task is BADINITIATE, TERMINATED, or NEVERUSED.

WAIT (<task state>)

The job stack waits until the task is completed or achieves the given state. This statement has no effect on the task when the value of the STATUS attribute for the task is BADINITIATE, TERMINATED, or NEVERUSED.

WAIT (<simple task relation>)

The job stack waits until either the task is completed or the task attribute satisfies the relation. The job stack waits for its own exception event. The job does not resume execution until this event is caused and one of the previously mentioned conditions is met.

The exception event of the job can be caused by the following:

- A task that changes task state
- Entering the HI (Cause Exception Event) system command at the ODT
- Programmatic cause from the task

The real expression in a simple task relation cannot contain another reference to a task identifier.

WAIT (<task mnemonic comparison>)

The job stack waits until the comparison is satisfied.

WAIT (<task identifier> (<Boolean task attribute>))

The job stack waits either until the task is completed or until the Boolean task attribute is TRUE.

WAIT (FILE <file title> IS RESIDENT)

The job stack waits until the file is resident. If the file is absent, a "NO FILE" condition displays.

The exception event of the job need not be caused for this form of the WAIT statement to be completed.

Note: *There is a maximum WAIT time limit of 164925 seconds (approximately 45.8 hours). If the time specified in a WAIT statement is longer than 164925 seconds, the WAIT time will be truncated to 45.8 hours.*

Examples

The following examples illustrate the WAIT statement.

The simple form of the WAIT statement causes the job task to wait for its own exception event to be caused. The WFL job can then wait for several conditions because each condition can cause an exception event. For example:

```
DO
  WAIT    % wait for exception event
UNTIL ( T1 IS COMPLETED OR
        T2 IS COMPLETED );
```

To test for several conditions without relying on an exception event, you must use a WAIT (<real expression>) to wait a specified number of seconds before testing the conditions. For example:

```
DO
  WAIT(60)    % check condition every 60 seconds
UNTIL ( T1(ACCUMPROCTIME) GTR 1000 OR
        FILE A IS RESIDENT );
```

When a WAIT statement includes a <task state> option, the job stack waits until the task is completed or achieves the given state. This statement has no effect on a task when the value of the STATUS attribute for the task is BADINITIATE, TERMINATED, or NEVERUSED. Therefore, the following statement does not wait if the STATUS attribute is BADINITIATE, TERMINATED, or NEVERUSED. Otherwise, the statement waits until the STATUS attribute for the task is completed:

```
WAIT (T1 IS COMPLETED)
```

When a Boolean expression contains a <task state> with an IS, then the condition is TRUE only if the task variable is in the specified state. If the value of the STATUS attribute for the task is BADINITIATE, TERMINATED, or NEVERUSED, then the <task state> Boolean expression with IS always yields a FALSE value. Therefore, the following statement always waits until the STATUS attribute for the task is completed:

```
DO WAIT UNTIL (T1 IS COMPLETED)
```

The following are examples of WAIT statements:

```
WAIT (OK)
WAIT ("NEXT EVENT NOT HAPPENED", OK)
WAIT (30)
WAIT (TSK)
WAIT (TSK IS ACTIVE)
```

```

WAIT (TSK (TASKVALUE) EQL 9)
WAIT (TSK (STATUS) IS TERMINATED)
WAIT (TSK (SW1))
WAIT (FILE A IS RESIDENT)

```

If a WFL job processed an ALGOL task and is waiting, the exception event can be caused in the ALGOL task. For example, the following WFL statements process an ALGOL task and then wait for the TASKVALUE of that task to reach a certain value:

```

PROCESS RUN X[T];
WAIT (T(TASKVALUE) = 10);

```

The following ALGOL statements are from the program X, which is referenced in the previous example. These statements set the TASKVALUE to 10 and then cause the exception event of the WFL job. By causing the job exception event, the task enables the job to detect the fact that the TASKVALUE has changed.

```

MYSELF.TASKVALUE := 10;
CAUSE(MYJOB.EXCEPTIONEVENT);

```

WHILE Statement

<while statement>

```

— WHILE — <Boolean expression> — DO — <statement> —————|

```

Explanation

The WHILE statement enables you to perform a statement while a condition is TRUE.

The Boolean expression is evaluated; if the result is TRUE, the statement following the DO is executed. This sequence of events continues until the Boolean expression becomes FALSE or the statement following the DO transfers control outside itself.

Note: *If the Boolean expression never evaluates to FALSE, and no GO statement is used to transfer control outside, the WHILE statement can result in an infinite loop.*

If the same PROCESS statement is executed repeatedly by a WHILE statement, a run-time error might result. This can occur because the asynchronous task initiated by an earlier pass through the WHILE statement might still be running, and a given task variable can only be used by one task at a time.

Example

The following is an example of the WHILE statement:

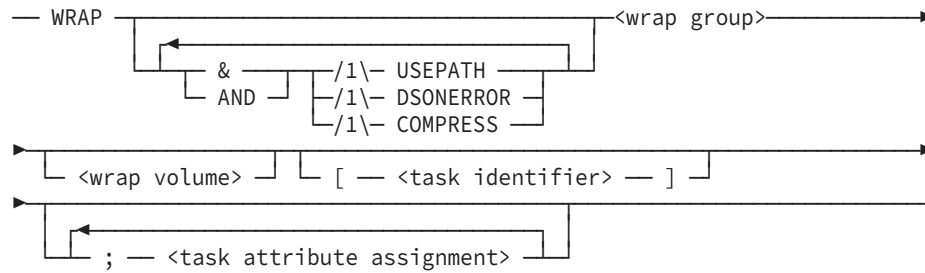
```

WHILE T(TASKVALUE) NEQ 1 DO
  BEGIN
    INITIALIZE(T);
    RUN X[T];
  END;

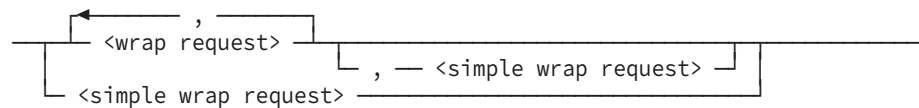
```

WRAP Statement

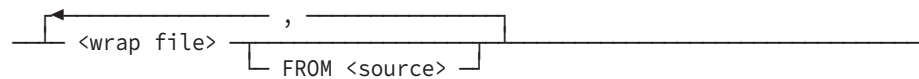
<wrap statement>



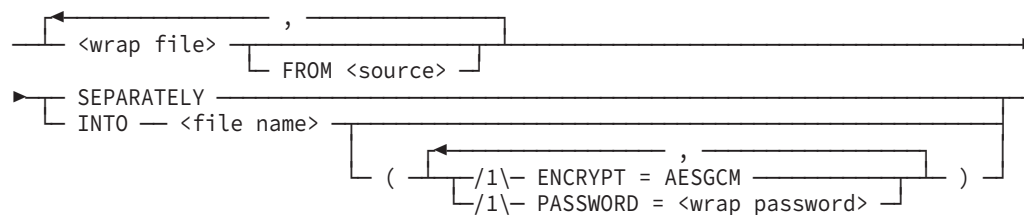
<wrap group>



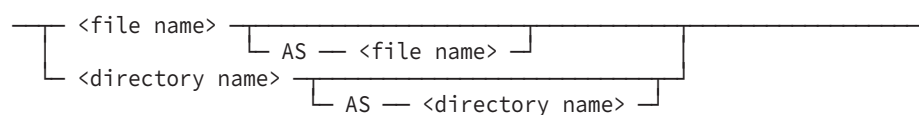
<simple wrap request>



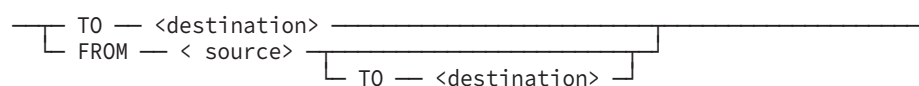
<wrap request>



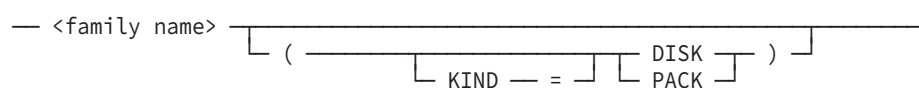
<wrap file>



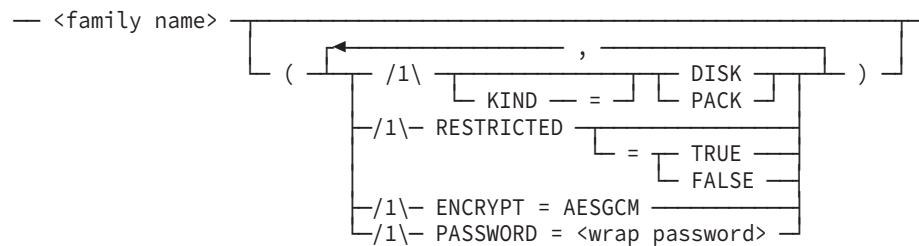
<wrap volume>



<source>



<destination>



Explanation

The WRAP statement enables users to

- Wrap disk files into new files with a FILEKIND value of WRAPPEDDATA.
- Wrap multiple disk files into a single file with a FILEKIND value of CONTAINERDATA.
- Create wrapped files and wrapped containers with digital signatures.
- Create compressed files and containers.
- Create encrypted files and containers.

A wrapped file created using the WRAP statement contains the file data and the original disk file headers. A wrapped container is made up of one or more wrapped files. Both wrapped files and containers can be transported across an open network and then be recreated on another ClearPath MCP system using the UNWRAP statement.

To wrap files from multiple sources, a source that directly follows a wrap file list has scope only for that list. If a destination does not follow a wrap file list, then the list uses a destination in the wrap volume. Family DISK is used if there is no source in the wrap volume.

A source can be interpreted as being in a simple wrap request or a wrap volume. This ambiguity is resolved to maintain consistency with previous semantics, which had only one source. If the statement contains a source that might be in a wrap volume and it is the only source specified, then it is applied to all the files.

The COMPRESS option causes the wrapped files and containers to be created compressed. The UNWRAP statement automatically decompresses wrapped files. The COMPRESSION product key (nnn-COMPRESSION-CPR) is required if the wrapped container file is not for Unisys support use.

If the PASSWORD attribute is specified, the file or container is encrypted using an encryption key derived from the password. The encryption algorithm can be specified using the ENCRYPT attribute (only AESGCM is currently supported). If the ENCRYPT attribute is not specified, the default encryption algorithm (currently AESGCM) is used. The encryption attributes for WRAPPEDDATA files are specified on the destination family name and for CONTAINERDATA files on the container file name.

If the DSONERROR option is specified, the wrap process aborts whenever it encounters an error.

USEPATH causes the DATAPATH attribute to be used when resolving nonqualified file names. This includes file names without a usercode or system (*) prefix and without a family name or with a family name of DISK. Only the first element in the DATAPATH attribute string is used. Substitution is applied only when the source volume is disk. It is not used if the source is CD-ROM.

The TASKVALUE attribute can be tested to determine the success or failure of the wrap or unwrap of disk files. When the value of the TASKVALUE attribute is 0, all requests were satisfied. When the value of the TASKVALUE attribute is not 0, one or more files were not wrapped or unwrapped. A nonzero value is also returned if the wrap or unwrap operation is discontinued.

To wrap files into a container, use the INTO syntax. To unwrap files out of a container, use the OUTOF syntax. For details on using the "=" and "*"=" syntax, refer to the COPY or ADD statement earlier in this section.

Although WRAP statement syntax permits the specification of the RESTRICTED attribute, primarily for consistency with the UNWRAP statement, the attribute has no meaning upon a wrap operation and is ignored. Refer to the UNWRAP statement earlier in this section for information on how to use the RESTRICTED attribute.

To wrap a SYSTEMDIRECTORY file, you must specify its file name in full. For example, the statement "WRAP *SYSTEMDIRECTORY/= AS SYSDIR/= FROM . . ." does not wrap a system directory file but the following statement does:

```
WRAP *SYSTEMDIRECTORY/001 AS SYSDIR/1 FROM. . .
```

When the MCP wraps a SYSTEMDIRECTORY file, it erases the special system file mark from the internal attributes of the wrapped file so that when the file is unwrapped, the file can be removed with the WFL REMOVE statement or be renamed with the WFL CHANGE statement.

To wrap a JOBDESC file, which is a file with a FILEKIND attribute value of JOBDESCFILE, you must specify the file name in full. For example, the statement "WRAP *= INTO . . ." does not wrap a job description file, but the following statement does:

```
WRAP *JOBDESC INTO . . .
```

When the MCP wraps a JOBDESC file, it changes the FILEKIND attribute of the wrapped file to DATA.

MCP_FILEWRAPPER takes special action to ensure that the copies of the various files making up a KEYEDIOII set are consistent with each other. In general, when you wrap a KEYEDIOII file, it is advisable to wrap all the related files in the same wrap statement. If any of the files in the KEYEDIOII set are opened for update, then none of the files are wrapped.

Digital Signatures

The WRAP statement enables also you to create a wrapped file or wrapped container with an embedded digital signature. A digital signature is a hash pattern that is created by applying an industry standard signature algorithm to the file along with a private key. This hash pattern travels with the file across a network and is used with a public key to ensure that the file has not been compromised during the transfer process.

To wrap files with a digital signature, you must

- Obtain a public and private key pair for the software level that the files are to be wrapped against. For information on how to generate digital signature public and private keys, refer to the procedure MCP_GENERATEDSAKEYS in the *MCP System Interfaces Programming Reference Manual*.
- Specify the hex string value of the private key through the task attribute TASKSTRING. If the files are to be wrapped against a software level other than the current level, specify that software level through the task attribute TARGET.

Note: A digital signature key pair generated for one software level cannot be used to wrap files against another software level.

- Pass on the corresponding public key to the party responsible for unwrapping the digitally signed wrapped files or wrapped containers.

Digital signatures are optional for most files. However, when wrapping a file whose FILEKIND attribute value is KEYSFILE, the system requires the file to be digitally signed. This restriction ensures that sensitive data contained in the keyfile is not being modified in any way and guarantees the authenticity of the originator of the file.

Examples

The following example shows how to create the wrapped file WRAPPED/FILEA from FILEA and WRAPPED/FILEB from FILEB:

```
WRAP  FILEA AS WRAPPED/FILEA,  
      FILEB AS WRAPPED/FILEB;
```

The following example shows how to create the wrapped file FILED by overwriting the original file with the same title:

```
WRAP  FILED;
```

The following example shows how to create the container CONTAINER1 that includes FILEE and FILEF:

```
WRAP  FILEE, FILEF INTO CONTAINER;
```

The following example shows how to create the container CONTAINER2 that includes the renamed files FILEG2 and FILEH2:

```
WRAP  FILEG AS FILEG2,  
      FILEH AS FILEH2  
      INTO CONTAINER2;
```

The following example shows how to create both wrapped files and containers in one single WRAP statement:

```
WRAP  FILEI AS WRAPPED/FILEI,  
      FILEJ SEPARATELY,  
      FILEK, FILEM INTO CONTAINER3,  
      FILEN AS FILEN2, FILEO INTO CONTAINER4;
```

The following example demonstrates the DSONERROR option. If any of the specified files is missing, the container MYCONTAINER is not created.

```
WRAP &DSONERROR MYFILE/A, MYFILE/B, MYFILE/C  
      INTO MYCONTAINER FROM DISK TO PACK;
```

The following example shows how to create a digitally signed wrapped file against SSR level 59.1. The private key has already been generated based on SSR level 59.1.

```
WRAP MY/FILE AS MY/SIGNED/WRAPPED/FILE;  
      TARGET=591; TASKSTRING=<59.1 private key's hex string>;
```

The following example shows how to create a digitally signed wrapped container against the current software level. The private key has already been generated based on the current software level.

```
WRAP MY/DIRECTORY/= INTO MY/SIGNED/CONTAINER;  
      TASKSTRING=<current level private key's hex string>;
```

The following example shows the use of the USEPATH modifier to wrap the files *DIR/SHARED/PROJECTFILES/MYFILE from PROJECTPACK and *YOURFILE from DISK into a container named *DIR/SHARED/PROJECTFILES/MYCONTAINER on PROJECTPACK.

```
WRAP & USEPATH MYFILE, *YOURFILE INTO MYCONTAINER;  
      DATAPATH = *DIR/SHARED/PROJECTFILES ON PROJECTPACK;
```

The following examples show how to wrap files from multiple volumes in a single WRAP statement. The container C1 is created by wrapping FILEJ from DISK, and FILEK and FILEL from MYPACK.

```
WRAP FILEJ FROM DISK, FILEK, FILEL FROM MYPACK INTO C1;  
WRAP FILEJ FROM DISK, FILEK, FILEL INTO C1 FROM MYPACK;
```


Section 7

Expressions

Overview

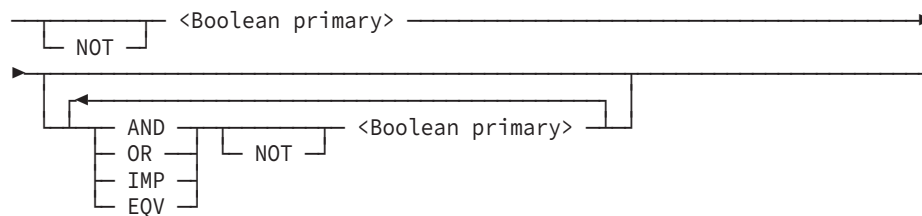
An expression is a combination of basic elements which, on evaluation, yields a result of a given type (for example, Boolean, real, integer, or string). All variables must be explicitly declared before they are used in an expression.

The expressions in this section are grouped into Boolean, integer, real, string, and mnemonic expressions, according to the type of result they return.

Expressions that enable only constant values (no variables) are described under [Constant Expressions](#) later in this section.

Boolean Expressions

<Boolean expression>



Explanation

The relational operators in a Boolean expression have the following meanings.

Operator	Meaning
NOT	Returns the opposite of the truth value of the Boolean primary it precedes.
AND	Returns TRUE if both Boolean primaries are TRUE.
OR	Returns TRUE if either or both of the Boolean primaries are TRUE.

Expressions

Operator	Meaning
IMP	Returns TRUE unless the first Boolean primary is TRUE and the second one is FALSE.
EQV	Returns TRUE if both of the Boolean primaries have the same truth value.

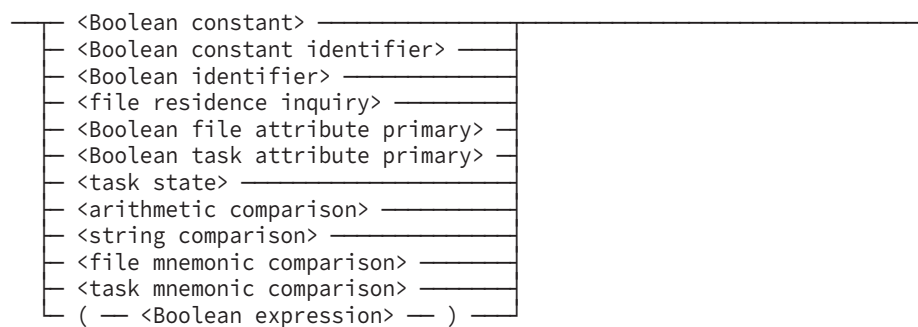
The order of priority (highest first) for the execution of Boolean operations is as follows:

- Boolean primary
- NOT
- AND
- OR
- IMP
- EQV

All Boolean primaries are evaluated first; then the NOT operators are applied to the Boolean primaries that follow them. The other operations are performed in decreasing order of priority. If two operations are of the same priority, the left operation is performed first. If a Boolean expression is enclosed in parentheses, it becomes a Boolean primary.

Boolean Primary

<Boolean primary>

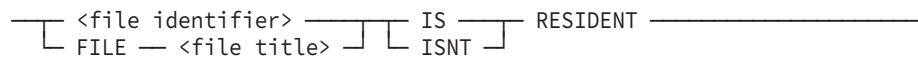


Explanation

Syntax for Boolean constants, Boolean constant identifiers, and Boolean identifiers is provided in [Section 8, Basic Constructs](#). The other types of Boolean primaries are defined in the following topics.

File Residence Inquiry

<file residence inquiry>



Explanation

The file inquiry checks to see whether a file is available. If the file was declared in the job, the file identifier is used in the inquiry.

Examples

The following example illustrates a file residence inquiry expression:

```
IF INFILE1 IS RESIDENT THEN SUB1;
```

The residence of files that are not declared in the job can be checked, using the FILE <file title> form. For example:

```
FILE (JACOB)OBJECT/LION IS RESIDENT
```

The file residence inquiry can also be used to check whether or not a tape is available. However, it cannot be used to check on the existence of individual files on a tape.

The following example is invalid:

```
?BEGIN JOB;
FILE F (KIND=TAPE, SERIALNO="PAYROL", TITLE=PAYROL/FILE000);
IF F IS RESIDENT THEN
  DISPLAY"TAPE F IS AVAILABLE"
ELSE
  ISPLAY"TAPE F IS NOT AVAILABLE";
?END JOB.
```

The file residence inquiry can even be used to check on the existence of a global data specification, because a data specification is read as if it were an input file. For example:

```
?BEGIN JOB;
FILE G (KIND=READER, TITLE=IN/COUNT);
DATA IN/COUNT
  1 2 3
?
IF G IS RESIDENT THEN
  DISPLAY"GLOBAL DATA IN/COUNT IS PRESENT"
ELSE
  DISPLAY"GLOBAL DATA IN/COUNT IS MISSING";
?END JOB.
```

Using File Attributes to Inquire about File Residence

The Boolean file attribute RESIDENT can be used instead of the file residence inquiry to check on the residence of files at the local host that is declared in the job. For example;

```
?BEGIN JOB;
FILE F (TITLE=BRANCH/SALES, KIND=DISK);
IF F(RESIDENT) THEN
    DISPLAY"FILE F IS RESIDENT"
ELSE
    DISPLAY"FILE F IS NOT RESIDENT";
?END JOB.
```

The file residence inquiry cannot be used to check on files that reside on a remote system connected through a BNA network. However, this capability is provided by the AVAILABLE file attribute. AVAILABLE attempts to open a file, and then returns an integer value indicating a successful open (and thus the existence of the file) or the reason for failure, without suspending the program or requiring operator intervention. For example:

```
?BEGIN JOB;
FILE F (TITLE=BRANCH/SALES, KIND=DISK, HOSTNAME=MLM);
IF F(AVAILABLE) = 1 THEN
    DISPLAY"FILE F IS RESIDENT AT MLM"
ELSE
    DISPLAY"FILE F IS NOT RESIDENT AT MLM";
?END JOB.
```

Boolean File Attribute Primary

<Boolean file attribute primary>

— <file identifier> — (— <Boolean file attribute> —) —————|

Explanation

The Boolean file attribute primary returns the value of a Boolean file attribute of the specified file.

Boolean Task Attribute Primary

<Boolean task attribute primary>

— <task identifier> — (— <Boolean task attribute> —) —————|

Explanation

The Boolean task attribute primary returns the value of a Boolean task attribute associated with the specified task variable.

Task State

<task state>

— <task identifier> —

IS
ISNT

INUSE
COMPLETED
SCHEDULED
ACTIVE
STOPPED
ABORTED

 —————|

```

┌ COMPLETEDOK ─┐
└ COMPILEDOK   ─┘

```

Explanation

The state returns information about the status of a task. The task must be associated with a task variable that has the specified task identifier.

The following are the different task states.

Task State	Meaning
INUSE	The task is SCHEDULED, ACTIVE, or STOPPED.
COMPLETED	The task was initiated and has terminated.
SCHEDULED	The task has not yet been initiated by the system.
ACTIVE	The task is currently running.
STOPPED	The task was stopped by the operator, suspended by the system, or programmatically suspended.
ABORTED	The task faulted or was discontinued.
COMPLETEDOK	The task is completed and was terminated without faulting or being discontinued.
COMPILEDOK	The task compiled without syntax errors.

The result of the task state expression is dependent only on the value of the STATUS attribute for the task. If the value of the STATUS attribute for the task is NEVERUSED, then all task state expressions return the values FALSE for IS and TRUE for ISNT.

Example

The following example uses the task state expression:

```

RUN OBJECT/X [TVAR];
IF TVAR IS COMPLETEDOK THEN
  SUB1
ELSE IF TVAR IS ABORTED THEN
  DISPLAY"UNSUCCESSFUL RUN";

```

Arithmetic Comparison

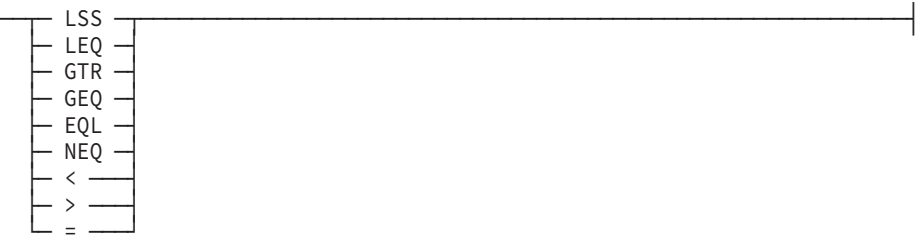
<arithmetic comparison>

```

┌ <real expression> ─┐ <real relation> ────────────────────▶
└ <integer expression> ─┘
▶┌ <real expression> ───────────────────────────────────────────▶
└ <integer expression> ─┘

```

<real relation>



Explanation

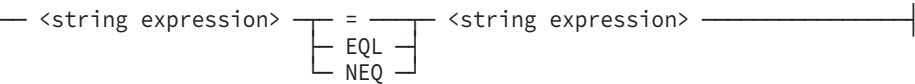
An arithmetic comparison compares the values of two real or integer expressions. If their values are related in the way specified by the real relation, the arithmetic comparison returns TRUE; otherwise, it returns FALSE.

The real relations have the following meanings.

Real Relation	Meaning
LSS, <	Less than
LEQ	Less than or equal to
GTR, >	Greater than
GEQ	Greater than or equal to
EQL, =	Equal to
NEQ	Not equal to

String Comparison

<string comparison>

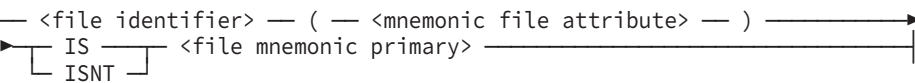


Explanation

A string comparison is a Boolean primary that enables comparison of the values of two string expressions. Two strings are equal only if all characters in the first string occur in the same order as in the second string, and the lengths of the two strings are equal.

File Mnemonic Comparison

<file mnemonic comparison>

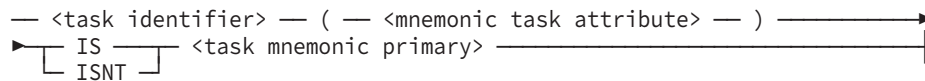


Explanation

A file mnemonic comparison is used to inquire about the values of mnemonic file attributes associated with a file. The file mnemonic comparison returns TRUE if the specified file attribute has the same value as the file mnemonic primary. Refer to [Mnemonic Primaries](#) and [Interrogating File Attributes](#) for related information.

Task Mnemonic Comparison

<task mnemonic comparison>



Explanation

A task mnemonic comparison is used to inquire about the values of task attributes associated with a task variable. The task mnemonic comparison returns TRUE if the specified task attribute has the same value as the task mnemonic primary.

Example

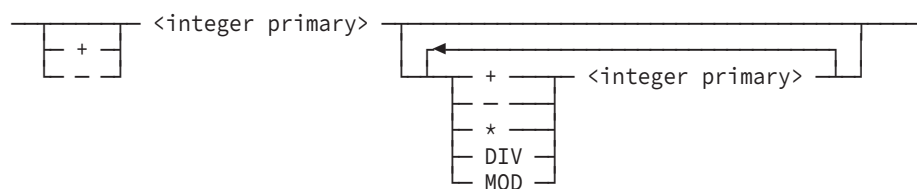
The following example includes two task mnemonic comparisons that interrogate the value of the STATUS attribute:

```
PROCESS RUN OBJECT/X [TVAR1];
PROCESS RUN OBJECT/Y [TVAR2];
DO WAIT UNTIL
  TVAR1(STATUS) IS TERMINATED AND TVAR2(STATUS) IS TERMINATED;
```

This example initiates two asynchronous tasks and then waits for both of them to finish before proceeding.

Integer Expressions

<integer expression>



Explanation

The plus sign (+) operator gives the sum of the integer primaries; the minus sign (-) operator gives their difference; the asterisk (*) operator gives their product. The DIV operator gives a quotient with the fractional part truncated. The MOD operator gives the remainder after the first integer primary has been divided by the second.

The order of evaluation (highest first) for arithmetic operations is as follows:

- Integer primary
- Prefix + or –
- *, DIV, or MOD
- Infix + or –

First, all integer primaries are evaluated; second, the prefix + or – (if any) is applied to the integer primary that it precedes. Finally, the remaining operations are performed in decreasing order of priority. If two operations are of the same priority, the left operation is performed first. When an integer expression is enclosed in parentheses, it becomes an integer primary.

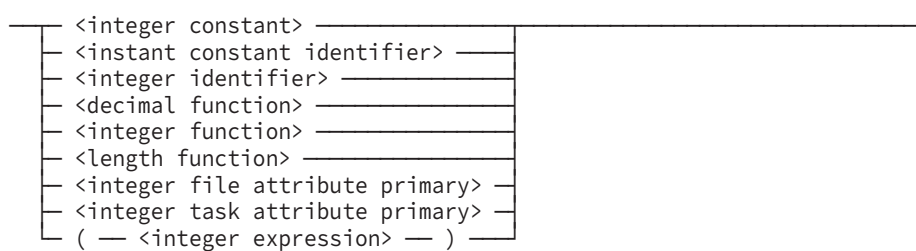
Example

3 + 1 = 4

7 MOD 4 = 3

Integer Primary

<integer primary>



Explanation

Syntax for integer constants, integer constant identifiers, and integer identifiers is given in [Section 8, Basic Constructs](#). The other kinds of integer primaries are defined in the following topics.

DECIMAL Function

<decimal function>

— DECIMAL — (— <string function> —)

Explanation

The DECIMAL function returns an integer value equal to the decimal (base 10) number represented by the value of the string expression. The string expression must contain at least 1 and not more than 12 characters. All characters included in the value of the string expression must be within the set of characters "0123456789". A runtime error occurs if the string expression does not satisfy these requirements.

Examples

The following are examples of the DECIMAL function:

```
DECIMAL("10") % YIELDS 10 (DECIMAL)
DECIMAL("255") % YIELDS 255 (DECIMAL)
```

INTEGER Function**<integer function>**

```
— INTEGER — ( — <real expression> — ) —————|
```

Explanation

The INTEGER function returns a result equal to the real expression but without the fractional part. Truncation of the fractional part occurs whether the function is invoked explicitly or implicitly.

Example

The following is an example where I is an INTEGER variable and R is a REAL variable containing the value "123.673":

```
I := INTEGER(123.673); % YIELDS 123 (EXPLICITLY INVOKED)
I := R; % YIELDS 123 (IMPLICITLY INVOKED)
```

LENGTH Function**<length function>**

```
— LENGTH — ( — <string expression> — ) —————|
```

Explanation

The LENGTH function returns the number of characters contained in the value of the string expression. The LENGTH function supports string expressions of up to 1800 characters.

Example

```
LENGTH("ABCDEF") % YIELDS 6
```

Integer File Attribute Primary**<integer file attribute primary>**

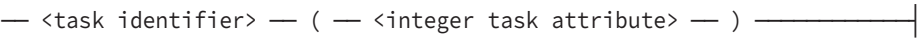
```
— <file identifier> — ( — <integer file attribute> — ) —————|
```

Explanation

The integer file attribute primary returns the value of an integer file attribute associated with the specified file.

Integer Task Attribute Primary

<integer task attribute primary>



Explanation

The integer task attribute primary returns the value of an integer task attribute associated with the specified task variable.

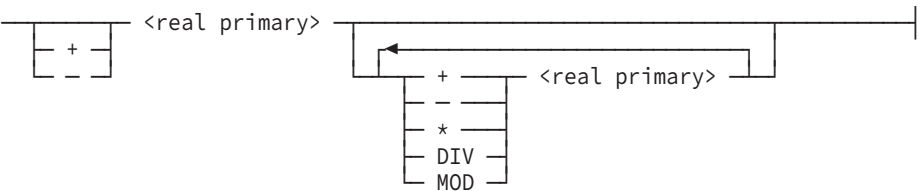
Example

In the following example, MYJOB(SOURCESTATION) returns the logical station number (LSN) of the station that originated the job:

```
RUN (WALLY)NUMB/COUNT;  
STATION = MYJOB(SOURCESTATION);
```

Real Expressions

<real expression>



Explanation

The plus sign (+) operator gives the sum of the real primaries; the minus sign (–) operator gives their difference; the asterisk (*) operator gives their product. The slash (/) operator gives a quotient that preserves the fractional part; the DIV operator gives a quotient with the fractional part truncated. The MOD operator gives the remainder after the first real primary has been divided by the second.

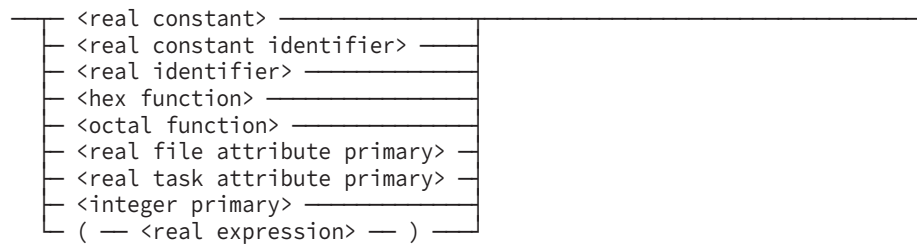
The order of evaluation (highest first) for arithmetic operations is as follows:

- Real primary
- Prefix + or –
- *, /, DIV, or MOD
- Infix + or –

First, all real primaries are evaluated; second, the prefix + or – (if any) is applied to the real primary that it precedes. Finally, the remaining operations are performed in decreasing order of priority. If two operations are of the same priority, the left operation is performed first. When a real expression is enclosed in parentheses, it becomes a real primary.

Real Primary

<real primary>

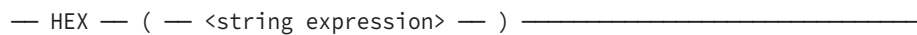


Explanation

Syntax for real constants, real constant identifiers, and real identifiers is provided in [Section 8, Basic Constructs](#). The syntax for an integer primary is given earlier in this section, and the other real primaries are defined in the following topics.

HEX Function

<hex function>



Explanation

The HEX function returns a real value equal to the hexadecimal (base 16) number represented by the value of the string expression. The string expression must contain at least 1 and not more than 12 characters. All characters in the value of the string expression must be within the set of characters "0123456789ABCDEF". A runtime error occurs if the string expression does not satisfy these requirements.

Examples

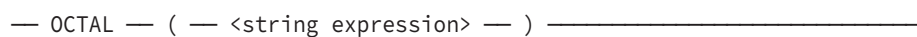
The following examples are HEX functions:

```
HEX("10") % YIELDS 16 (DECIMAL)
```

```
HEX("FF") % YIELDS 255 (DECIMAL)
```

OCTAL Function

<octal function>



Explanation

The OCTAL function returns a real value equal to the octal (base 8) number represented by the value of the string expression. The string expression must contain at least 1 and not more than 16 characters. All characters in the value of the expression must be within the set of characters "01234567". A runtime error occurs if the string expression does not satisfy these requirements.

Examples

The following examples illustrate OCTAL functions:

```
OCTAL("10") % YIELDS 8 (DECIMAL)
```

```
OCTAL("377") % YIELDS 255 (DECIMAL)
```

Real File Attribute Primary

<real file attribute primary>

— <file identifier> — (— <real file attribute> —) —————|

Explanation

The real file attribute primary returns the value of a real file attribute associated with the specified file.

Real Task Attribute Primary

<real task attribute primary>

— <task identifier> — (— <real task attribute> —) —————|

Explanation

The real task attribute primary returns the value of a real task attribute associated with the specified task variable.

Example

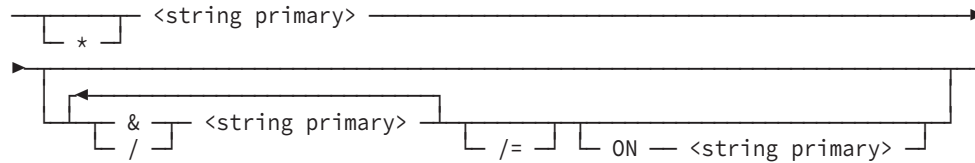
The following example illustrates a real task attribute primary:

```
RUN STAT/PROG [TV1];  
RUN NEW/STAT/PROG [TV2];  
IF TV2(ACCUMIOTIME) < TV1(ACCUMIOTIME) THEN  
  DISPLAY"NEW FEATURE SAVES IO TIME"  
ELSE  
  DISPLAY"NEW FEATURE IS A FLOP";
```

In this example, real task attribute primaries are used to return the accumulated I/O time of two tasks.

String Expressions

<string expression>



Explanation

The operators in a string expression have the following meanings.

Operator	Meaning
*	Adds an asterisk (*) prefix to the string primary
&	Concatenates two or more strings
/	Concatenates two or more strings and inserts a slash (/) between each string primary
/=	Adds a slash-equal (/=) suffix to the string primary
ON	Inserts the string " ON " between two string primaries. Note: The word ON is preceded and followed by a blank character.

When the ampersand (&) operator is used to concatenate two or more strings, a new string is created with a length equal to the sum of the lengths of the original strings. The new string is formed by joining a copy of the last string to the end of a copy of the previous string, and so forth, until all strings are concatenated. Therefore, all string concatenation operations are performed from left to right.

String expressions containing at least one string variable can contain up to 1800 characters. A runtime error results from an attempt to create a larger string than is allowed.

The asterisk (*), slash (/), slash-equal (/=) and ON operators are intended to aid in building file titles from strings.

When a string expression is enclosed in parentheses, it becomes a string primary.

Examples

The following two examples illustrate how to use these operators to build file titles.

This example creates the file title SALESJANUARY ON PAK.

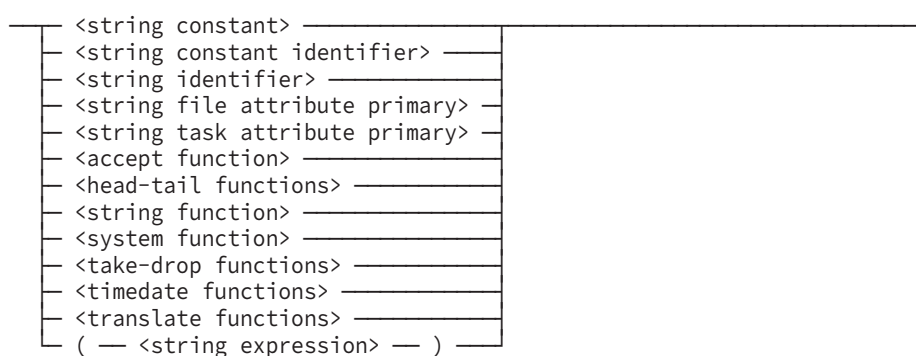
```
STR1:="SALES";
STR2:="JANUARY";
PGMNAME:= STR1 & STR2 ON"PAK";
```

This example creates the file title SALES/JONES/= ON ABC.

```
STR1:="SALES";
STR2:="JONES";
PGMNAME:= STR1/STR2/= ON"ABC";
```

String Primary

<string primary>

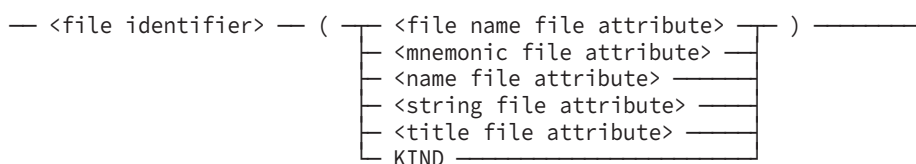


Explanation

Syntax for string constants, string constant identifiers, and string identifiers is provided in [Section 8, Basic Constructs](#). The other kinds of string primaries are defined in the following topics.

String File Attribute Primary

<string file attribute primary>



Explanation

A string file attribute primary returns the value of an attribute of the file with the specified file identifier. The file attribute must be of type mnemonic, name, file name, string, or title.

The KIND attribute returns the kind of device the file resides on.

The file attribute SERIALNO cannot be used as a string primary.

Refer to [Interrogating File Attributes](#) for related information.

Example

The following example returns the current value of the SECURITYTYPE attribute of the file variable FILEA:

```
FILEA (SECURITYTYPE)
```

Possible values for SECURITYTYPE are CONTROLLED, GUARDED, PRIVATE, or PUBLIC. See the *File Attributes Programming Reference Manual* for more information.

String Task Attribute Primary

<string task attribute primary>

```
— <task identifier> — ( — <file name task attribute> — ) —————|
                        |
                        | — <mnemonic task attribute> —
                        | — <name task attribute> —
                        | — <string task attribute> —
                        | — <title task attribute> —
                        | — ACCESSCODE —
                        | — FAMILY —
                        | — OPTION —
                        | — USERCODE —
```

Explanation

A string task attribute primary returns the value of a task attribute associated with the specified task variable. The task attribute can be of type mnemonic, name, file name, string, or title.

Additionally, a string task attribute primary can be used to return the value of the complex task attributes FAMILY, USERCODE, ACCESSCODE, and OPTION. Refer to [Interrogating Complex Task Attributes](#) for further information.

Example

The following example returns the current value of the STATUS attribute of task variable TASKA:

```
TASKA (STATUS)
```

Possible values include NEVERUSED, SCHEDULED, and ACTIVE.

ACCEPT Function

<accept function>

```
— ACCEPT — ( — <string expression> — ) —————|
```

Explanation

The ACCEPT function displays a string on the ODT and waits for an operator to respond by way of an AX (Accept) system command. For a description of this command, refer to the *System Commands Reference*.

The AX command can also be entered from the MARC Action line, as explained in the *MARC Operations Guide*. If the job was initiated through CANDE, the string is also displayed at the originating terminal, and can be replied to by using CANDE ?AX command. For details, refer to the control commands in the *CANDE Operations Reference Manual*.

The first 430 characters of the AX operator response are returned as the value of the ACCEPT function.

The response to the ACCEPT function can be entered before the actual execution of that function.

The string expression specified in the ACCEPT function displays a maximum of 430 characters, although the string expression can contain up to 1799 characters.

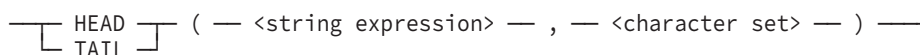
Example

The following example runs a program and then examines the task state to see whether the program terminated normally. If it did not, then the value of the HISTORYTYPE attribute is displayed, and an ACCEPT function asks the user whether they want the job to continue:


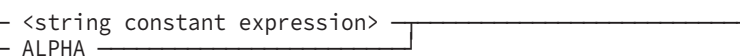
```
RUN OBJECT/PROG ON ORDSPK [T];
IF T ISNT COMPLETEDOK THEN
BEGIN
  DISPLAY"JULY RUN TERMINATED ABNORMALLY DUE TO"
  & T(HISTORYTYPE);
  IF ACCEPT("DO YOU WISH TO CONTINUE? YES OR NO") NEQ"YES" THEN
  ABORT"JOB ABORTED AT YOUR REQUEST";
END;
```

HEAD and TAIL Functions

<head-tail functions>

 (— <string expression> — , — <character set> —) — |

<character set>

 NOT — |  ALPHA — <string constant expression> — |

Explanation

The character set specifies a collection of characters used to control the action of the HEAD and TAIL functions. The predefined character set ALPHA consists of the letters A through Z and the digits 0 through 9. The string constant expression can have characters in any order. If the character set is of the form NOT string constant expression, then the character set consists of all EBCDIC characters except those specified in the string constant expression.

The HEAD function returns a string consisting of a copy of all leading characters in the string expression that belong to the set of characters in the character set. If the first character in the string expression is not a member of the character set, a null string ("") is returned.

The TAIL function returns a new string consisting of a copy of all the characters in the string expression that remains after the removal of all the leading characters belonging to the character set. If all characters in the string expression are members of the character set, a null string is returned.

For any string expression S and any character set C, the following relation is always true:

$$S = \text{HEAD}(S, C) \ \& \ \text{TAIL}(S, C)$$

The HEAD and TAIL functions allow string expressions of up to 1800 characters for the parameter.

Examples

The following examples illustrate use of the HEAD and TAIL functions:

```
STR1:=" A B C";
STR2:= HEAD(STR1," "); % RESULT =" "
STR2:= TAIL(STR1," "); % RESULT ="A B C"

STR1:="FILE/NAME";
STR2:= HEAD(STR1,NOT"/"); % RESULT ="FILE"
STR2:= TAIL(STR1,NOT"/"); % RESULT ="/NAME"
```

STRING Function

<string function>

— STRING — (— <integer expression> — , —————>
 ▶ <integer expression>) —————|
 * —————

Explanation

The STRING function returns a string that is formed by taking the absolute value of the first integer expression and converting it to its EBCDIC character representation. The length of the returned string is specified by the second parameter; if this second parameter is an integer expression with a value less than or equal to zero, the returned string is of length zero.

If the value of the second integer expression is greater than the minimum number of characters needed to represent the first argument, a sufficient number of leading zeros are provided.

If the value of the second integer expression is less than the number of characters needed to represent the first integer expression, the right-most characters are returned.

If the second parameter of the STRING function is an asterisk (*), the returned string will be long enough to contain all the digits of the character representation of the first argument with no leading zeros.

A run-time error occurs if the value of the second parameter is less than 0 or greater than 1800.

If you use a real expression in place of the first integer expression, the value is rounded to the nearest integer. However, if you use a real expression in place of the second integer expression, the fractional part of the value is simply truncated. For example, the function STRING(433.5, 2.5) returns the value "34". This result occurs because the value 433.5 is rounded up to 434, but the length value of 2.5 is truncated to 2.

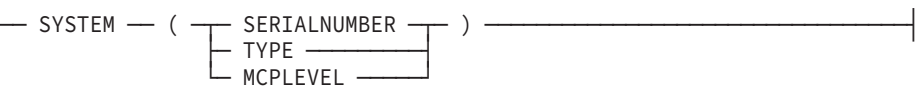
Examples

The following examples illustrate the STRING function:

```
STR1:= STRING(123,*); % RESULT ="123"  
STR2:= STRING(123,6); % RESULT ="000123"  
  
STR1:= STRING(123.456, 7); % RESULT ="0000123"  
STR2:= STRING(1234,3); % RESULT
```

SYSTEM Function

<system function>



Explanation

The SYSTEM function is a string function that returns system identification information.

Examples

The following information can be requested:

SYSTEM (SERIALNUMBER)

Returns the serial number of the system as a string; the length of the string will vary. For example:

```
"1000"
```

SYSTEM (TYPE)

Returns the machine type as a string; the length of the string will vary. For example:

```
"CS7101"
```

SYSTEM (MCPLEVEL)

Returns the version and cycle of the current MCP as a string of six characters. The first two characters represent the version of the MCP and the last three characters represent the cycle of the MCP. A period (.) separates the version and the cycle. For example:

```
"37.100"
```

TAKE and DROP Functions**<take-drop functions>**

```
┌ TAKE ─ ( ─ <string expression> ─ , ─ <integer expression> ) ─┐
└ DROP ─┘
```

Explanation

The TAKE function returns a string formed by taking the first integer expression number of characters from the string expression.

The DROP function returns a string with the characters remaining in the string expression after the first integer expression number of characters have been discarded.

A run-time error occurs if the value of the second parameter is less than 0 or greater than 1024, or is greater than the number of characters in the string expression.

For any string expression S and any integer expression I in the range 0 LEQ I LEQ LENGTH(S), the following relation is always true:

```
S = TAKE(S,I) & DROP(S,I)
```

If you use a real expression instead of an integer expression, the WFL compiler automatically truncates the fractional part, rather than rounding off the value. For example, the function TAKE("ABCDEF", 2.9) returns the value "AB", not "ABC".

The TAKE and DROP functions allow string expressions of up to 1032 characters for the parameter and the result.

Example

The following example illustrates the TAKE and DROP functions:

```
STR1 := "ABCDEF" ;
STR2 := TAKE(STR1, 2) ;           % RESULT = "AB"
STR2 := DROP(STR1, 2) ;          % RESULT = "CDEF"
```

TIMEDATE Functions

<timedate function>

— TIMEDATE — (HHMMSS)
	HHMMSSPPPP	
	YYYYMMDDHHMMSS	
	YYYYMMDDHHMMSSPPPP	
	DISPLAY	
	MONTH	
	DAY	
	DAYNUMBER	
	YYDDD	
	YYMMDD	
	MMDDYY	
	DDMMYY	
	YYYYDDD	
	YYYYMMDD	
	MMDDYYYY	
	DDMMYYYY	

Explanation

The TIMEDATE function is a string function that can be used to obtain the time or date, or both, in various forms. All strings returned by the TIMEDATE function appear in uppercase.

Examples

The forms of the TIMEDATE function are described in the following paragraphs. Each description includes an example that indicates the result that would be returned at 5:09:00.7124 p.m., Wednesday, July 4, 2001.

TIMEDATE (HHMMSS)

Returns the time as a string of six characters. The first two characters represent the hours on a 24-hour clock, the next two characters represent the minutes, and the last two characters represent the seconds. For example:

"170900"

TIMEDATE (HHMMSSPPPP)

Returns the time as a string of 10 characters. The first two characters represent the hours on a 24-hour clock, the next two characters represent the minutes, the next two characters represent the seconds, and the last four characters represent the ten thousandths of a second. For example:

"1709007124"

TIMEDATE (YYYYMMDDHHMMSS)

Returns the time and date as a string of 14 characters. The first eight characters represent the date: four characters for the year, two characters for the month, and two characters for the day of the month. The last six characters represent the time: two characters for the hours on a 24-hour clock, two characters for the minutes, and two characters for the seconds. For example:

"20010704170900"

TIME DATE (YYYYMMDDHHMMSSPPPP)

Returns the time and date as a string of 18 characters. The first eight characters represent the date: four characters for the year, two characters for the month, and two characters for the day of the month. The last 10 characters represent the time: two characters for the hours on a 24-hour clock, two characters for the minutes, two characters for the seconds, and four characters for the ten thousandths of a second. For example:

"200107041709007124"

TIME DATE (DISPLAY)

Returns the time, day of the week, and date in a display-type format. The length of the string varies from 27 to 38 characters. For example:

"5:09 PM WEDNESDAY, JULY 4, 2001"

TIME DATE (MONTH)

Returns the name of the month as a string. The length of the string varies from three to nine characters. For example:

"JULY"

TIME DATE (DAY)

Returns the name of the day of the week as a string. The length of the string varies from six to nine characters. For example:

"WEDNESDAY"

TIME DATE (DAYNUMBER)

Returns the number of the day of the week as a string of one character. The days of the week are numbered as follows: Sunday=0, Monday=1, Tuesday=2, Wednesday=3, Thursday=4, Friday=5, Saturday=6. For example:

"3"

TIME DATE (YYDDD)

Returns the date as a string of five characters. The first two characters represent the year (modulo 100). The last three characters represent the day of the year. For example:

"01185"

TIME DATE (YYMMDD)

Returns the date as a string of six characters. The first two characters represent the year (modulo 100). The following two characters represent the month, and the last two characters represent the day of the month. For example:

"010704"

TIMEDATE (MMDDYY)

Returns the date as a string of six characters. The first two characters represent the month. The following two characters represent the day of the month, and the last two characters represent the year (modulo 100). For example:

"070401"

TIMEDATE (DDMMYY)

Returns the date as a string of six characters. The first two characters represent the day of the month. The following two characters represent the month, and the last two characters represent the year (modulo 100). For example:

"040701"

TIMEDATE (YYYYDD)

Returns the date as a string of seven characters. The first four characters represent the year. The last three characters represent the day of the year. For example:

"2001185"

TIMEDATE (YYYYMMDD)

Returns the date as a string of eight characters. The first four characters represent the year. The following two characters represent the month, and the last two characters represent the day of the month. For example:

"20010704"

TIMEDATE (MMDDYYYY)

Returns the date as a string of eight characters. The first two characters represent the month. The following two characters represent the day of the month, and the last four characters represent the year. For example:

"07042001"

TIMEDATE (DDMMYYYY)

Returns the date as a string of eight characters. The first two characters represent the day of the month. The following two characters represent the month, and the last four characters represent the year. For example:

"04072001"

TRANSLATE Functions

<translate functions>

┌───┐ ┌───┐ ┌───┐
└───┘ LOWERCASE └───┘ (— <string expression> —) └───┘
└───┘ UPPERCASE └───┘

Explanation

The LOWERCASE and UPPERCASE functions return a string of the same length as the string expression. Each character of the string expression is translated to upper or lower case characters according to the designated function.

Examples

The following examples illustrate the LOWERCASE and UPPERCASE functions:

```
STR1 := "a1b2c3d4e5f6g7";
STR2 := UPPERCASE(STR1);           % RESULT = "A1B2C3D4E5F6G7"
STR2 := LOWERCASE("TRANSLATEME"); % RESULT = "translateme"
```

Mnemonic Primaries

<file mnemonic primary>

```

<file mnemonic> —————
|
| <file identifier> — ( — <mnemonic file attribute> — ) —
|
| # — <string primary> —————
```

<task mnemonic primary>

```

<task mnemonic> —————
|
| <task identifier> — ( — <mnemonic task attribute> — ) —
|
| # — <string primary> —————
```

The following table defines the mnemonic syntax elements.

Mnemonic	Description
<file mnemonic>	Any mnemonic value that can be assigned to a file attribute of type mnemonic. The mnemonic values available for each mnemonic file attribute are listed in the <i>File Attributes Programming Reference Manual</i> .
<task mnemonic>	Any mnemonic value that can be assigned to a task attribute of type mnemonic. The mnemonic values available for each mnemonic task attribute are listed in the <i>Task Attributes Programming Reference Manual</i> .

Explanation

File mnemonic primaries and task mnemonic primaries enable mnemonic-valued attributes to be used in comparisons and assignments.

In mnemonic attribute comparisons and assignments, the specific attributes and mnemonics must be compatible. For example, the MYUSE attribute of one file (or one of the MYUSE mnemonics CLOSED, IN, OUT, or IO) can be compared with or assigned to

the MYUSE attribute of another file. However, the values of the MYUSE and KIND attributes cannot be compared or assigned to each other, because they are different attributes. If the # string primary form is used, it must also represent a compatible mnemonic.

Example

In the following example, F is a file identifier, T is a task identifier, and S is a string identifier:

```
S := "ALGOL" ;
IF F(FILEKIND) IS #(S&"SYMBOL") THEN...
S := "NEVERUSED" ;
T(STATUS=#S) ;
STRING DISK ;
DISK := "PRINTER" ;
F(KIND=#DISK) ;                                % Yields KIND = PRINTER
```

Constant Expressions

A constant expression is a combination of basic elements with values that can be determined at compile time. These basic elements can be Boolean constants, integer constants, real constants, string constants, or job parameters that appeared in the job parameter list.

The constant expressions allowed by WFL are a subset of the various types of expressions that have been described earlier in this section. For example, Boolean constant expressions are a subset of Boolean expressions. The difference is that variables are not allowed in constant expressions.

A constant expression can be used anywhere an expression of the same type is allowed. Constant expressions can also be assigned as values in variable declarations, where expressions that use variables are not permitted.

The following pages give the formal syntax of Boolean constant expressions, integer constant expressions, real constant expressions, and string constant expressions.

Example

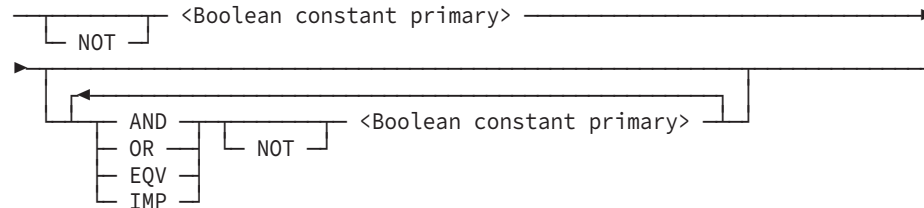
The following example illustrates constant expressions:

```
?BEGIN JOB PARAMPASS (BOOLEAN TF1, INTEGER X, STRING S);
  BOOLEAN BOOL1:= NOT TF1; % Boolean declaration
  INTEGER INT1:= X * 2; % Integer declaration
  RUN (DEVA)OBJECT/POOCH(BOOL1,INT1);
  RUN (NATTY)OBJECT/FREE(TAKE(S,2));
?END JOB.
```

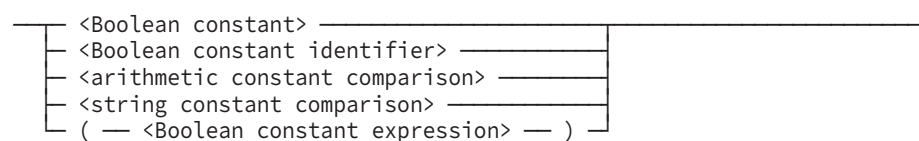

The Boolean declaration in this example uses a Boolean constant expression to initialize the variable. TF1 is a Boolean constant identifier that was passed in as a job parameter. The integer declaration uses an integer constant expression. X is an integer constant identifier that was passed in as a job parameter.

Boolean Constant Expression

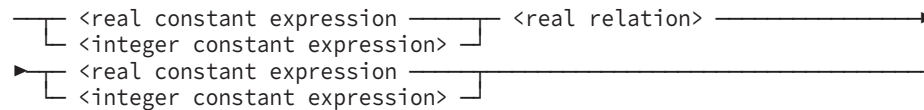
<Boolean constant expression>



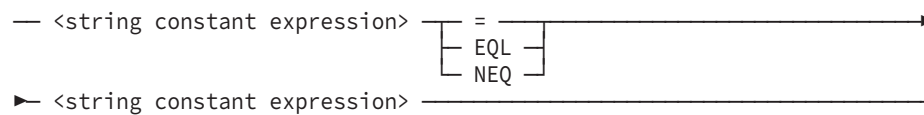
<Boolean constant primary>



<arithmetic constant comparison>



<string constant comparison>

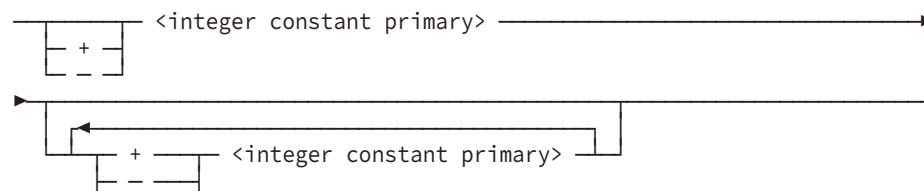


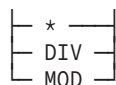
Explanation

Refer to [Boolean Expressions](#) for explanations of the various kinds of Boolean expressions.

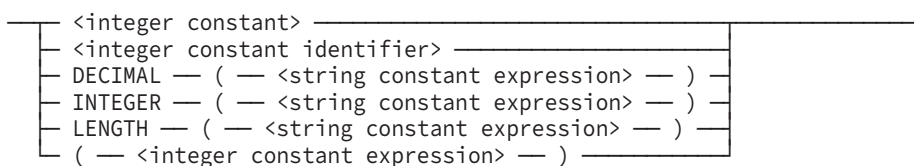
Integer Constant Expression

<integer constant expression>





<integer constant primary>

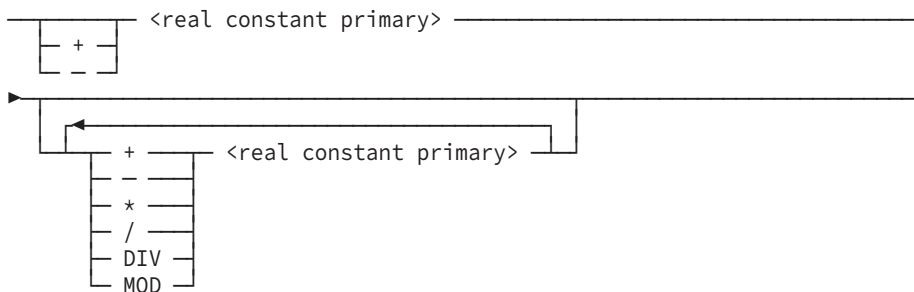


Explanation

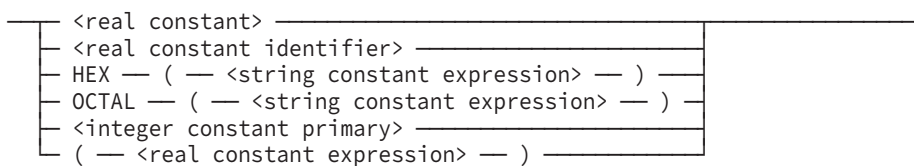
Refer to [Integer Expressions](#) earlier in this section for explanations of the various kinds of integer expressions.

Real Constant Expression

<real constant expression>

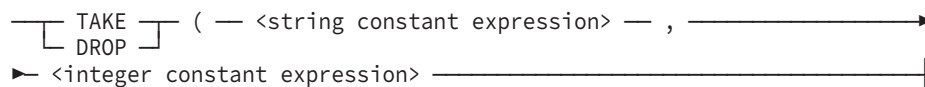
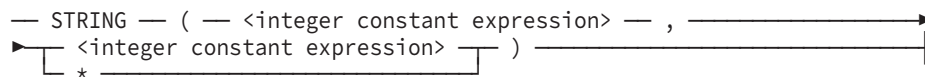
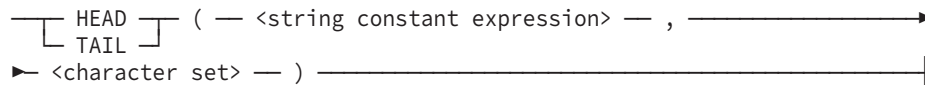
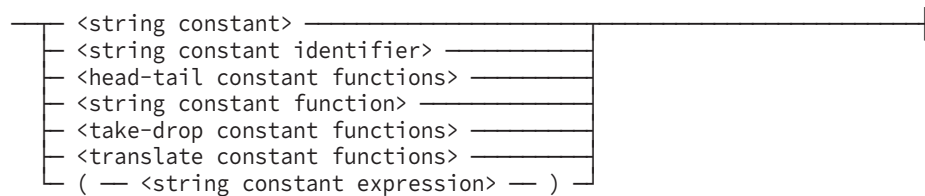
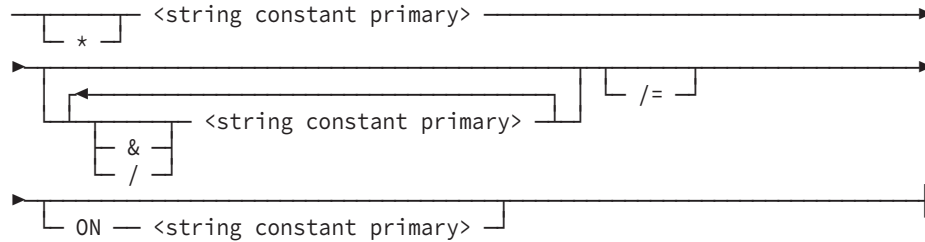


<real constant primary>



Explanation

Refer to [Real Expressions](#) earlier in this section for explanations of the various kinds of real expressions.



Refer to [String Expressions](#) earlier in this section for explanations of the various kinds of string expressions.

Section 8

Basic Constructs

Overview

The basic elements and constructs necessary to use WFL are listed and described in this section. Some general facts and restrictions regarding WFL elements are discussed in the following paragraphs, and many of the basic elements and constructs are syntactically defined on the following topics.

WFL uses the EBCDIC character set; use of any invalid EBCDIC character is illegal except in column 1.

Identifiers and numbers are terminated by any nonalphanumeric character (including a blank).

No identifier, constant, string, or multicharacter delimiter can be broken across a card boundary. Numeric constants and multicharacter delimiters cannot have embedded blank characters.

WFL source records can be terminated by a percent sign (%). The remainder of the WFL source record is not scanned and can contain any comments.

Invalid and Valid Characters

Using the <i> Construct

An <i> construct can be used instead of a semicolon (;) to separate statements, declarations, or job attributes. For an example of the use of the <i> construct as a statement separator, refer to [Statement List](#).

The <i> construct can precede the BEGIN JOB and END JOB constructs in WFL jobs, and is a required terminator for global or local data specifications. See [WFL Job Example](#).

Invalid Character Elements

The question mark (?) character is invalid when used as an <i> construct in the first column of a record.

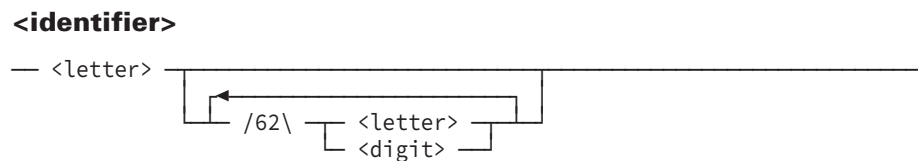
Valid Character Elements

The valid character elements are syntactically defined in the following table.

Element	Definition
<letter>	Any one of the 26 uppercase characters A through Z
<digit>	Any one of the 10 Arabic numerals 0 through 9
<nonquote EBCDIC character>	Any EBCDIC character for which the hexadecimal code is greater than or equal to 4"40" and that is not the EBCDIC double quotation mark (").
<nonsingle quote EBCDIC character>	Any EBCDIC character for which the hexadecimal code is greater than or equal to 4"40" and that is not the EBCDIC single quotation mark (').
<hyphen>	The single character hyphen (-)
<underscore>	The single character underscore (_)

Valid Character Elements

Identifiers



Explanation

The following is a list of the types of identifiers:

- Boolean identifier
- Boolean constant identifier
- File identifier
- Integer identifier
- Integer constant identifier
- Label identifier
- Real identifier
- Real constant identifier

- String identifier
- String constant identifier
- Subroutine identifier
- Task identifier

Identifiers all share the identifier syntax. Identifiers can serve as names for variables, constant identifiers, subroutines, and statements.

Some combinations of characters form words that have a special meaning to the WFL compiler. These words cannot be used as identifiers, or can only be used as identifiers in certain contexts. Refer to [Appendix B, Reserved Words, Predefined Words, and Keywords](#), for further information.

Examples

```
A
Z123
ABC123
```

Constants

<Boolean constant>

```

┌ TRUE ───────────────────────────────────────────────────────────────────────────────────┐
└ FALSE ───────────────────────────────────────────────────────────────────────────────────┘

```

<integer constant>

```

┌ /12\ ───────────────────────────────────────────────────────────────────────────────────┐
└ <digit> ───────────────────────────────────────────────────────────────────────────────────┘

```

<real constant>

```

┌ /1\ ─ . ───────────────────────────────────────────────────────────────────────────────────┐
└ /12\ ─ <digit> ───────────────────────────────────────────────────────────────────────────────────┘

```

<string constant>

```

┌ " ───────────────────────────────────────────────────────────────────────────────────┐
└ " ───────────────────────────────────────────────────────────────────────────────────┘
  ┌ /1024\ ───────────────────────────────────────────────────────────────────────────────────┐
  └ <nonquote EBCDIC character> ───────────────────────────────────────────────────────────────────────────────────┘
    ┌ " ───────────────────────────────────────────────────────────────────────────────────┐
    └ " ───────────────────────────────────────────────────────────────────────────────────┘

```

Explanation

A constant is a literal that contains information and is not changed by any operation. Boolean, integer, real, and string constants are defined in the following discussion.

The following constraints apply to string constants:

Basic Constructs

- A pair of quotation marks ("") appearing alone represents a null string (a string of length zero).
- A pair of quotation marks ("") appearing in a string represents one quotation mark (") within the string.
- A string constant cannot be broken across a card boundary.

Examples

The following are examples of integer constants:

```
12
750
12345
```

The following are examples of real constants:

```
3.1416
.2
1.0
```

The following are examples of string constants:

```
"ABC "
"?*-> "
"8-1 "
```

Names

Each file has a unique name that distinguishes it from every other file. A file name consists of parts, called nodes, separated by a slash (/). A file name can have from 1 to 12 nodes.

Long file names are available if you set the LONGFILENAME option of the SYSOPS command. Long file names are similar to traditional file names, except they can have from 1 to 20 nodes and a maximum node size of 215 characters.

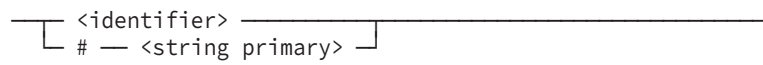
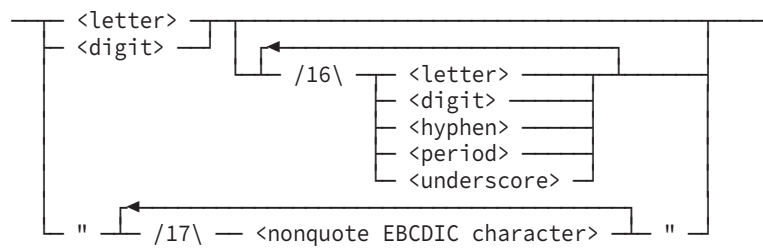
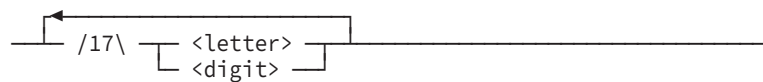
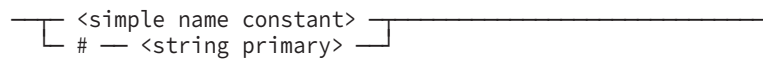
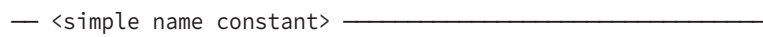
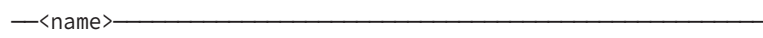
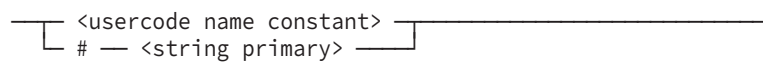
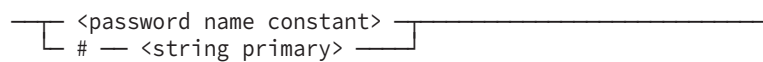
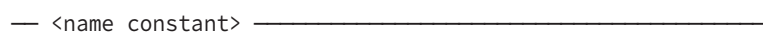
Not all Unisys software supports long file names. For more information about long file names, refer to the *System Operations Guide*.

<name>

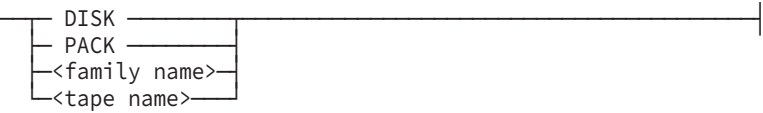
```
┌ <name constant> ───────────────────────────────────┐
└ # ─ <string primary> ─────────────────────────────────┘
```

<groupname>

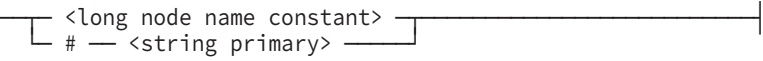
```
┌ <name constant> ───────────────────────────────────┐
└ # ─ <string primary> ─────────────────────────────────┘
```


<intname>**<name constant>****<groupname constant>****<simple name constant>****<hostname>****<family name>****<hostname constant>****<family name constant>****<tape name>****<usercode>****<password>****<usercode name constant>****<password name constant>**

<volume name>



<wrap password>



Explanation

The # <string primary> syntax can be used to dynamically build names, host names, family names, usercodes, and passwords. The run-time result must form a valid name constant, host name constant, family name constant, usercode name constant, or password name constant, respectively. Otherwise, a run-time error occurs. For further information, see [String Primary](#). Examples of using this syntax are provided in [File Names, Titles, and Directories](#).

When the security option CASESENSITIVEPW is set, lowercase characters in a <password name constant> are not converted to uppercase, and some special characters can be included without enclosing the token in double quotes. The following special characters are accepted in such a password.

&	-	{	}	\	~
'	[]		:	\$
<	*	@	(_	+
>	=	!	^	?	#

The pound sign (#) cannot be the first character of the password, and the ampersand (&) cannot be used when the password is used in a WRAP or UNWRAP statement unless the password is enclosed in double quotes.

A <wrap password> consisting of a <wrap password constant> is limited to 215 characters. A longer password can be supplied using a <string primary>. The maximum length is 256 characters.

Examples

USERPACK	% Simple name constant
6PACK	% Simple name constant
OUTPUT-FILE	% Name constant
Secret^*Passw0rd	% Password name constant

File Names, Titles, and Directories

Each file has a unique name that distinguishes it from every other file. A file name consists of parts, called nodes, separated by a slash (/). A file name can have from 1 to 12 nodes.

It is not a good idea to create files with names that contain "=" (quoted equal sign) as a <node name constant>. In ADD, ALTER, ARCHIVE, CATALOG, CHANGE, COPY, MOVE, REMOVE, REPLACE, UNWRAP, and WRAP statements, when the last node of a source file name is "=", the system treats it as a directory name.

For example, the COPY statement, COPY *X/"= TO PACK, copies all files in *X/= to PACK. The REMOVE statement, REMOVE (USER) "= ON PACK, removes all files under the USER usercode on PACK.

Note: You do not need to enclose a node in quotation marks if the node includes a period. Any file name displayed by the MCP continues to have nodes containing periods surrounded by quotation marks.

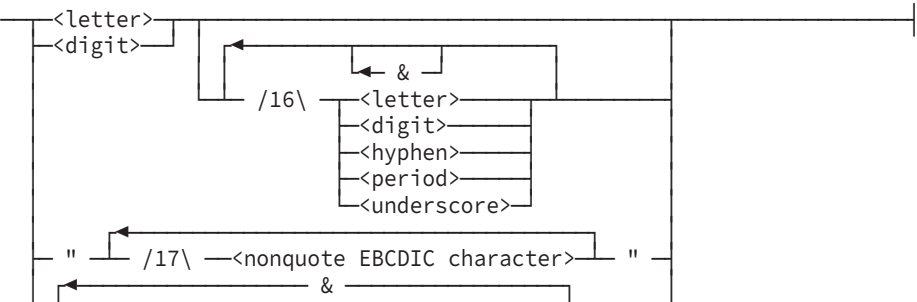
Long file names are available if you have set the LONGFILENAMES option of the SYSOPS command. Long file names are similar to traditional file names, except they can have from 1 to 20 nodes and a maximum node size of 215 characters.

If long file names are disabled, the long name constructs are redefined as follows.

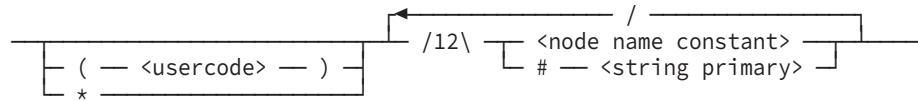
If SYSOPS LONGFILENAMES is reset, then . ..	Is equivalent to ...
<long file name constant>	<file name constant>
<long directory name constant>	<directory name constant>
<long file name>	<file name>
<long file title>	<file title>
<long directory name>	<directory name>
<long directory title>	<directory title>

Not all Unisys software supports long file names. For more information about long file names, refer to the *System Operations Guide*.

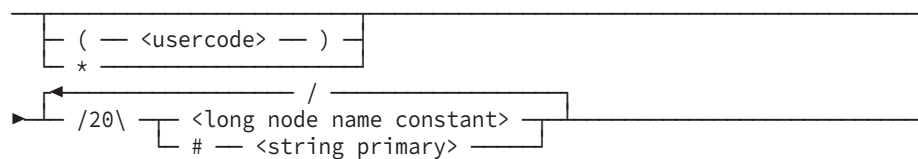
<node name constant>



<file name>



<long file name>



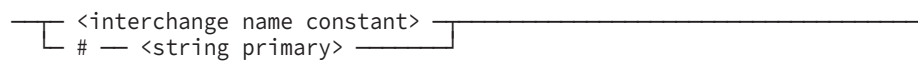
<file title>



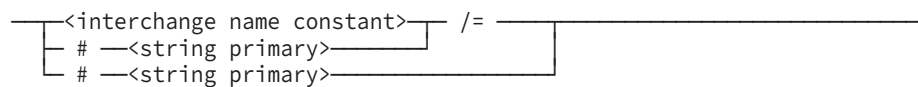
<long file title>



<universal file name>

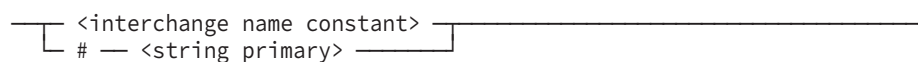


<universal directory name>

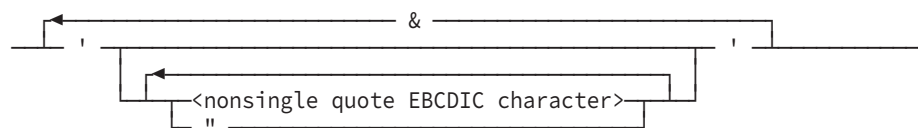


Note: If the string referenced by **<string primary>** is terminated with **"/=**", the **"/=**" in **<universal directory name>** cannot be used.

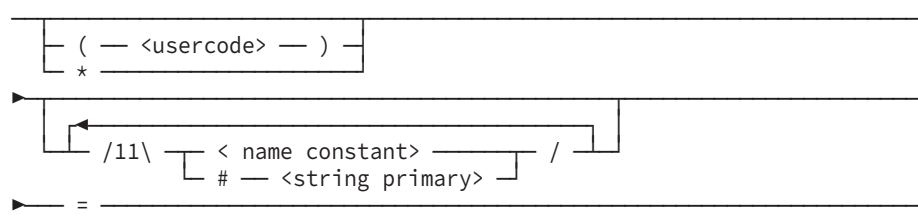
<interchange file name>



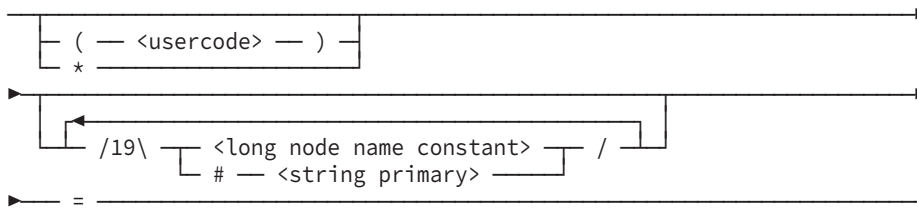
<interchange name constant>



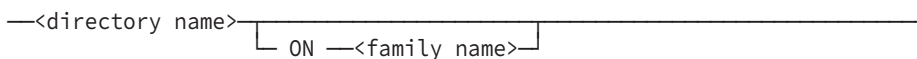
<directory name>



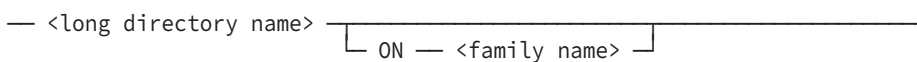
<long directory name>



<directory title>



<long directory title>



Explanation

In the node name constant and the long node name constant, the ampersand (&) operator must be on the same line as the preceding text. When the & operator is used to concatenate two or more names in double quotation marks (""), a new node name is created within pair of double quotation marks. If the new concatenated node name contains more than 215 characters (not counting the double quotation marks), an error message is issued.

In the interchange name constant, the ampersand (&) operator must be on the same line as the preceding text. When the & operator is used to concatenate two or more names in single quotation marks ('), a new name is created within a pair of single quotation marks. If the new concatenated name contains more than 250 characters (not counting the single quotation marks), an error message is issued.

If a usercode is not specified for a name or title and the task is running under a usercode, the file or directory will first be looked for under the usercode of the task and, if not found, will then be looked for without a usercode.

Note: When you use the *ALTER*, *CATALOG*, *CHANGE*, *DELETE*, *MODIFY*, *REMOVE* and *SECURITY* statements, only the usercode of the task is searched.

If a name or title is preceded by the optional asterisk (*), it indicates that the search for the file or directory is to be done as if the task were not running under a usercode.

As an alternate syntax for indicating a file associated with a usercode, instead of preceding the file name with the usercode parentheses, you can precede the file name with `*USERCODE/<usercode>`. You can use this alternate form when you refer to either a file or to a directory of files under the specified usercode. The directory `*USERCODE/=` refers to all usercoded files on a given family.

Examples

```
?BEGIN JOB;  
  USERCODE=UC;  
  FAMILY DISK = PRIMARY OTHERWISE ALTERNATE;  
  RUN OBJECT/TEST;  
?END JOB.
```

In this example, the file OBJECT/TEST will be searched for under different file titles and family names in the following order:

- (UC)OBJECT/TEST on the primary family
- *OBJECT/TEST on the primary family
- (UC)OBJECT/TEST on the alternate family
- *OBJECT/TEST on the alternate family

The system will use the first file it finds.

If OBJECT/TEST were changed to *OBJECT/TEST in this example, only the file titles and family names given in items 2 and 4 in the preceding list would be used in the search.

The statement COPY *= FROM T1(KIND=TAPE) copies all of the files from the tape T1, whether or not their titles have usercodes, because the usercode of the task is not used to precede the file names.

An interchange file name is a string of up to 250 characters surrounded by single quotation marks ('). A single quotation mark can be embedded in the string by including a pair of adjacent single quotation marks.

The following CHANGE statement changes the first node of each filename in *X/= into a usercode:

```
CHANGE *X/= TO *USERCODE/=
```

For example, the previous statement would change file *X/A/B to (A)B. Note that a job must have privileged status to place files under a usercode different from that of the job itself.

Using String Primaries

The # <string primary> syntax can be used to dynamically build file names, file titles, directory names, and directory titles. The run-time result must form a valid file name constant, file title constant, directory name constant, or directory title constant respectively; otherwise, a run-time error occurs.

A string primary can contain an entire file name, file title, directory name, or directory title. A string primary can also take the place of any part of these names and titles, as long as the result is of the correct form. The following examples clarify this constraint.

In the following examples, S1 and S2 are string identifiers and F is a file identifier:

Basic Constructs

```
S1:="B";
F(TITLE = A/#S1/C);           % Resulting title = A/B/C
S1:="(A)B";
F(TITLE = #S1/C);           % Resulting title = (A)B/C
F(TITLE = S1/C);           % Resulting title = S1/C
S1:="*USERCODE/X";
F(TITLE = #S1/OBJECT/T);    % Resulting title = *USERCODE/X/OBJECT/T
                             % Which is an alternate method of specifying (X)OBJECT/T
S1:="A/B";
S2:="PQ";
F(TITLE = *#(S1 ON S2));    % Resulting title = *A/B ON PQ
S1:="LONG20CHARACTERTITLE";
F(LTITLE = A/#S1);         % Resulting long title = A/LONG20CHARACTERTITLE
                             % Assuming SYSOPS LONGFILENAMES is set
```

Restrictions on the Use of String Primaries

Note that the # <string primary> syntax is used to form an individual file name, file title, directory name, or directory title. Multiple names cannot be combined into a single string primary. Also, parameters to an object program cannot be combined in the same string primary as the object code file title. The following examples illustrate this concept.

Passing Parameters to a Task

The following examples show both the correct and incorrect WFL job program for parameter passing to a task.

Correct

In the following program example, the parameter XYZ is successfully passed to the object program REPORT/PGM:

The job JOB/WFLRUN:

```
?BEGIN JOB PROGRAM(STRING PROGTORUN,
                    STRING PARAMNAME);
    TASK PROGTASK;
    RUN OBJECT/#PROGTORUN(PARAMNAME) [PROGTASK];
?END JOB
```

Started as:

```
START JOB/WFLRUN( "REPORT/PGM", "XYZ" )
```

Incorrect

The following program example is incorrect because the parameter XYZ is not passed separately from the task name:

The job JOB/WFLRUN:


```
?BEGIN JOB PROGRAM(STRING PROGTORUN);  
    TASK PROGTASK;  
    RUN OBJECT/#PROGTORUN [PROGTASK];  
?END JOB.
```

Started as:

```
START JOB/WFLRUN( "REPORT/PGM(XYZ)" );
```

Copying Multiple Files

The following examples show both the correct and incorrect use of a WFL job program for copying multiple files.

Correct

In the following program, the file name string parameters TEST/A and TEST/B are successfully passed to the WFL copy job:

The job JOB/WFLCOPY:

```
?BEGIN JOB WFLCOPY(STRING FNAME1,  
                    STRING FNAME2);  
    COPY #FNAME1, #FNAME2  
    FROM DISK TO PACK;  
?END JOB.
```

Started as:

```
ST JOB/WFLCOPY( "TEST/A", "TEST/B" )
```

Incorrect

The following program example is incorrect because the file name string parameters TEST/A and TEST/B are not passed with their own string parameters:

The job JOB/WFLCOPY:

```
?BEGIN JOB WFLCOPY(STRING FNAME);  
    COPY #FNAME  
    FROM DISK TO PACK;  
?END JOB.
```

Started as:

```
ST JOB/WFLCOPY( "TEST/A, TEST/B" )
```


Section 9

WFL Control Options

Overview

WFL control options affect the way the WFL compiler processes a job.

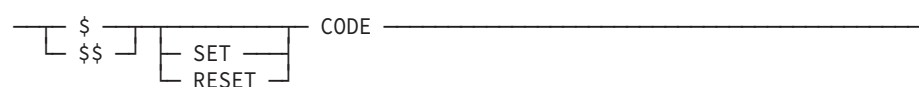
The control options can appear anywhere in a job except within data cards and strings. However, if the job is started from a file, then any line of the job containing control options must begin with a dollar sign (\$) in column 1 or 2. The occurrence of dollar signs in both columns 1 and 2 is equivalent to a single dollar sign in column 2. The dollar sign can be followed by one or more of the WFL control options.

If the NEWSOURCE job disposition is specified in the job, then only control options following a dollar sign in column 2 are written to the new source file. Control options following a dollar sign in column 1 are not written to the new source file. The position of the dollar sign also determines whether text specified by the INCLUDE control option is written to the new source file (refer to "INCLUDE Option" in this section for details). Refer to "Job Disposition" in Section 3 for information about the NEWSOURCE job disposition.

If the job is submitted from an ODT, or in array form from a user program, then control options must still be preceded by a dollar sign. The dollar sign can be in any column, and can be followed by one or more control options. A semicolon (;) must be included after the control options.

CODE Option

<code control option>



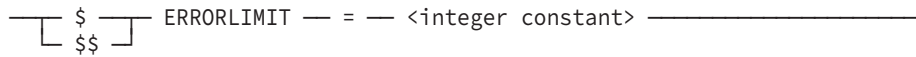
Explanation

This control option controls the printing of the object code into a printer backup file. CODE can generate output from a START FOR SYNTAX statement.

SET is an optional action indicator that causes the source file to be printed. RESET is an optional action indicator that prevents the object code from printing. The default value is RESET.

ERRORLIMIT Option

<errorlimit control option>



Explanation

This control option sets the error limit for job compilation to the value of the integer constant. Job compilation is terminated if the number of errors detected by the WFL compiler becomes greater than or equal to the error limit. Note that this option affects compilation, not execution, of a job. A job containing even one error is not executed, regardless of the error limit setting.

If no ERRORLIMIT option appears, the default error limit is 100 unless the job was started through CANDE. If the job was started through CANDE, the default error limit is 6.

More than one ERRORLIMIT option can be included in the job. In this case, the error limit is changed whenever the option appears, and the next time the compiler encounters an error, it compares the new error total with the current error limit to determine whether to terminate compilation.

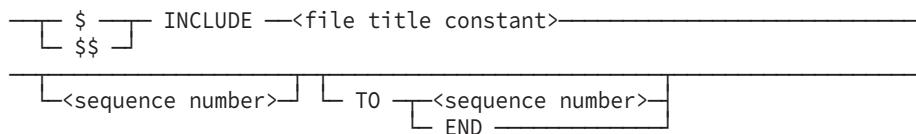
Example

Compilation of the following job is terminated if the number of errors detected becomes greater than or equal to 50:

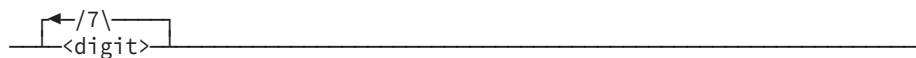
```
?BEGIN JOB RUNPROG;  
$ERRORLIMIT = 50;  
RUN OBJECT/PROG1;  
RUN OBJECT/PROG2;  
.  
.  
.  
?END JOB.
```

INCLUDE Option

<include control option>



<sequence number>



Explanation

This option incorporates card images from another file into the current job during compilation. An INCLUDE option can appear in a job stored in a disk file and initiated by a START. The text is included each time the job is initiated. The file title constant is the name of the file that contains the included text. The FILEKIND of the file must be JOBSYMBOL.

If NEWSOURCE is specified and the dollar sign (\$) is in column 1, the card images specified by the INCLUDE control option is written to the new source file where the option appeared in the old source file. If the dollar sign is in column 2, the INCLUDE control option is written to the new source file, but the card images specified by that option is not. The included text is in the WFL job listing in the job summary printout.

If a sequence range is specified, only that portion of the file is included. If the sequence range is omitted, the entire file is included.

The included records can themselves contain INCLUDE options; in this way, included source input can be nested up to five levels deep.

After parsing the job attribute list, WFL runs under the USERCODE , FAMILY, and EXECUTE PATH attribute specifications of the job (if they are supplied).

If an INCLUDE is encountered in the job heading, the usercode, family, and execute path of the initiator are used to locate the file and determine access. If an INCLUDE is encountered in the job body, the usercode, family and execute path seen in the job heading (if any are supplied) are used to locate the file and determine access.

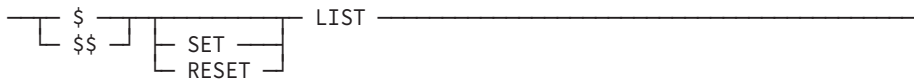
Note: The value of the EXECUTE PATH attribute and not the DATAPATH is used. This is because the system applies execute access to INCLUDE files.

A job started from the ODT without a usercode can also include a file in the same directory as the started job if the INCLUDE occurs in the job heading.

Example

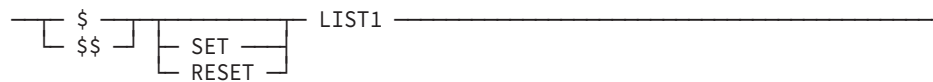
The following partial job uses the INCLUDE option to incorporate text from the file CARDLINE/ATTRIBUTES, which is located on the MYPACK pack, into the ZZZ job during compilation:

```
?BEGIN JOB ZZZ;  
RUN SYSTEM/CARDLINE;  
$INCLUDE CARDLINE/ATTRIBUTES ON MYPACK  
.  
.  
.  
?END JOB.
```



LIST1 Option

<list1 control option>



Explanation

This control option controls the printing of the WFL source file into a printer backup file. LIST1 can generate output from a START FOR SYNTAX statement.

SET is an optional action indicator that causes the source file to be printed. RESET is an optional action indicator that prevents the source file from printing. The default value is RESET.

The \$LIST1 compiler option is reset if the user does not have read access to the job source file and any included files. The source images are not printed in the printer backup file.

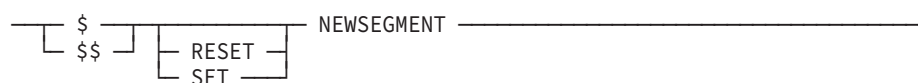
Example

In this example, the LIST1 option is used to print the WFL source file and the included file:

```
-      % #2: *BD/0020801/0008864/201612150536/000LINE ON PACK (Records: 1-9 of 9)
00000000  $ list1
00000100  begin job testit;
00000200  integer i;
00000300  $ include j/include/1
1.00000100 myjob (options = (todisk, fault, dsed, arrays,
1.00000200                      base, libraries, files, code));
00000400  i := i / 0;
00000500  end job
----- End of *BD/0020801/0008864/201612150536/000LINE ON PACK
```

NEWSEGMENT Option

<newsegment control option>



Explanation

This control option causes subroutines to be stored in new code segments, which enables excessively large WFL subroutines to compile. Before you use the NEWSEGMENT option, you should attempt to reduce the size of a large subroutine so that WFL compiles without the NEWSEGMENT option.

WFL Control Options

If you use the NEWSEGMENT option, the option can be set before the subroutine and reset after the subroutine. The default value is RESET.

Note: You should only use this option if the following message displays while compiling a large subroutine:

```
ERROR: CODE SEGMENT CAPACITY EXCEEDED - THE SUBROUTINE <name>
IS TOO LARGE AND MUST BE BROKEN INTO SMALL SUBROUTINES.
***** COMPILATION ABORTED *****
```

Example

In the following job, the NEWSEGMENT option creates a new code segment for each procedure. This option increases the size of the WFL code file and might cause the WFL job to execute more slowly. However, it might enable compilation of some excessively large WFL jobs that otherwise could not compile.

```
$$SET NEWSEGMENT
BEGIN JOB NEWSEGMENT:
FILE F;
SUBROUTINE SETUP;
BEGIN
F(TITLE=FILE/NAME,KIND=DISK,NEWFILE);
.
.
.
END;

SUBROUTINE MAKEFILE;
BEGIN
OPEN(F);
LOCK(F);
.
.
.
END MAKEFILE;
.
.
.

SETUP;
MAKEFILE;
.
.
.
END JOB;
```

WARNSUPPRESS Option

<warnsuppress control option>



Explanation

This control option controls the emitting of warning messages by the WFL compiler. SET is an optional action indicator that prevents warning messages from being emitted by the WFL compiler. RESET is an optional action indicator that causes warning messages to be emitted by the WFL compiler. SET is the default value. If the job was started through CANDE, the default value is RESET.

Example

In this example, the WARNSUPPRESS option is used to prevent a warning message from being emitted by the WFL compiler:

```
$SET WARNSUPPRESS
BEGIN JOB NOWARN;
FAMILY ABCDEFGHIJKLMNOPQRSTUVWXYZ = DISK ONLY;
COPY X AS Y;
END JOB
```

If the above job had \$RESET WARNSUPPRESS on the first line, then the following warning message would be emitted:

```
WARNING: FAMILY NAME LONGER THAN 17 CHARACTERS HAS BEEN TRUNCATED
```

XREF Option

<xref control option>



Explanation

If set, the compiler generates cross-reference information containing an alphabetized list of identifiers that appear in the program. For each identifier, the compiler generates the type of the item named by that identifier, the sequence number of the source input record on which the identifier is declared, the sequence number of the source input record on which the identifier is referenced, and other relevant information.

The cross-reference information is written to a disk file in raw form. The compiler initiates SYSTEM/XREFANALYZER to process this file. This information is discarded if any syntax errors occur during compilation.

If used, this option should be SET before any source input has been processed. Once SET, attempts to RESET it are ignored.

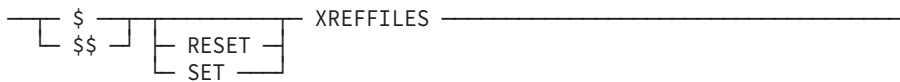
Example

The following partial job sets the XREF option to generate a printer backup file containing cross-reference information:

```
$SET XREF  
BEGIN JOB ZZZ;  
...  
END JOB
```

XREFFILES Option

<xreffiles option>



Explanation

If set, the compiler generates files containing analyzed cross-reference information in the program. These files can be used in EDITOR and INTERACTIVEXREF. The files have the titles XREFFILES/OBJECT/<file title>/DECS and XREFFILES/OBJECT/<file title>/REFS where <file title> is the title of the file containing the job.

The cross-reference information is written to a disk file in raw form. The compiler initiates SYSTEM/XREFANALYZER to process this file to generate the XREFFILES. This information is discarded if any syntax errors occur during compilation.

If used, this option should be SET before any source input is processed. Once SET, attempts to RESET it are ignored.

WFL is unlike other languages in that certain statements often continue for many lines. For example, the DATA statement continues until the <invalid> character is seen. The COMPILE statement can have multiple task and file equations to both the compiler and the resulting code file. These are declared in their own environment to make it easier to find the begin and end of the statement. The name of the environment is <statement>#<number>. For example, COMPILE#0 or DATA#10.

Example

The following partial job sets the XREFFILES option to generate xref files.

```
$SET XREFFILES  
BEGIN JOB ZZZ;  
...  
END JOB
```

Appendix A

Sample WFL Jobs

Overview

This appendix contains three sample jobs that demonstrate how the WFL features described in this manual are applied to some common situations.

Compiling a Program

This job applies a patch to a source file, compiles the source file into an executable object code file, and runs XREFANALYZER to produce cross-reference files.

```
?BEGIN JOB COMP (STRING PROGRAMNAME);
  CLASS=4;
  TASK
    PATCHTASK,          % TASK VARIABLE FOR SYSTEM/PATCH
    COMPILETASK;        % TASK VARIABLE FOR COMPILE
DATA PATCHINPUT
$# D O L L A R   O P T I O N S
$ SET MERGE SET NEW SET LINEINFO ERRLIST RESET LIST
$ SET XREF NOXREFLIST
$.% P A T C H E S
$#PATCH 1
$.FILE PATCH/1
? % END OF PATCHINPUT
RUN SYSTEM/PATCH [PATCHTASK];
  FILE SOURCE (TITLE=#PROGRAMNAME);
  FILE CARD (TITLE=PATCHINPUT,KIND=READER);
  FILE PATCH (TITLE=SYSTEMPATCH/#PROGRAMNAME,NEWFILE=TRUE);
IF PATCHTASK(TASKVALUE) = 1 THEN
  BEGIN
    REMOVE ERRORFILE/#PROGRAMNAME;
    COMPILE OBJECT/#PROGRAMNAME WITH ALGOL [COMPILETASK] LIBRARY;
    COMPILER FILE SOURCE      (TITLE=#PROGRAMNAME);
    COMPILER FILE CARD        (TITLE=SYSTEMPATCH/#PROGRAMNAME, DISK);
    COMPILER FILE NEWSOURCE   (TITLE=NEW/#PROGRAMNAME);
    COMPILER FILE ERRORS      (TITLE=ERRORS/#PROGRAMNAME, NEWFILE,
                              KIND=DISK, PROTECTION=PROTECTED);
  REMOVE SYSTEMPATCH/#PROGRAMNAME;
END;
```

```
IF COMPILETASK IS COMPILEDOK THEN
  BEGIN
    DISPLAY "COMPILED OK";
    IF FILE ERRORS/#PROGRAMNAME IS RESIDENT THEN
      DISPLAY "*** CHECK ERRORS/" & PROGRAMNAME
        & " FOR WARNINGS ***";
    MYSELF(JOBSUMMARY=SUPPRESSED); % DON'T NEED JOBSUMMARY
    SECURITY OBJECT/#PROGRAMNAME PUBLIC IO;
    IF FILE XREF/OBJECT/#PROGRAMNAME IS RESIDENT THEN
      BEGIN
        RUN SYSTEM/XREFANALYZER (0);
        TASKVALUE = -1; % PRODUCE XREFFILES
        FILE XREFFILE(TITLE=XREF/OBJECT/#PROGRAMNAME);
        SECURITY XREFFILES/OBJECT/#PROGRAMNAME/DECS,
          XREFFILES/OBJECT/#PROGRAMNAME/REFS PUBLIC IO;
      END;
    END
  ELSE
    DISPLAY "*** SYNTAX ERRORS: LIST ERRORS/" & PROGRAMNAME
      & " FOR ERRORS ***";
  ?END JOB.
```

Explanation

Each component of this sample job is explained in the following discussion.

Job Heading

The job accepts a string parameter PROGRAMNAME, which is the name of the program that is to be compiled.

The CLASS = 4 form is a job attribute specification that causes the job to be initiated from queue 4.

Declarations

The TASK declaration that follows declares two task variables for use later in the job.

PATCHINPUT Data Specification

The next section of the job is a global data specification named PATCHINPUT. This will be used as an input file by SYSTEM/PATCH in the next section of the job.

The first three lines of the global data specification provide options that modify the behavior of SYSTEM/PATCH. Refer to SYSTEM/PATCH in the *System Software Utilities Operations Reference Manual*, and to compiling programs in the *ALGOL Reference Manual, Volume 1* for descriptions of the various compiler control options that are used in this global data specification.

The next three lines of the global data specification specify the names of any patch files that are to be used. In this case, only one is specified, PATCH/1.

SYSTEM/PATCH Run

The next section of the job runs SYSTEM/PATCH. The following file equations are used.

Equation	Meaning
FILE SOURCE	Specifies the file that is to be patched. In this case, the file whose title was passed in as the WFL job parameter is used.
FILE CARD	Specifies the file to be used as the input to SYSTEM/PATCH. In this case, the global data specification titled PATCHINPUT is used.
FILE PATCH	Specifies the title of the file that SYSTEM/PATCH creates by merging the patches and compiler control options included in the CARD file.

Compilation

The job verifies that the SYSTEM/PATCH run was successful by checking the TASKVALUE of the task variable PATCHTASK. This attribute will have a value of 1 if the run was successful. If the run was successful, then any error file that might have been generated by an earlier compilation of the program is removed, and the program is compiled. The following file equations follow the COMPILE statement.

Equation	Meaning
FILE SOURCE	Specifies the file to be used as the secondary source input.
FILE CARD	Specifies the file to be used as the primary source input. In this case, the merged patch file produced by the earlier run of SYSTEM/PATCH is used.
FILE NEWSOURCE	Specifies the title of the updated source file that is produced by merging the CARD file and the TAPE file. (This file is produced because the NEW option was included in the PATCHINPUT file.)
FILE ERRORS	Specifies the title of the error file that is produced if errors or warnings occur during compilation.

Refer to compiling programs in the *ALGOL Reference Manual, Volume 1* for further information about the input and output files used by the ALGOL compiler.

The merged patch file produced by SYSTEM/PATCH is then removed, as it is no longer needed.

SYSTEM/XREFANALYZER Run

Next, the job verifies that the compilation was successful by checking the task state of the task variable that was attached to the compilation.

If the compilation was successful, the message "COMPILED OK" is displayed. The job then checks to see if an error file was created by the compilation. If it was, a message is displayed stating that the error file should be checked for warnings.

At this point, the JOBSUMMARY attribute of the job is set to SUPPRESSED. This statement is located here so that it will suppress the job summary only if the compile was successful. If the compile was unsuccessful, the job summary printout might contain useful information about why it failed.

The security of the object code file is then set to PUBLIC IO, so that it will be available to users under other usercodes.

The job then runs the XREFANALYZER utility to produce an analysis of where all the identifiers in the program are declared and used. For a description of this utility, refer to XREFANALYZER in the *System Software Utilities Operations Reference Manual*. XREFANALYZER uses a file called XREF/OBJECT/#PROGRAMNAME, which was created during the compilation of the program because the compiler control options XREF and NOXREFLIST were included in the PATCHINPUT global data specification.

XREFANALYZER produces two output files. The next statement sets the security of these two files to PUBLIC IO, so that they will be available to users under different usercodes.

If the compilation had not been successful, the job would have skipped these last few actions and simply displayed a message about syntax errors being present in the program.

Initiating Other Jobs

This job initiates several other jobs and programs on a daily basis. The jobs and programs perform routine maintenance tasks such as removing unwanted files, updating other files to the most current version, and setting system options.

```
?BEGIN JOB DAILY/MAINT;
CLASS=5;
STARTTIME=7:00 ON + 1;
STRING D;
D:=TIMEDATE (MMDDYY);
DISPLAY D;
CASE DECIMAL(D) OF
  BEGIN
    (010190,
    041790,
    052590,
    070390,
    090790,
    112690,
    112790,
    122390,
```

```

122490,
122590,
123190):
    ; % TODAY'S A HOLIDAY, NO RUN NEEDED
ELSE:
    BEGIN
        START CLEANUP/FILES;
        START SFA9E/INITIALIZE;
        RUN *SETUP/CANDE/OPTIONS;
        START SFA9E/DAILY/UPDATED;
    END;
END; START DAILY/MAINT; ?END JOB.

```

Explanation

This job only needs to be initiated by an operator once. After that, it reinitiates itself every day. It does this with the START DAILY/MAINT statement at the end of the job. (DAILY/MAINT is the name of the file this job is stored in.) The STARTTIME specification in the job heading causes job initiation to be delayed until 7:00 a.m. on the following day.

The daily maintenance that this job performs is not needed on holidays. Therefore, the TIMEDATE function is used to check the current date, and the date value is assigned to the variable D. The CASE statement compares the value of D with the dates of all the holidays in the year. If D equals any of these dates, the daily maintenance jobs and programs are not initiated.

Updating Files

This job is a simplified example of a job that updates files used on the system.

```

?BEGIN JOB UPDATE/FILES;
CLASS=7; %SPECIAL QUEUE FOR OPS JOBS.
JOBSUMMARY=UNCONDITIONAL;
STRING AX;
TASK CTASK;
SUBROUTINE REMOVEFILES;
BEGIN
REMOVE
    *SYMBOL/ABE , *SYMBOL/ALGOL , *SYMBOL/ALGOLSUPPLEMENT
    *SYMBOL/ALGOLTABLEGEN, *SYMBOL/ATTABLEGEN, *SYMBOL/BACKUP ,
    *SYMBOL/BNA , *SYMBOL/BNAV1/= , *SYMBOL/BNAENVIRONMENT,
    *SYMBOL/BOOTSTRAP, *SYMBOL/BUFFERMANAGER, *SYMBOL/CANDE ,
    *SYMBOL/CARDLINE, *SYMBOL/CCTABLEGEN
    FROM DISK;
END REMOVEFILES;
SUBROUTINE COPYFILES;
BEGIN
COPY
    *SYMBOL/ABE , *SYMBOL/ALGOL , *SYMBOL/ALGOLSUPPLEMENT
    *SYMBOL/ALGOLTABLEGEN, *SYMBOL/ATTABLEGEN, *SYMBOL/BACKUP ,
    *SYMBOL/BNA , *SYMBOL/BNAV1/= , *SYMBOL/BNAENVIRONMENT,
    *SYMBOL/BOOTSTRAP, *SYMBOL/BUFFERMANAGER, *SYMBOL/CANDE ,
    *SYMBOL/CARDLINE, *SYMBOL/CCTABLEGEN

```

```
      FROM UPTAPE (TAPE) TO DISK (DISK) [CTASK];
IF CTASK (TASKVALUE) = 1 THEN
  BEGIN
    DISPLAY "COPYFILES DIDN'T WORK";
    AX:=ACCEPT ("ENTER YES TO RETRY COPYFILES OR NO TO GO ON");
    IF AX = "YES" THEN
      BEGIN
        INITIALIZE (CTASK);
        COPYFILES;
      END;
    END;
  END COPYFILES;
ON RESTART, GO TRYAGAIN;
TRYAGAIN:
REMOVEFILES;
COPYFILES;
?END JOB.
```

Explanation

The job attributes at the start specify the queue the job will be initiated from and ensure that a job summary will be printed.

The first subroutine, REMOVEFILES, removes a list of files from DISK. The second subroutine, COPYFILES, copies the new versions of those files from a tape named UPTAPE to DISK.

The REMOVEFILES subroutine is not strictly necessary if the system option 5 (AUTORM) is set, because in that case the COPY statement automatically removes any files on the destination volume that have the same titles as files that are being copied to that volume.

However, the COPY statement does not remove each old file until the file that is to replace it is completely copied over. This can cause a temporary shortage of disk space that could prevent the COPY from completing. Removing the old files prior to using the COPY statement ensures that this problem will not occur.

In the COPYFILES subroutine, the COPY is assigned the task variable CTASK. The TASKVALUE of the task is checked after the COPY completes to ensure that all files were copied successfully. If one or more of the files were not copied successfully (for example, because they were not present on the tape), then the TASKVALUE of the COPY task is 1.

If the COPY was not successful, messages are displayed asking the operator whether the COPY should be retried. The ACCEPT function assigns the operator's reply to a variable AX. If the reply was YES, the task variable CTASK is reinitialized and the subroutine invokes itself, thus causing the COPY to be tried over again.

The ON RESTART statement causes specified actions to be taken after a job is interrupted by a halt/load. In this case, both subroutines are rerun after a halt/load.

Appendix B

Reserved Words, Predefined Words, and Keywords

Overview

The WFL compiler recognizes certain words in a WFL job and associates them with specific meanings. There are three different categories of such words: reserved words, predefined words, and keywords.

Reserved words can never be used as identifiers. Predefined words can be used as identifiers, but lose their original meanings for the scope of the declaration. Keywords can be used as identifiers, and will be interpreted either as identifiers or keywords according to the context in which they are used.

Reserved words, predefined words, and keywords can all be used as names and will be interpreted as names where the context implies it.

File attributes and task attributes are treated as keywords in WFL.

Reserved Words

These words cannot be used as identifiers in WFL:

BEGIN	BOOLEAN	CONSTANT
DATA	EBCDIC	ELSE
END	FALSE	FILE
INTEGER	JOB	REAL
STRING	SUBROUTINE	TASK
THEN	TRUE	UNTIL

Predefined Words

The following words have predefined meanings in WFL. They can be declared as identifiers; however, they lose their predefined meanings for the scope of the declaration.

Reserved Words, Predefined Words, and Keywords

ABORT	ACCEPT	ACCESS	ALPHA
ALTER	ARCHIVE	BIND	CASE
COMPILE	COPY	CREATE	CRUNCH
DATABASE	DECIMAL	DECK	DROP
EXECUTE	HEAD	HEX	IF
INITIALIZE	INSTRUCTION	LENGTH	LOCK
LOG	LOWERCASE	MKDIR	MODIFY
MOVE	MYJOB	MYSELF	NOT
OCTAL	OK	OPEN	PB
PRINT	PROCESS	REMOVE	REPLACE
RERUN	RESTOREREPLACE	RETURN	REWIND
RUN	SECURITY	START	STARTJOB
STOP	SYSTEM	TAIL	TAKE
TIMEDATE	UNWRAP	UPPERCASE	USER
VOLUME	WAIT	WHILE	WRAP

The following words are predefined when they are used as statements; however, in some cases they can also be context-sensitive. The following table lists each word that falls into this category and explains the cases when the word is context-sensitive.

Word	Cases When Context-Sensitive
ADD	VOLUME statement CATALOG statement
CATALOG	COPY statement ADD statement
CHANGE	VOLUME statement
DISPLAY	TIMEDATE function
DO	WHILE statement
GO	COMPILE statement BIND statement
ON	COMPILE statement BIND statement
PASSWORD	ACCESS statement
PURGE	CATALOG statement

Word	Cases When Context-Sensitive
RELEASE	ARCHIVE statement
RESTORE	ARCHIVE statement
RESTOREADD	ARCHIVE statement

Keywords

Any words used in the syntax but not listed as reserved words or predefined words are keywords. Keywords are context-sensitive. If they appear in the correct context, their predetermined meanings are used; otherwise, they are assumed to be identifiers.

All file and task attributes and associated mnemonic values are keywords. File attributes and their mnemonics are described in the *File Attributes Programming Reference Manual*. Task attributes and their mnemonics are described in the *Task Attributes Programming Reference Manual*.

ABORTED	ACTIVE	ALGOL	ALL
AND	APRIL	AS	AT
AUGUST	BACKUP	BECOMEOWNER	BINDER
C	CC	CDROM	CLASS
COBOL	COBOL85	COMPILE	COMPILEDOK
COMPILER	COMPLETED	COMPLETEDOK	DAY
DAYNUMBER	DCALGOL	DDMMYY	DDMMYYYY
DECEMBER	DEFAULT	DELETE	DESTROY
DESTROYED	DIFFERENTIAL	DISK	DIV
DMALGOL	DRC	DSOERROR	EQL
EQV	ERRORLIMIT	FEBRUARY	FILES
FOR	FORTRAN	FORTRAN77	FRIDAY
FROM	FROMSTART	FTP	FULL
GEQ	GTR	HHMMSS	HOSTSERVICES
HS	IMP	IN	INCLUDE
INCREMENTAL	INTO	INUSE	IO
IS	ISNT	JANUARY	JULY
JUNE	LEQ	LIBMAINTDIR	LIBRARY
LIST	LSS	MARCH	MAY
MCPLEVEL	MERGE	MMDDYY	MMDDYYYY

Reserved Words, Predefined Words, and Keywords

MOD	MODULA2	MONDAY	MONTH
NDLII	NEQ	NEWP	NEWSOURCE
NFT	NOLIST	NOVEMBER	OCTOBER
OF	OFFSITE	ON	ONLY
ONSITE	ONTO	OPTIONAL	OR
ORIGIN	OTHERWISE	OUT	OUT OF
PACK	PASCAL	PASCAL83	PRIVATE
PROPAGATE	PUBLIC	QUEUE	RECOVER
RECOVERED	REFERENCE	REPORT	RESET
RESIDENT	RESTART	ROLLOUT	RPG
SATURDAY	SCHEDULED	SCRATCH	SECTORS
SELECT	SEPARATELY	SEPTEMBER	SERIALNUMBER
SET	SFTP	SKIPEXCLUSIVE	SORT
STARTTIME	STOPPED	SUNDAY	SYNTAX
TASKFAULT	THURSDAY	TO	TODAY
TOMORROW	TUESDAY	TYPE	UNTIL
USEPATH	USERS	VERIFY	WAITONERROR
WEDNESDAY	WITH	YYDDD	YYMMDD
YYYYDDD	YYYYMMDD	YYYYMMDDHHMMSS	

Appendix C

Understanding Railroad Diagrams

This appendix explains railroad diagrams, including the following concepts:

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

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

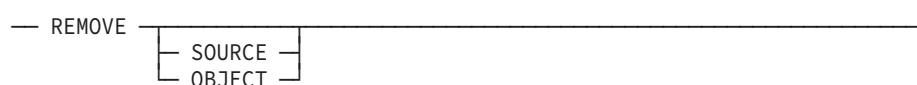
Railroad Diagram Concepts

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

Paths

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

The following example has three paths:



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

- REMOVE
- REMOVE SOURCE
- REMOVE OBJECT

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

Railroad diagrams are intended to show

- Mandatory items
- User-selected items

Understanding Railroad Diagrams

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

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

[Table C-1](#) introduces the elements of a railroad diagram.

Table C-1. Elements of a Railroad Diagram

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

Constants and Variables

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

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

If no part of the constant appears in boldface, the constant cannot be abbreviated.

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

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

In railroad diagrams, variables are enclosed in angle brackets.

In the following example, BEGIN and END are constants, whereas <statement list> is a variable. The constant BEGIN can be abbreviated, since part of it appears in boldface.

— **BEGIN** —<statement list>— END —————|

Valid abbreviations for BEGIN are

- BE
- BEG

- BEGI

Constraints

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

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

A description of each item follows.

Vertical Bar

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

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

Percent Sign

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

— STOP —————%

Right Arrow

The right arrow symbol (>) is used when the railroad diagram is too long to fit on one line and must continue on the next

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

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

Required Item

A required item can be

- A constant
- A variable
- Punctuation

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

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

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

— **EVENT** —`<identifier>`—————|

User-Selected Item

A user-selected item can be

- A constant
- A variable
- Punctuation

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

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

—

	+	
	–	

`<arithmetic expression>`—————|

Loop

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

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

—

	←	
	;	

`<field value>`—————|

Bridge

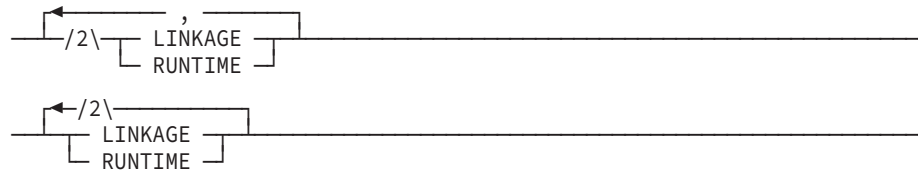
A loop can also include a bridge. A bridge is an integer enclosed in sloping lines (/ \) that

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

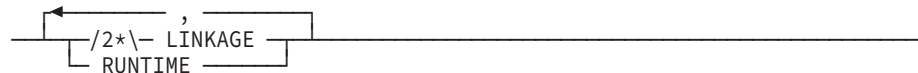
The bridge can precede both the contents of the loop and the return character (if any) on the upper line of the loop.

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

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



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

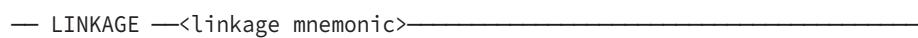


In the previous bridge example, you must enter LINKAGE at least once but no more than twice, and you can enter RUNTIME any number of times.

Following the Paths of a Railroad Diagram

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

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



Alternate paths are provided by

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

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

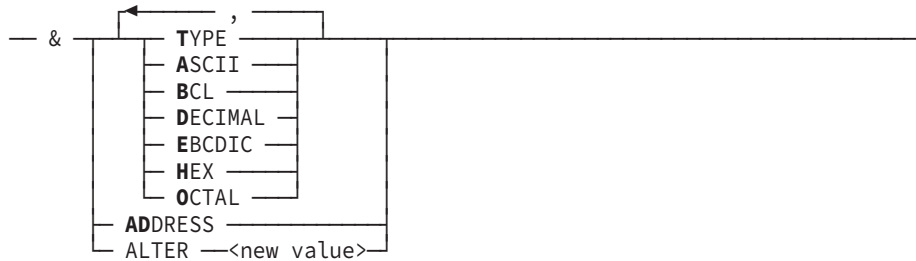
For example, the following railroad diagram consists of a top path and two alternate paths. The top path includes

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

Understanding Railroad Diagrams

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

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



Railroad Diagram Examples with Sample Input

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

Example 1

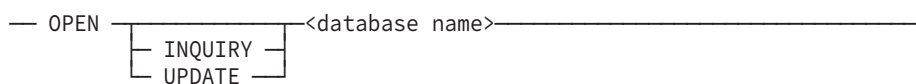
<lock statement>



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

Example 2

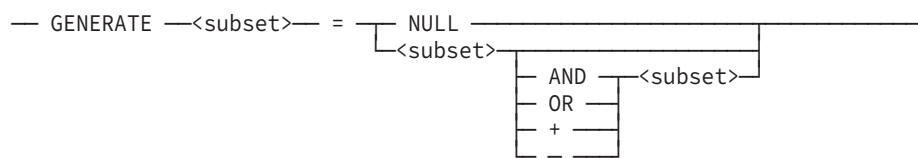
<open statement>



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

Example 3

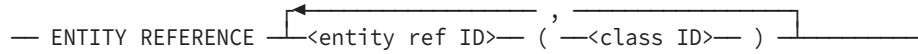
<generate statement>



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

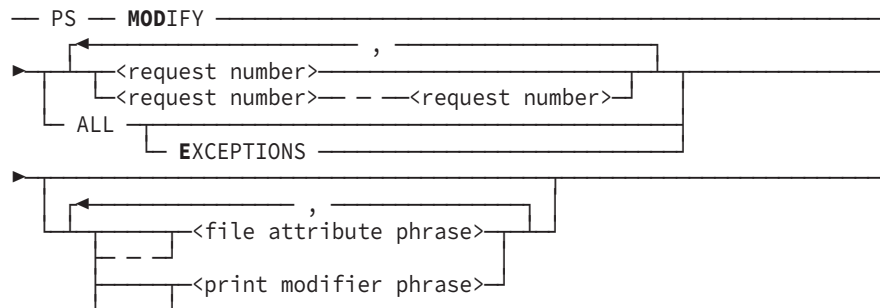
Example 4

<entity reference declaration>



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

Example 5



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

Sample Input	Explanation
PS MOD ALL EXCEPTIONS	The constants PS and MODIFY are followed by the user-selected constants ALL and EXCEPTIONS.

Index

A

- abort statement, 1–11, 6–4
 - example, 1–11
 - example, 6–5
- ABORTED task state, 7–5
- absolute pathname
 - in currentdirectory assignment, 5–13
- accept function, 1–16, 7–15
 - example, 1–16
 - in string primary, 7–14
- access statement, 6–5
 - explanation, 6–5
 - example, 6–5
- accesscode
 - password, 6–5
 - task attribute, 5–29
- accesscode task attribute
 - assigning value to, 5–11
 - example, 5–12
 - explanation, 5–11
 - in complex task attribute assignment, 5–11
 - in accesscode assignment, 5–11
- ACTIVE task state, 7–5
- add command
 - in CANDE, 2–3
 - in MARC, 2–6
- add or copy statement, 6–60, 6–61
- add statement, 1–15, 5–1, 6–6
 - copying FIFOs, 6–63
 - example, 1–15
 - in process statement, 6–141
 - example, 6–6, 6–105
- ALGOL programs, initiating WFL jobs, 2–7
- alignfile attribute
 - in alter statement, 6–8
- alignment attribute
 - in alter statement, 6–8
- alter attribute statement
 - in alter statement, 6–7
 - in alter statement, 6–7
- alter statement, 6–7, 6–8
 - alignfile attribute, 6–8
 - alignment attribute, 6–8
 - alternategroups attribute, 6–8
 - APL attribute, 6–9
 - banner attribute, 6–9
 - ccsversion attribute, 6–9
 - extdelimiter attribute, 6–9
 - extmode attribute, 6–9
 - filekind attribute, 6–10
 - formid attribute, 6–10
 - group attribute, 6–10
 - grouppr attribute, 6–10
 - groupprwx attribute, 6–10
 - groupw attribute, 6–10
 - groupx attribute, 6–10
 - guardowner attribute, 6–11
 - label attribute, 6–11
 - lockedfile attribute, 6–11
 - note attribute, 6–11
 - otherrr attribute, 6–11
 - otherrrwx attribute, 6–12
 - otherw attribute, 6–12
 - otherx attribute, 6–12
 - owner attribute, 6–12
 - ownerr attribute, 6–12
 - ownerrwx attribute, 6–13
 - ownerw attribute, 6–12
 - ownerx attribute, 6–13
 - pagecomp attribute, 6–13
 - printerkind attribute, 6–13
 - product attribute, 6–13
 - propagatesecuritytodirs attribute, 6–13
 - propagatesecuritytofiles attribute, 6–14
 - releaseid attribute, 6–14
 - savefactor attribute, 6–14
 - securityadmin attribute, 6–14
 - securityguard attribute, 6–14
 - securitymode attribute, 6–14
 - securitytype attribute, 6–14
 - securityuse attribute, 6–15
 - sensitivedata attribute, 6–15
 - setgroupcode attribute, 6–15
 - setusercode attribute, 6–15
 - trainid attribute, 6–15

- transform attribute, 6–15
- useguardfile attribute, 6–15
- userinfo attribute, 6–15
- example, 6–15
- alternate family name, 5–17
 - in family specification, 5–16
- alternategroups attribute
 - in alter statement, 6–8
- alternategroups value, 6–160
 - in security specification, 6–160
- ampersand (&), as a string operator, 7–13
- AND, as a Boolean operator, 7–1
- APL attribute
 - in alter statement, 6–9
- append attribute
 - in FTP transform, 6–113
- archive backup statement, 6–19
 - checking progress, 6–20
 - example, 6–20
- archive CD volume
 - attributes, 6–34
 - cdcopies attribute, 6–34
 - density attribute, 6–34
 - multivolume attribute, 6–35
 - packetwrite attribute, 6–35
 - savefactor attribute, 6–35
 - serialno attribute, 6–36
 - encrypt attribute, 6–34
 - offsite attribute, 6–35
- archive differential statement, 6–17
- archive disk volume, 6–22
 - attribute list, 6–23
 - attribute list in, 6–22
 - familyindex attribute, 6–23
 - in archive merge statement, 6–37
 - in archive purge statement, 6–38
 - in archive restore statement, 6–39
 - kind attribute, 6–23
 - serialno attribute, 6–23
- archive full statement, 6–17
- archive incremental statement, 6–17
- archive merge statement, 6–17, 6–37
 - example, 6–38
- archive options
 - in archive merge statement, 6–37
 - in archive restore statement, 6–39
 - in archive rollout statement, 6–41
 - in archive volume statement, 6–43
- archive purge statement, 6–17, 6–38
 - example, 6–38
- archive release statement, 6–17, 6–38, 6–39
- archive restoreadd statement, 6–17
 - example, 6–41
- archive restore statement, 6–17, 6–39
 - example, 6–41
- archive rollout statement, 6–17, 6–41
 - checking progress, 6–43
 - DRC option, 6–42
 - example, 6–43
 - sectors option, 6–42
 - using the DRC option with, 6–42
- archive statement, 5–1, 6–16
 - archive differential, 6–17
 - archive incremental, 6–17
 - archive purge, 6–17
 - archive restore, 6–17
 - archive rollout, 6–17
 - compare option, 6–21
 - dsonerror option, 6–21
 - archive full, 6–17
 - archive merge, 6–17
 - archive release, 6–17
 - archive restoreadd, 6–17
 - release option, 6–21
 - report option, 6–22
 - skipexclusive option, 6–22
 - specifying different, 6–17
 - waitonerror option, 6–22
 - options, 6–21
- archive subsystem, 6–16
 - autounload attribute, 6–25
 - blocksize attribute, 6–26
 - cdcopies attribute, 6–34
 - compressioncontrol attribute, 6–26
 - density attribute, 6–27, 6–34
 - different types, 6–17
 - encrypt attribute, 6–27, 6–34
 - familyowner attribute, 6–27
 - kind attribute, 6–28
 - libmaintappend attribute, 6–28
 - libmaintdir attribute, 6–30
 - locatecapable attribute, 6–30
 - multivolume attribute, 6–35
 - offsite attribute, 6–31, 6–35
 - packetwrite attribute, 6–35
 - savefactor attribute, 6–31, 6–35
 - scratchpool attribute, 6–32
 - securityguard attribute, 6–31
 - securitytype attribute, 6–32
 - securityuse attribute, 6–32
 - serialno attribute, 6–32, 6–36
 - usecatalog attribute, 6–32

- generations, 6-17
 - archive tape volume, 6-23
 - attribute list, 6-23
 - attribute list in, 6-23
 - attributes, 6-25
 - autounload attribute, 6-25
 - blocksize attribute, 6-26
 - compressioncontrol attribute, 6-26
 - compressionrequested attribute, 6-26
 - density attribute, 6-27
 - encrypt attribute, 6-27
 - example, 6-24
 - familyowner attribute, 6-27
 - in archive merge statement, 6-37
 - in archive rollout statement, 6-41
 - kind attribute, 6-28
 - libmaintappend attribute, 6-28
 - libmaintdir attribute, 6-30
 - locatecapable attribute, 6-30
 - offsite attribute, 6-31
 - savefactor attribute, 6-31
 - scratchpool attribute, 6-32
 - securityguard attribute, 6-31
 - securitytype attribute, 6-32
 - securityuse attribute, 6-32
 - serialno attribute, 6-32
 - usecatalog attribute, 6-32
 - archive tape volume attributes, 6-26
 - archive task equation
 - task attribute assignment, 6-36
 - library equation, 6-36
 - archive task equation list, 6-36
 - in archive merge statement, 6-37
 - in archive restore statement, 6-39
 - in archive rollout statement, 6-41
 - archive volume delete statement
 - example, 6-45
 - archive volume offsite statement
 - example, 6-45
 - archive volume onsite statement
 - example, 6-45
 - archive volume statement, 6-43
 - example, 6-45
 - arithmetic comparison, 7-5
 - arithmetic constant comparison, 7-25
 - in Boolean constant primary, 7-25
 - assigning values
 - task attributes, 5-24
 - task variables, example, 5-25
 - task variables, 5-23
 - to complex task attributes, 5-11
 - to file attributes, 5-39
 - to task attributes, 5-8
 - assignment statement, 6-2, 6-45, 6-46
 - example, 6-46
 - asterisk (*)
 - copy statement, 6-100
 - as a real operator, 7-10
 - as a string operator, 7-13
 - as an integer operator, 7-7
 - in library equation, 5-43
 - asynchronous processing with process
 - statement, 6-141
 - asynchronous task initiation, 5-2
 - asynchronous tasks, 5-2
 - AT specification, example, 2-7
 - AT <hostname constant>, 3-2
 - attribute lists, 6-77
 - authmode attribute, 6-80
 - autounload attribute, 6-80, 6-118
 - archive tape volume, 6-25
 - in restore statement, 6-152
 - available file attribute, inquiring file
 - residence, 7-4
 - AX examples in a run statement, 6-156
- ## B
- backup option
 - in copy statement, 6-65
 - backup utility, 6-136
 - banner attribute
 - in alter statement, 6-9
 - basic constructs, 8-1
 - becomeowner option
 - in move statement, 6-129
 - in copy statement, 6-66
 - begin job construct, 3-2
 - bind option
 - compile or bind statement, 6-55
 - bind statement, 5-1, 6-47
 - in process statement, 6-141
 - example, 6-47
 - binder title, 6-47
 - in bind statement, 6-47
 - blocksize attribute, 6-81
 - archive tape volume attributes, 6-26
 - Boolean assignment example, 5-9
 - Boolean assignment statement, 6-45
 - Boolean constant, 8-3
 - in Boolean constant primary, 7-25

- Boolean constant expression, 4–2, 7–25
 - in Boolean constant primary, 7–25
 - in Boolean declaration, 4–3
- Boolean constant identifier, 4–2, 8–3
 - in Boolean constant primary, 7–25
- Boolean constant primary, 7–25
 - in Boolean constant expression, 7–25
- Boolean declaration, 4–1, 4–3
- Boolean expression, 7–1
 - in Boolean assignment statement, 6–45
 - in do statement, 6–120
 - in if statement, 6–121
 - in named parameter list, 6–162
 - in positional parameter list, 6–162
 - in print attribute phrase, 6–136
 - in print modifier phrase, 6–137
 - in run parameter list, 6–155
 - in security specification, 6–160
 - in source volume attributes, 6–148
 - in subroutine invocation statements, 6–167
 - in task attribute assignment, 5–8
 - in volume attribute list, 6–174
 - in while statement, 6–187
- Boolean file attribute, 5–37
 - in Boolean file attribute primary, 7–4
 - primary, 7–4
- Boolean formal parameter, 3–5
 - in named parameter list, 6–162
- Boolean format parameter, 3–5
- Boolean identifier, 8–3
 - in Boolean assignment statement, 6–45
 - in subroutine parameters, 4–7
 - in Boolean declaration, 4–3
- Boolean operators
 - AND, 7–1
 - EQV, 7–2
 - IMP, 7–2
 - NOT, 7–1
 - OR, 7–1
- Boolean parameter declaration, 3–4
- Boolean primary, 7–2
- Boolean print attribute
 - in print attribute phrase, 6–136
- Boolean print modifier
 - in print modifier phrase, 6–137
- Boolean relational operators, 7–1
- Boolean task attribute
 - in Boolean task attribute primary, 7–4
 - primary, 7–4
 - in task attribute assignment, 5–8

- in wait specification, 6–184
 - wait statement options, 6–185
- Boolean variables, 4–3
- buffer underrun, 6–94

C

- call-by-reference parameters, 4–9
- call-by-value parameters, 4–9
- CANDE
 - add command, 2–3
 - start command, 2–2
 - WFL command, 2–1
 - commands, 2–3
 - copy statement, 2–3
 - initiating WFL jobs, limitations, 2–2
 - make command, 2–2
- case constant expression, 6–48
 - in case statement, 6–48
- case expression, 6–48
 - in case statement, 6–48
- case statement, 1–7, 6–48
 - example, 6–48
- catalog option in copy statement, 6–66
- catalog statement
 - add, 6–49
 - delete, 6–49
 - example, 6–50
 - purge, 6–49
 - serialno option, 6–49
- cataloging
 - statement, 6–4
 - volume library, 6–175
- cataloging statements
 - catalog, 6–49
 - volume, 6–175
- ccsversion attribute
 - in alter statement, 6–9
- cdcopies attribute
 - archive CD volume attributes, 6–34
- CD-ROMs and copy statement, 6–61, 6–64
- change statement, 1–15, 6–50
 - entering individually from ODT, 2–5
 - example, 1–15
 - example, 6–52
- changing
 - a password, 6–5
 - file attributes, 5–31
- character elements, 8–2
- character set, 7–16, 8–1
 - in head-tail function, 7–16

- in head-tail constant function, 7-27
- checkpoint, restarting a job at, 6-147
- choosing a compiler, 6-54
- class job attribute, 3-10
- class specification, 3-10
- clauses, onto, 6-75
- closing files, 1-13
 - example, 1-14
- COBOL74, initiating WFL jobs, 2-7
- COBOL85, initiating WFL jobs, 2-7
- code control option, 9-1
- code core, 5-12
 - in core assignment, 5-12
- comment delimiter, percent sign (, 3-2), 3-2
- comments, 1-17
 - jobs, 1-17
- communication statement, 1-15, 6-3
 - display, 6-3
 - instruction, 6-3
- compact discs and copy statement, 6-64
- compare files, 6-21
- compare option
 - in archive statement, 6-21
 - in copy statement, 6-72
 - in move statement, 6-129
 - in restore statement, 6-150
 - in copy statement, 6-67
- compile or bind statement, 6-53, 6-54
 - compiler name, 6-54
 - example, 6-54, 6-55, 6-58, 6-59
 - go option, 6-56
 - library option, 6-56
 - local data specifications, 6-58
 - object code file disposition, 6-55
 - object code file title, 6-54
 - syntax option, 6-56
 - bind option, 6-55
 - library go option, 6-56
- compile statement, 5-1
 - example, 1-4, 6-56
 - in process statement, 6-141
 - task variables, 6-56
- COMPILEDOK task state, 7-5
- compiler data specification
 - in compiler task equation list, 6-57
- compiler name, 6-54
 - in compile or bind statement, 6-53
 - in compiler task equation list, 6-57
- compiler task equation
 - overriding at run time, example, 6-58
- compiler task equation list, 6-57
 - example, 6-58
 - file equations, 6-57
 - library equations, 6-57
 - task attributes, 6-57
 - in bind statement, 6-47
 - in compile or bind statement, 6-53
 - in database equations, 6-57
- compiler title, 6-53
- compiling WFL jobs for syntax, 3-7
- COMPLETED task state, 7-5
- COMPLETEDOK task state, 7-5
- complex task attribute assignment, 5-11
 - accesscode, 5-11
 - core, 5-12
 - option, 5-18
 - resource, 5-21
 - currentdirectory, 5-13
 - datapath, 5-14
 - executepath, 5-15
 - family, 5-16
 - in task attribute assignment, 5-8
 - printdefaults assignment, 5-19
 - usercode, 5-22
- complex task attributes
 - interrogating value of, 2-9, 5-28, 5-29, 6-11, 6-12, 6-13, 6-18, 6-50, 6-51, 6-74, 6-127, 6-129, 6-145, 6-161, 7-15
- compound statement, 6-2, 6-59
 - example, 6-59
- COMPRESS option
 - with wrap statement, 6-189
- compressioncontrol attribute, 6-88
 - archive tape volume attributes, 6-26
- compressionrequested attribute
 - archive tape volume attributes, 6-26
- concatenation operators, 7-13
- constant declaration, 4-2
- constant declaration element, 4-2
- constant expressions, 7-24
- constant identifiers, 4-2
 - declaring, 3-5
 - example, 4-3
- constants, 8-3
 - declaring, 4-3
- context-sensitive words, B-2
- control options, 9-1
- controller, 2-5
- copy * = AS, 6-100
- copy = AS, 6-100

- copy file, 6-73
- copy file transfer services, 6-106
- copy files, 6-17
- copy from group, 6-62
- copy options
 - backup, 6-65
 - becomeowner, 6-66
 - catalog, 6-66
 - compare, 6-67
 - dsonerror, 6-67
 - fromstart, 6-68
 - propagate, 6-68
 - remove, 6-69
 - report, 6-69
 - select, 6-69
 - usepath, 6-70
 - verify, 6-71
 - waitonerror, 6-72
 - skipexclusive, 6-70
- copy or add statement, 6-60, 6-61
 - example, 6-103
- copy request
 - in add statement, 6-6
 - in copy or add statement, 6-60
 - onto clause, 6-75
 - in copy or add statement, 6-60
- copy statement, 1-14, 5-1
 - CD-ROMs, 6-61
 - example, 1-15
 - file attributes, 6-80
 - file transfer services, 6-106
 - format type, 6-63
 - in CANDE, 2-3
 - in MARC, 2-6
 - in process statement, 6-141
 - library maintenance, 6-63
 - multiple copies by multiple destination
 - volumes, 6-73
 - restarting interrupted file transfers, 6-115
 - restrictions, 6-108
 - special cases, 6-100
 - specifying copy options, 6-65
 - specifying file attributes, 6-60
 - transfer services, 6-106
 - CD-ROMs, 6-64
 - copying FIFOs, 6-63
 - example, 6-87
 - file attributes, 6-87
 - FTP transform attributes, 6-113
 - host services, 6-110
 - NFT transfer, 6-109

- copying
 - CD-ROM images, 6-61
 - files, 6-61, 6-63
- copying files, 6-17, 8-13
 - from tape, 6-95
 - KEYEDIOII, 6-64
 - locally, library maintenance program, 6-63
 - remote hosts, file transfer service, 6-63
 - with usercodes, 6-96
 - without usercodes, 6-96
- core assignment, 5-12
 - in complex task attribute assignment,
 - 5-11
- core task attribute, assigning value to, 5-12
- create libmaintdir statement, 6-116
 - autounload attribute, 6-118
 - cycle attribute, 6-118
 - familyowner attribute, 6-118
 - file attributes, 6-117
 - options, 6-117
 - serialno attribute, 6-117
 - version attribute, 6-118
- creating permanent directories, 6-125
- crunch statement, 1-13, 6-119
- currentdirectory assignment, 5-13
 - example, 5-13
 - in complex task attribute assignment,
 - 5-11
- cycle attribute, 6-82, 6-118
 - in restore statement, 6-152

D

- data core, 5-12
 - in core assignment, 5-12
- DATA, data type, 4-10
- data decks, 4-9
- data images, 4-9
 - in global data specification, 4-9
 - in local data specification, 5-44
- data processing, 1-7
- data specifications, 1-6
 - example, 1-6
 - global, 4-9
- data types
 - DATA, 4-10
 - EBCDIC, 4-10
- database equation, 5-44
 - in modify statement, 6-126
 - in task equation list, 5-3
 - example, 5-44

- in compiler task equation list, 6-57
- database equations compiler task equation list, 6-57
- database name, 5-44
 - in database equation, 5-44
- database title, 5-44
 - in database equation, 5-44
- datapath assignment, 5-14, 6-51 , 6-145
- date, 3-11
- date information, 7-20
- day interval, 3-12
- DCALGOL, initiating WFL jobs, 2-7
- dd in date, 3-11
- decimal function, 7-8
 - in integer primary, 7-8
- declaration list, 3-15
 - example, 3-15
 - in subroutine block, 4-7
- declarations, 4-1
 - Boolean variables, 4-3
 - global data specifications, 4-9
 - real variables, 4-4
 - string variables, 4-5
 - subroutines, 4-7
 - task variables, 4-6
 - constants, 4-2
 - file variables, 4-5
 - integer variables, 4-4
- default clause, 3-6
- delaying job initiation, 3-13
- density attribute, 6-88
 - archive tape volume attributes, 6-27
- density cdcopies attribute
 - archive CD volume attributes, 6-34
- destination, 6-170, 6-189
 - in unwrap volume, 6-170
 - in wrap volume, 6-188
- destination volume file, 6-73
- device kind assignment, 5-35
- device kind attribute
 - example, 5-36
- device mnemonic, 5-36
 - in device kind assignment, 5-35
- digit, 8-2
 - in date, 3-11
 - in day interval, 3-12
 - in dd, 3-12
 - in identifier, 8-2
 - in integer constant, 8-3
 - in real constant, 8-3
 - in sequence number, 9-2
 - in source volume name, 6-148
 - in time, 3-11
 - in time interval, 3-11
 - in yy, 3-12
 - in yyyy, 3-12
 - in mm, 3-12
- digital signatures
 - wrapping files with, 6-191
- directories, 8-7
- directory, 5-14, 5-15
 - in path specification, 5-14, 5-15
- directory name
 - in archive restore statement, 6-39
 - in directory selection, 6-148
 - in mkdir statement, 6-125
 - in remove from group, 6-39
 - in remove list, 6-39
 - in unwrap file, 6-170
 - in wrap file, 6-188
- directory name constant
 - in directory title constant, 8-8
- directory selection, 6-148
 - in restore statement, 6-148
- directory title
 - in print specification, 6-136
- directory title constant, 8-8
- discontinuing a task, 6-4
- disk, transferring files to, 6-109
- display statement, 1-8, 1-16 , 6-119
 - example, 1-16
- displaying messages, 6-119
- distributed systems services (DSS), 2-6
 - in file transfers, 6-106
- DIV
 - as a real operator, 7-10
 - as an integer operator, 7-7
- do statement, 1-7, 6-120
 - example, 1-9
- dollar sign (\$)
 - in compiler control options, 9-1
- DRC option
 - archive rollout statement, 6-42
- drop constant function, 7-27
- drop function, 7-19
- dsonerror option
 - in archive statement, 6-21
 - in copy statement, 6-67
 - in create libmaintdir statement, 6-117
 - in move statement, 6-130
 - in restore statement, 6-150
 - in unwrap statement, 6-169

DSOERROR option
 with wrap statement, 6-189
DSS, 2-6
dummy files, job execution, 2-12

E

EBCDIC
 data type, 4-10
 character set, 8-1
encrypt attribute, 6-88
 archive CD volume attributes, 6-34
 archive tape volume attributes, 6-27
end job construct, 3-2
ending identifiers, subroutines, 1-17
equations, file, 5-30
EQV, as a Boolean operator, 7-2
errorlimit control option, 9-2
event file attributes, 5-37
examples
 accept function, 7-16
 add statement, 6-105
 abort statement, 1-11
 case statement, 1-8
 accept function, 1-16
 compile statement, 1-4
 constant expressions, 7-24
 copy statement, 6-87
 add statement, 1-15
 do statement, 6-120
 drop function, 7-19
 go statement, 6-121
 head function, 7-17
 hex function, 7-11
 identifiers, 8-3
 if statement, 6-122
 initialize statement, 6-123
 instruction statement, 6-124
 integer task attribute primary, 7-10
 length function, 7-9
 list1 control option, 9-5
 lock statement, 6-124
 name constant, 8-6
 octal function, 7-12
 real constants, 8-4
 return statement, 1-10
 simple name constant, 8-6
 string constants, 8-4
 string file attribute primary, 7-15
 string function, 7-18
 system function, 7-18
 tail function, 7-17
 take function, 7-19
 task attributes, 1-5
 task file attribute primary, 7-15
 timedate function, 7-20
 translate functions, 7-23
 while statement, 6-187
change statement, 1-15
closing files, 1-14
constant identifiers, 4-3
copy or add statement, 6-103
copy statement, 1-15
crunch statement, 6-119
data specification, 1-6
decimal function, 7-9
display statement, 1-16
do statement, 1-9
errorlimit control option, 9-2
file mnemonic primary, 7-24
if statement, 1-7
include control option, 9-3
integer constants, 8-4
integer function, 7-9
job tile, 3-4
list control option, 9-4
log statement, 6-125
modify statement, 6-128
newsegment control option, 9-6
null statement, 6-131
on statement, 6-133
open statement, 6-135
PB statement, 6-136
print statement, 6-140
process statement, 6-142
purge statement, 6-143
release statement, 6-143
remove statement, 1-15, 6-145
rerun statement, 6-147
restore statement, 6-153, 6-154
rewind statement, 6-155
run statement, 1-4, 6-156
security statement, 1-15, 6-161
start statement, 6-164
starttime specification, 3-13
stop statement, 6-166
string expressions, 7-14
subroutine invocation statement, 6-168
subroutines, 1-9, 1-17
unwrap statement, 6-172
user statement, 1-12, 6-174
volume statement, 6-183

- wait statement, 1–12, 6–186
- warnsuppress control option, 9–7
- wrap statement, 6–191
- xref control option, 9–7
- xreffiles option, 9–8
- executepath assignment, 5–15
- expressions, 7–1
 - constant, 7–24
- extdelimiter attribute
 - in alter statement, 6–9
- extmode attribute
 - in alter statement, 6–9

F

- family assignment, 5–16
 - in complex task attribute assignment, 5–11
 - example, 5–17
- family name
 - in alternate family name, 5–17
 - in bind statement, 6–47
 - in destination, 6–170, 6–189
 - in directory selection, 6–148
 - in mkdir statement, 6–125
 - in move statement, 6–128
 - in path specification, 5–14, 5–15
 - in primary family name, 5–17
 - in remove from group, 6–39, 6–144
 - in security from group, 6–159
 - in source, 6–170, 6–188
 - in volume name, 8–6
 - in target family name, 5–17
- family name constant
 - in directory title constant, 8–8
- family name file, 6–74
- family specification, 5–16
 - in family assignment, 5–16
- family specifications, 6–18
 - in archive subsystem statements, 6–18
- family substitution, 6–18
 - in archive subsystem statements, 6–18
- family task attribute
 - assigning value to, 5–16
 - inquiring value of, 5–29
- familyindex attribute, 6–89
 - archive disk volume, 6–23
- familyindex option
 - in move statement, 6–130
- familyowner attribute, 6–82, 6–118
 - archive tape volume attributes, 6–27

- in restore statement, 6–152
 - in volume add statement, 6–180
- fetch job attribute, 3–11
- fetch specification, 1–16
 - example, 3–11
- FIFOs, restrictions when copying files, 6–63
- file assignment statement, 6–45, 6–46
- file attribute assignment
 - in file assignment statement, 6–46
 - in file declaration, 4–5
 - in file equation, 5–30
- file attribute inquiry, 5–28
- file attributes, 5–37
 - authmode, 6–80
 - autounload, 6–80
 - Boolean, 5–37
 - changing, 5–31
 - compressioncontrol, 6–88
 - copy statement, 6–87
 - density, 6–88
 - encrypt, 6–88
 - familyowner, 6–82
 - file name, 5–37
 - hostname, 6–83
 - inquiring value of, 2–13, 4–5 , 5–34 , 5–40 , 7–7 , 7–15
 - ipaddress, 6–83
 - kind, 6–83
 - libmaintdir, 6–83, 6–92
 - locatecapable, 6–84
 - lockedfile, 6–92
 - mnemonic, 5–37
 - name, 5–38
 - offsite, 6–85
 - scratchpool, 6–85
 - securityuse, 6–94
 - sensitivedata, 6–95
 - singleunit, 6–95
 - specifying in copy statement, 6–60
 - assigning, example, 5–32
 - unitno, 6–86
 - usercode, 6–86
 - using to check file residence, 7–3
 - version, 6–95
 - windowssize, 6–86
 - yourusercode, 6–86
- blocksize, 6–81
- copy statement, 6–80
- cycle, 6–82
- familyindex, 6–89
- implicit assignments, 5–39

- integer, 5–37
- keywords, B–3
- libmaintappend, 6–89
- long file name, 5–37
- long title file, 5–37
- multivolume, 6–92
- myipaddress, 6–85
- real, 5–38, 5–39
- savefactor, 6–94
- securityguard, 6–94
- securitytype, 6–94
- serialno, 6–85
- string, 5–38
- title, 5–38
- usecatalog, 6–95
- file declaration, 4–1, 4–5
- file equations, 5–30
 - in modify statement, 6–126
 - in task assignment statement, 6–46
 - in task declaration, 4–6
 - in task equation list, 5–3
 - in compiler task equation list, 6–57
 - overriding, 5–32
 - resolving repeated, 5–32
- file handling statements, 1–13, 6–3
 - lock, 6–4
 - release, 6–4
 - change purge, 6–4
 - rewind, 6–4
- file identifier, 8–3
 - in Boolean file attribute primary, 7–4
 - in crunch statement, 6–119
 - in file assignment statement, 6–46
 - in file declaration, 4–5
 - in file mnemonic comparison, 7–6
 - in file residence inquiry, 7–3
 - in global file assignment, 5–33
 - in lock statement, 6–124
 - in mnemonic primary, 7–23
 - in open statement, 6–135
 - in purge statement, 6–142
 - in real file attribute primary, 7–12
 - in release statement, 6–143
 - in subroutine invocation statements, 6–167
 - in subroutine parameters, 4–7
 - in integer file attribute primary, 7–9
- file management, 1–14
- file management statement, 6–4
 - alter, 6–4
 - archive release, 6–4
 - change, 6–4
 - modify, 6–4
 - archive purge, 6–4
 - security, 6–4
 - mkdir, 6–4
 - print, 6–4
 - remove, 6–4
 - volume, 6–4
- file mnemonic
 - in mnemonic primary, 7–23
- file mnemonic comparison, 7–6
- file mnemonic primary
 - in file mnemonic comparison, 7–6
 - in print attribute phrase, 6–136
 - in volume attribute list, 6–174
- file name
 - in archive restore statement, 6–39
 - in file selection, 6–148
 - in global data specification, 4–9
 - in remove from group, 6–39
 - in remove list, 6–39
 - in task attribute assignment, 5–8
 - in unwrap file, 6–170
 - in unwrap request, 6–169
 - in wrap file, 6–188
 - in wrap request, 6–188
- file name constant, 8–8
 - in local data specification, 5–44
- file name file attribute, 5–37
 - in string file attribute primary, 7–14
- file name task attribute
 - in string task attribute primary, 7–15
 - in task attribute assignment, 5–8
- file names, 8–4, 8–7
- file residence
 - using file attributes to check, 7–3
- file residence inquiry, 7–3
- file security
 - changing, 6–161
 - attributes, 6–161
- file selection, 6–148
 - in restore statement, 6–148
- file specification, 6–159
 - in security statement, 6–159
- file title
 - in binder title, 6–47
 - in compiler title, 6–53
 - in database title, 5–44
 - in file equation, 5–30
 - in file residence inquiry, 7–3
 - in print attribute phrase, 6–136

- in print specification, 6-136
- in security specification, 6-160
- in start statement, 6-162
- in task attribute assignment, 5-8
- in traditional security specification, 6-160
- in volume attribute list, 6-174
- in wait specification, 6-184

file title constant

- in include control option, 9-2

File Transfer Protocol (FTP)

- in WFL, 6-106
- restrictions, 6-112
- transferring files with, 6-112
- transform attributes for copying files, 6-113

file transfer restrictions, 6-108

file transfer services, 6-106

- File Transfer Protocol (FTP), 6-112
- host services file transfer, 6-110
- Native File Transfer (NFT), 6-109

file variables, 4-5

filecards, 5-31

filekind attribute

- in alter statement, 6-10

files

- between hosts, 6-106
- changing names, 6-50
- comparing, 6-21
- copying, 6-17
- declaring, 4-5
- example, 1-14
- inquiring the residence of, 7-3
- merging, 6-17
- removing, 6-17, 6-144
- restoring, 6-17
- transferring to disk, 6-109
- closing, 1-13
- copying, 6-63
- copying multiple, 8-13
- copying with usercodes, 6-96
- copying without usercodes, 6-96
- nonresident, 5-42
- opening, 1-13
- printing portions of, 6-139
- restrictions, 6-108

flow-of-control statements, 1-6, 6-2

- case, 6-2
- do, 6-2
- go, 6-2
- if, 6-2
- while, 6-2

formid attribute

- in alter statement, 6-10

free formatting

- job, 1-17

fromstart option

- in copy statement, 6-68

FROMSTART option

- in copy statement, 6-115

fromstart option

- in copy statement, 6-72

FTP file transfers, 6-112

FTP statement, 6-112

FTP transform

- append attribute, 6-113
- ftpstructure attribute, 6-113
- ftpstype attribute, 6-113
- ftpsite attribute, 6-113

FTP transform attributes

- copy statement, 6-113

ftpsite attribute

- in FTP transform, 6-113

ftpstructure attribute

- in FTP transform, 6-113

ftpstype attribute

- in FTP transform, 6-113

G

generations, 6-17

- in archive subsystem statements, 6-17

global data specification, 4-1, 4-9

- example, 4-11

global file assignment, 5-33

- example, 5-33
- in file equation, 5-30

global file declaration, 4-5

global variables, 1-9, 4-1, 4-8

go option

- compile or bind statement, 6-56

go statement, 6-120

GO statement, 6-121

group attribute

- in alter statement, 6-10
- in volume add statement, 6-180
- in volume statement, 6-177

groupr attribute

- in alter statement, 6-10
- in volume add statement, 6-180
- in volume statement, 6-178

grouprwx attribute

- in alter statement, 6-10

- in volume add statement, 6-181
- in volume statement, 6-178
- groupw attribute
 - in alter statement, 6-10
 - in volume statement, 6-178
- groupx attribute
 - in alter statement, 6-10
 - in volume add statement, 6-180
 - in volume statement, 6-178
- guardowner attribute
 - in alter statement, 6-11
 - in volume add statement, 6-181
 - in volume statement, 6-178

H

- halt/load
 - restarting a job after, 2-9
 - values of variables after, 2-9
- head-tail constant functions, 7-27
 - in string constant primary, 7-27
- head-tail functions, 7-16
 - in string primary, 7-14
- hex function, 7-11
 - in real primary, 7-11
- HI system command, 6-20, 6-43
- historycause attribute, 2-9
- historytype attribute, 2-9
- host name unknown, 3-3
- host services
 - in copy statement, 6-110
 - file transfer restrictions, 6-110
- Host Services file transfer
 - in WFL, 6-106
- host specification, 3-3
- hostname attribute, 6-83
- hosts, copying files between, 6-106
- hyphen, 8-2
 - in source volume name, 6-148

I

- I, statement separator, 8-1
- identifier, 8-2
 - in database name, 5-44
 - in global file assignment, 5-33
- identifying transfer services, 6-106
- if statement, 1-7, 6-121
 - example, 1-7
- IMP, as a Boolean operator, 7-2
- include control option, 9-2

- initialize statement, 6-122
- initiating WFL jobs, 2-1
- input sources, 2-1
 - ALGOL programs, 2-7
 - CANDE, 2-1
 - COBOL74 programs, 2-7
 - COBOL85 programs, 2-7
 - DCALGOL programs, 2-7
 - load control tapes, 2-8
 - MARC, 2-5
 - ODTs, 2-4
 - RPG programs, 2-7
 - start statement, 2-4
- input/output, 5-30
- instruction statement, 1-16, 6-123
- instruction text
 - in instruction statement, 6-123
- integer assignment
 - example, 5-10
- integer assignment statement, 6-45
- integer constant, 8-3
 - in errorlimit control option, 9-2
 - in integer primary, 7-8
- integer constant expression, 4-2, 4-4 , 7-25
 - in take-drop constant function, 7-27
 - in arithmetic constant comparison, 7-25
 - in case constant expression, 6-48
 - in resource assignment, 5-21
 - in string constant function, 7-27
- integer constant identifier, 4-2, 8-3
 - in integer primary, 7-8
- integer constant primary
 - in instruction statement, 6-123
 - in integer constant expression, 7-25
 - in real constant primary, 7-26
 - in rerun statement, 6-147
- integer declaration, 4-1, 4-4
- integer expression, 7-7
 - in archive rollout statement, 6-41
 - in arithmetic comparison, 7-5
 - in case expression, 6-48
 - in code core, 5-12
 - in data core, 5-12
 - in integer assignment statement, 6-45
 - in integer primary, 7-8
 - in named parameter list, 6-162
 - in positional parameter list, 6-162
 - in print attribute phrase, 6-136
 - in print modifier phrase, 6-137
 - in run parameter list, 6-155
 - in security specification, 6-160

- in serial number, 5–36
- in source volume attributes, 6–148
- in string function, 7–17
- in subroutine invocation statements, 6–167
- in take-drop functions, 7–19
- in task attribute assignment, 5–8
- in total core, 5–12
- in volume attribute list, 6–174
- in archive disk volume attribute list, 6–23
- integer file attribute, 5–37
 - in integer file attribute primary, 7–9
- integer file attribute primary, 7–9
 - in integer primary, 7–8
- integer formal parameter, 3–5
 - in named parameter list, 6–162
- integer function, 7–9
 - in integer primary, 7–8
- integer identifier, 8–3
 - in integer assignment statement, 6–45
 - in integer declaration, 4–4
 - in integer primary, 7–8
 - in subroutine parameters, 4–7
- integer operators, 7–7
- integer parameter declaration, 3–4
- integer primary, 7–8
 - in integer expression, 7–7
 - in real primary, 7–11
- integer print attribute
 - in print attribute phrase, 6–136
- integer print modifier
 - in print modifier phrase, 6–137
- integer task attribute
 - in integer task attribute primary, 7–10
 - in simple task relation, 6–184
 - in task attribute assignment, 5–8
- integer task attribute primary, 7–10
 - in integer primary, 7–8
- integer variables, 4–4
- interpretively executing statements, 2–1
- inname
 - in file equation, 5–30
 - in library equation, 5–42
 - example, 5–31
- INUSE task state, 7–5
- invalid and valid characters, 8–1
 - problems in text entry, 2–2
- invalid character, 8–1
- I/O
 - file declaration, 4–5
 - global data specification, 4–9

- remote files, 5–34
 - file equation, 5–30
- ipaddress attribute, 6–83

J

- job, 3–1
 - after a halt/load, 2–9
 - at a checkpoint, 6–147
 - changing the usercode, 6–173
 - comments, 1–17
 - continuing after a task fails, 2–9
 - delaying initiation, 3–13
 - free formatting, 1–17
 - initiation, 2–1
 - instructions to operators, 6–123
 - passing string parameters, 8–12
 - samples, A–1
 - storing in disk file, 2–2
 - syntax, 3–1
 - terminating, 6–166
 - attributes associated with, 3–8
 - comments, 1–17
 - compiling for syntax, 3–7
 - disposition, 3–7
 - format, 1–17, 3–2
 - from CANDE, 2–1
 - from MARC, 2–5
 - from user programs, 2–7
 - parameter list, 3–5
 - running on a remote host, 2–6
- job attribute list, 3–8
- job attributes
 - assigning, 3–8
 - example, 3–9
 - fetch specification, 3–11
- job disposition, 3–7
- job format, 3–2
- job instructions, supplying to operators, 6–123
- job parameter list, 3–4
- job title, 3–3
 - example, 3–4

K

- KEYEDIOII files, copying, 6–64
- keywords, B–3
- kind attribute, 5–36, 6–83
 - archive disk volume, 6–23
 - archive tape volume attributes, 6–28

- creating multiple copies, 6–88
- packet write recording, 6–93
- restrictions, 6–99

L

- label attribute
 - in alter statement, 6–11
- label identifier, 8–3
 - for statements, 3–16
 - in go statement, 6–120
- laissezfile, CANDE option, 5–35
- length function, 7–9
 - in integer primary, 7–8
- letter, 8–2
 - in identifier, 8–2
 - in source volume name, 6–148
- libmaintappend attribute, 6–89
 - archive tape volume attributes, 6–28
 - example, 6–29, 6–91
- libmaintmdir attribute, 6–83, 6–92
 - archive tape volume attributes, 6–30
 - restore statement, 6–152
- library attribute assignment, 5–42
 - in library equation, 5–42
- library attribute value
 - in library attribute assignment, 5–42
- library equations, 5–42
 - in archive task equation list, 6–36
 - in modify statement, 6–126
 - in task equation list, 5–3
 - overriding, 5–43
 - resolving repeated, 5–43
 - example, 5–42
 - in compiler task equation list, 6–57
- library go option
 - compile or bind statement, 6–56
- library maintenance, 6–63
 - restore statement, 6–150
- library maintenance statements, 1–14
 - copy statement, 6–63
- library option
 - compile or bind statement, 6–56
- list1 control option, 9–5
- list control option, 9–4
- load control tapes, initiating WFL jobs from, 2–8
- local data specification, 5–44
 - example, 5–45
 - in compiler task equation list, 6–57
 - in task equation list, 5–3

- local data specifications
 - compile or bind statement, 6–58
- local variables, 1–9, 4–1, 4–8
- locatecapable attribute, 6–84, 6–152
 - archive tape volume attributes, 6–30
- lock statement, 1–13, 6–124
- lockedfile attribute, 6–92
 - in alter statement, 6–11
- logalyzer options
 - in log statement, 6–125
- logalyzer utility, initiating, 6–125
- log statement, 5–2, 6–125
 - in process statement, 6–141
- log-on account, 6–79
- log-on info, 6–79
- log-on password, 6–79
 - copy statement, 6–61
- log-on usercode, 6–79
- long directory name
 - in move statement, 6–128
 - in remove from group, 6–144
 - in security from group, 6–159
- long directory name constant
 - in archive purge statement, 6–38
- long directory title
 - in alter statement, 6–7
 - in remove list, 6–144
 - in security list, 6–159
- long file name
 - in remove from group, 6–144
 - in security from group, 6–159
 - in move statement, 6–128
- long file name constant
 - in archive purge statement, 6–38
- long file name file attribute, 5–37
- long file names, 8–4, 8–7
 - disabled, 8–7
- long file title
 - in alter statement, 6–7
 - in remove list, 6–144
 - in security list, 6–159
 - in alter statement, 6–7
- long node name constant
 - in wrap password, 8–6
- long title file file attribute, 5–37
- LOWERCASE function, 7–22, 7–27

M

- maintenance, library, 6–63

- MARC
 - add command, 2–6
 - initiating WFL jobs from, 2–5
 - start command, 2–6
 - copy command, 2–6
 - matchonlyserialno attribute
 - in volume add statement, 6–181
 - maximum blocksize
 - tape drives, 6–81
 - MCP
 - compatible release level with WFLSupport, 1–3
 - merge files, 6–17
 - messages, displaying, 6–119
 - minus sign (-)
 - as a real operator, 7–10
 - as an integer operator, 7–7
 - mkdir statement, 6–125
 - MLS (MultiLingual System), 1–3
 - mm in date, 3–11
 - mnemonic
 - in print modifier phrase, 6–137
 - mnemonic assignment, example, 5–10
 - mnemonic file attribute, 5–37
 - in mnemonic primary, 7–23
 - in file mnemonic comparison, 7–6
 - in string file attribute primary, 7–14
 - mnemonic primary, 7–23
 - mnemonic print attribute
 - in print attribute phrase, 6–136
 - mnemonic print modifier
 - in print modifier phrase, 6–137
 - mnemonic task attribute
 - in string task attribute primary, 7–15
 - in task attribute assignment, 5–8
 - in task mnemonic comparison, 7–7
 - mnemonic task identifier
 - in task mnemonic primary, 7–23
 - MOD
 - as a real operator, 7–10
 - as an integer operator, 7–7
 - modify statement, 6–126
 - move statement, 6–128
 - becomeowner, 6–129
 - compare, 6–129
 - dsonerror, 6–130
 - familyindex, 6–130
 - options, 6–129
 - propagate, 6–130
 - report, 6–130
 - select, 6–130
 - skipexclusive, 6–130
 - verify, 6–130
 - waitonerror, 6–131
 - MultiLingual System (MLS), 1–3
 - multivolume attribute, 6–92
 - archive CD volume attributes, 6–35
 - myipaddress attribute, 6–85
 - myjob predeclared task variable, 5–29
 - example, 5–30
 - myself predeclared task variable, 5–29
 - example, 5–30
- N**
- name
 - in scratch pool name, 6–25
 - in task attribute assignment, 5–8
 - name constant
 - in file name constant, 8–8
 - in new accesscode password, 6–5
 - in old password, 6–135
 - in security specification, 6–160
 - in volume attribute list, 6–174
 - in new password, 6–135
 - name file attribute, 5–38
 - in string file attribute primary, 7–14
 - name task attribute
 - in task attribute assignment, 5–8
 - in string task attribute primary, 7–15
 - named parameter list, 6–162
 - in start parameter list, 6–162
 - names, 8–2
 - naming the object code file, 6–54
 - Native File Transfer (NFT)
 - recovery files, 6–109
 - in WFL, 6–106
 - restrictions, 6–109
 - nesting
 - statements, 6–1
 - subroutines, 4–8
 - new password, 6–135
 - in access statement, 6–5
 - new password in password statement, 6–135
 - newsegment control option, 9–5
 - NFT, 6–106
 - NFT file transfers
 - in copy statement, 6–109
 - nonquote EBCDIC character, 8–2
 - nonresident files, 5–42
 - nonsingle quote EBCDIC character, 8–2

NOT, as a Boolean operator, 7-1

note attribute

in alter statement, 6-11

null character, 3-3

null statement, 6-2, 6-131

number

in move statement, 6-128

O

object code file disposition

compile or bind statement, 6-55

object code file title

compile or bind statement, 6-54

in bind statement, 6-47

in compile or bind statement, 6-53

in modify statement, 6-126

in run statement, 6-155

octal function, 7-11

OCTAL function, 7-12

octal function

in real primary, 7-11

ODT, 2-4

offsite

archive tape volume attributes, 6-31

offsite attribute, 6-85

archive CD volume attributes, 6-35

OK, wait statement options, 6-185

old password, 6-135

old password in password statement, 6-135

ON, as a string operator, 7-13

on restart statement

job interruption by a halt/load, 2-12

on statement, 6-132

on taskfault statement, 2-9

onto clause, 6-75

of copy statement, 6-75

open file attribute, implicit setting of, 5-39

open statement, 1-13, 6-135

opening files, 1-13

operator display terminal (ODT), 2-4, 2-5

initiating WFL jobs from, 2-4

option assignment, 5-18

in complex task attribute assignment,
5-11

option statement, 6-116

option task attribute

assigning values to, 5-18

inquiring value of, 5-29

options

in archive statement, 6-21

in create libmaintdir statement, 6-116

options statement, 6-116

OR, as a Boolean operator, 7-1

other attribute

in volume statement, 6-178

otherrr attribute

in alter statement, 6-11

in volume add statement, 6-181

otherrrwx attribute

in alter statement, 6-12

in volume add statement, 6-181

in volume statement, 6-178

otherw attribute

in alter statement, 6-12

in volume statement, 6-178

volume add statement, 6-181

otherx attribute

in alter statement, 6-12

in volume add statement, 6-181

in volume statement, 6-178

overriding

file equations, 5-32

library equations, 5-43

example, 6-58

owner attribute

in alter statement, 6-12

ownerr attribute

in alter statement, 6-12

in volume add statement, 6-181

in volume statement, 6-178

ownerrwx attribute

in alter statement, 6-13

in volume add statement, 6-182

in volume statement, 6-179

ownerw attribute

in alter statement, 6-12

in volume add statement, 6-181

in volume statement, 6-178

ownerx attribute

in alter statement, 6-13

in volume statement, 6-179

volume add statement, 6-181

P

packet write recording, 6-93

packetwrite attribute

archive CD volume attributes, 6-35

pagecomp attribute

in alter statement, 6-13

parameters
 call by value, 4–9
 call by reference, 4–9
 start statement, 6–163
 subroutine, 4–8

password
 aging feature, 5–22
 changing, 6–5
 in accesscode assignment, 5–11
 in usercode assignment, 5–22

password-aging feature, 5–12, 5–22

password name constant
 in user statement, 6–173

password statement, 6–135

patching, A–2

path specification, 5–14, 5–15
 in datapath assignment, 5–14
 in executepath assignment, 5–15

PB statement, 5–2, 6–136
 in process statement, 6–141

percent sign (%, 3–2), as a comment delimiter, 3–2

permanent directories
 creating, 6–125

permanentlyowned attribute
 in volume add statement, 6–182

plus sign (+)
 as a real operator, 7–10
 as an integer operator, 7–7

port number, 6–79

positional parameter list, 6–162
 in start parameter list, 6–162

predefined words, B–1

primary family name, 5–17
 in family specification, 5–16

primary string
 in database name, 5–44

prindefault assignment list
 in print statement, 6–136

print attribute
 in printdefaults assignment list, 6–137

print attribute phrase, 6–136
 in print specification, 6–136
 in printdefaults assignment list, 6–137

print modifier phrase, 6–137
 in printdefaults assignment list, 6–137

print routing
 printdefaults assignment, 5–19

print specification, 6–136
 in print statement, 6–136

print statement, 6–136, 6–137

printdefaults, 5–20

printdefaults assignment, 5–19, 6–139
 in complex task attribute assignment, 5–11

printdefaults assignment list, 6–137
 in printdefaults assignment, 5–19

printdefaults attribute, 5–20

Printer Backup Files
 EXTMODE value, 6–9, 6–10

printerkind attribute
 in alter statement, 6–13

printing portions of a file, 6–139

process start, 1–4

process statement, 6–141

processing data, 1–7

product attribute
 in alter statement, 6–13

propagate option
 in copy statement, 6–68
 in move statement, 6–130

propagatesecuritytodirs attribute
 in alter statement, 6–13

propagatesecuritytofiles attribute
 in alter statement, 6–14

purge statement, 1–13, 6–142, 6–143

Q

question mark (?), as an invalid character, 8–1

QUEUEDAX system option, 6–156

quotes in a string constant, 8–3

R

railroad diagrams, explanation of, C–1

real assignment, example, 5–10

real assignment statement, 6–45, 6–46

real constant, 8–3
 in real constant primary, 7–26
 in real primary, 7–11

real constant expression, 4–2, 7–26
 in arithmetic constant comparison, 7–25
 in real constant primary, 7–26
 in real declaration, 4–4

real constant identifier, 4–2, 8–3
 in real primary, 7–11
 in real constant primary, 7–26

real constant primary, 7–26
 in real constant expression, 7–26

real declaration, 4–1, 4–4

- real expression, 7–10
 - in arithmetic comparison, 7–5
 - in integer function, 7–9
 - in named parameter list, 6–162
 - in positional parameter list, 6–162
 - in real assignment statement, 6–46
 - in real primary, 7–11
 - in run parameter list, 6–155
 - in simple task relation, 6–184
 - in subroutine invocation statements, 6–167
 - in task attribute assignment, 5–8
 - in wait specification, 6–184
 - wait statement options, 6–185
- real file attribute, 5–38, 5–39
 - in real file attribute primary, 7–12
- real file attribute primary, 7–12
 - in real primary, 7–11
- real formal parameter, 3–5
 - in named parameter list, 6–162
- real identifier, 8–3
 - in real assignment statement, 6–46
 - in real declaration, 4–4
 - in real primary, 7–11
 - in subroutine parameters, 4–7
- real parameter declaration, 3–5
- real primary, 7–11
 - in real expression, 7–10
- real relation, 7–6
 - in simple task relation, 6–184
 - in arithmetic constant comparison, 7–25
- real task attribute
 - in real task attribute primary, 7–12
 - in simple task relation, 6–184
 - in task attribute assignment, 5–8
- real task attribute primary, 7–12
 - in real primary, 7–11
- real variables, 4–4
- relative pathname
 - in currentdirectory assignment, 5–13
- release option
 - in archive statement, 6–21
- release statement, 1–13, 6–143
- releaseid attribute
 - in alter statement, 6–14
- remote files, 5–34
- remove files, 6–17
- remove from group, 6–39, 6–144
 - in archive release statement, 6–38
 - in remove statement, 6–143
- remove list, 6–39, 6–144
 - in archive release statement, 6–38
 - in remove statement, 6–143
- remove option
 - in copy statement, 6–69
- remove statement, 1–15, 6–143 , 6–144
 - entering individually from ODT, 2–5
 - example, 1–15
- removing files, 6–17
- replace statement, 6–146
- report option
 - in archive statement, 6–22
 - in copy statement, 6–69
 - in create libmaintdir statement, 6–117
 - in move statement, 6–130
 - in restore statement, 6–151
- Report Program Generator (RPG) programs
 - initiating WFL jobs, 2–7
- rerun statement, 6–147
 - entering individually from ODT, 2–5
- reserved words, B–1
- resident file
 - wait statement options, 6–185
- resolving repeated library equations, 5–43
- resource assignment, 5–21
 - in complex task attribute assignment, 5–11
- resource attribute, example, 5–21
- resource task attribute, assigning value to, 5–21
- resource-limiting attributes
 - in the job attribute list, 3–8
- restarting a job, 6–147
 - after a halt/load, 2–9
- restarting interrupted file transfers
 - copy statement, 6–115
- restore files, 6–17
- restore statement, 6–148, 6–149 , 6–154
 - autounload attribute, 6–152
 - compare option, 6–150
 - cycle attribute, 6–152
 - dsonerror option, 6–150
 - familyowner attribute, 6–152
 - file attributes, 6–151
 - libmaintdir attribute, 6–152
 - locatecapable attribute, 6–152
 - options, 6–150
 - report option, 6–151
 - serialno attribute, 6–153
 - unitno option, 6–151
 - verify option, 6–151

- version attribute, 6–153
- waitonerror option, 6–151
- restricted attribute
 - with unwrap statement, 6–170
- return statement, 6–154
 - example, 1–10
- reusing task variables, 5–25
 - example, 5–25
- rewind statement, 1–13, 6–154 , 6–155
- run parameter list, 6–155
 - in run statement, 6–155
- run statement, 5–2, 6–155
 - example, 1–4
 - in process statement, 6–141
- running jobs on a remote host, 2–6

S

- sample job, A–1
 - compiling a program, A–1
 - applying a patch, A–1
 - initiating other jobs, A–4
 - producing cross-reference files, A–1
 - updating files, A–5
- savefactor attribute, 6–94
 - archive CD volume attributes, 6–35
 - archive tape volume attributes, 6–31
 - in alter statement, 6–14
- SCHEDULED task state, 7–5
- scope of declarations, 1–9, 4–1 , 4–8
- scratch pool name, 6–25
- scratchpool attribute, 6–85
 - archive tape volume attributes, 6–32
- sectors option
 - archive rollout statement, 6–42
- securityadmin attribute
 - in alter statement, 6–14
- security from group, 6–159
 - in file specification, 6–159
- security list, 6–159
 - in file specification, 6–159
- security specification, 6–160
 - in security statement, 6–159
- security statement, 1–15, 6–159 , 6–161
 - entering individually from ODT, 2–5
 - example, 1–15
- security tape volumes, 6–180
- securityguard attribute, 6–94
 - archive tape volume attributes, 6–31
 - in alter statement, 6–14
 - in volume add statement, 6–182
- securitylabels attribute
 - in volume add statement, 6–182
 - in volume statement, 6–179
- securitymode attribute
 - in alter statement, 6–14
 - in volume add statement, 6–182
 - volume statement, 6–179
- securitytype attribute, 6–94
 - archive tape volume attributes, 6–32
 - in alter statement, 6–14
 - in volume add statement, 6–182
- securityuse attribute, 6–94
 - archive tape volume attributes, 6–32
 - in alter statement, 6–15
 - in volume add statement, 6–183
- select option
 - in copy statement, 6–69
 - in move statement, 6–130
 - in restore statement, 6–151
- select statement
 - report option, 6–151
- sensitivedata attribute, 6–95
 - in alter statement, 6–15
- sequence number, 9–2
 - in include control option, 9–2
- serial number, 5–36
 - in serial number list, 5–36
- serial number assignment, 5–36
 - example, 5–37
- serial number list, 5–36
 - in archive disk volume attribute list, 6–23
 - in source volume attributes, 6–148
 - in volume attribute list, 6–174
 - in serial number assignment, 5–36
- serialno attribute, 5–36, 6–85 , 6–117 , 6–153
 - archive CD volume attributes, 6–36
 - archive disk volume, 6–23
 - archive tape volume attributes, 6–32
 - in volume statement, 6–179
- serialno option
 - catalog statement, 6–49
- setgroupcode attribute
 - in alter statement, 6–15
 - in volume statement, 6–179
 - volume add statement, 6–183
- setusercode attribute
 - in alter statement, 6–15
 - in volume statement, 6–179
 - volume add statement, 6–183
- simple task relation, 6–184
 - in wait specification, 6–184

- wait statement options, 6-185
- simple unwrap request, 6-169
 - in unwrap group, 6-169
- simple wrap request, 6-188
 - in wrap group, 6-188
- singleunit attribute, 6-95
- skipexclusive option
 - in archive statement, 6-22
 - in copy statement, 6-70
 - in move statement, 6-130
- slash (/)
 - as a real operator, 7-10
 - as a string operator, 7-13
- slash-equal (/=), as a string operator, 7-13
- source, 6-170, 6-188
 - in unwrap volume, 6-170
 - in wrap volume, 6-188
- source volume, 6-148
- source volume file, 6-73
- source volume name, 6-148
 - in restore statement, 6-148
- specifications, family, 6-18
- specifying copy options, 6-65
- specifying file attributes
 - in copy statement, 6-60
- start and wait statement, 6-166
- start command, 2-1
 - MARC, 2-6
 - CANDE, 2-2
- start parameter list, 6-162
 - in start statement, 6-162
- start statement, 1-4, 1-8, 5-2, 6-162, 6-163
 - entering individually from ODT, 2-5
 - in process statement, 6-141
 - initiating jobs through, 2-4
- starttime spec
 - in start statement, 6-162
 - in starttime specification, 3-11
- starttime specification, 2-12, 3-11, 3-13
 - example, 3-13
- statement list, 3-15
 - example, 3-16
 - in compound statement, 6-59
 - in subroutine block, 4-7
- statements
 - abort, 6-4
 - add or copy, 6-61
 - alter, 6-4
 - archive differential, 6-3, 6-17
 - archive full, 6-3
 - archive incremental, 6-3, 6-17
 - archive merge, 6-3, 6-17
 - archive purge, 6-4, 6-17
 - archive release, 6-4, 6-17
 - archive restore, 6-3, 6-17
 - archive rollout, 6-3, 6-17
 - abort, 6-2
 - catalog delete, 6-49
 - cataloging catalog, 6-4
 - access, 6-3, 6-5
 - change, 6-4, 6-50
 - compare option, 6-21
 - add, 6-3, 6-6
 - display, 6-3
 - do, 6-120
 - executing interpretively, 2-1
 - go, 6-120
 - if, 6-121
 - initialize, 6-2
 - instruction, 6-3
 - alter, 6-8
 - modify, 6-4
 - nesting, 6-1
 - null, 6-131
 - on, 6-132
 - password, 6-3
 - PB, 6-136
 - archive backup, 6-19
 - archive full, 6-17
 - archive merge, 6-37
 - archive purge, 6-38
 - archive release, 6-39
 - archive restore, 6-39
 - archive restoreadd, 6-17
 - archive rollout, 6-41
 - release, 6-4
 - remove, 6-143
 - replace, 6-146
 - rerun, 6-147
 - security, 6-4
 - assignment, 6-2, 6-46
 - subroutine invocation statement, 6-2
 - task control statement, 1-11
 - task initiation, 1-4, 5-1
 - volume, 6-174
 - bind, 6-3, 6-47
 - case, 6-2, 6-48
 - catalog add, 6-49
 - catalog, 6-49
 - catalog purge, 6-49
 - cataloging, 6-4

- cataloging volume, 6-4
- change purge, 6-4
- communication, 1-15, 6-3
- compile, 6-3
- compile or bind, 6-54
- compound, 6-2, 6-59
- copy or add, 6-61
- create libmaintdir, 6-116
- crunch, 6-119
- display, 6-119
- do, 6-2
- file handling, 1-13, 6-3
- file management, 1-14, 6-4
- flow-of-control, 1-6, 6-2
- go, 6-2
- groupings, 6-1
- if, 6-2
- initialize, 6-122
- instruction, 6-123
- label identifiers, 3-16
- library maintenance, 1-14
- lock, 6-4, 6-124
- log, 6-3, 6-125
- mkdir, 6-4, 6-125
- modify, 6-126
- move, 6-128
- null, 6-2
- on, 6-2
- open, 1-13, 6-135
- options, 6-116
- password, 6-135
- PB, 6-3
- print, 6-4, 6-136
- process, 6-3, 6-141
- purge, 6-142
- recognized by the controller, 2-5
- release, 6-143
- remove, 6-4
- rerun, 6-2
- restore, 6-3, 6-148, 6-154
- return, 6-2
- rewind, 6-4, 6-154
- run, 6-3, 6-155
- security, 6-159
- start, 6-3
- stop, 6-2, 6-166
- subroutine control, 1-9, 6-2
- subroutine invocation, 6-167
- task control, 1-11, 6-2
- task initiation, 1-3, 6-3
- task security, 6-3
- unwrap, 6-169
- user, 6-3, 6-173
- volume, 6-3, 6-4
- wait, 6-2, 6-184
- while, 6-2, 6-187
- wrap, 6-165, 6-188
- station task attribute, remote files, 5-35
- stationmaster task attribute
 - and remote files, 5-34
- status attribute, 2-9
- stop statement, 1-11, 6-166
- STOPPED task state, 7-5
- storing jobs in disk files, 2-2
- string assignment statement, 6-45, 6-46
- string comparison, 7-6
- string concatenation operators, 7-13
- string constant, 8-3
 - in string constant primary, 7-27
 - in string primary, 7-14
 - quotation marks in, 8-3
- string constant comparison, 7-25
 - in Boolean constant primary, 7-25
- string constant expression, 4-2, 7-27
 - in take-drop constant function, 7-27
 - in translate constant function, 7-27
 - in case constant expression, 6-48
 - in character set, 7-16
 - in head-tail constant functions, 7-27
 - in real constant primary, 7-26
 - in string constant comparison, 7-25
 - in string constant primary, 7-27
 - in string declaration, 4-5
- string constant function, 7-27
 - in string constant primary, 7-27
- string constant identifier, 4-2, 8-3
 - in string constant primary, 7-27
 - in string primary, 7-14
- string constant primary, 7-27
 - in string constant expression, 7-27
- string declaration, 4-1, 4-4
- string expression, 7-13
 - in abort statement, 6-4
 - in accept function, 7-15
 - in case expression, 6-48
 - in display statement, 6-119
 - in family assignment, 5-16
 - in head-tail function, 7-16
 - in hex function, 7-11
 - in length function, 7-9
 - in named parameter list, 6-162
 - in octal function, 7-11

- in positional parameter list, 6–162
- in print attribute phrase, 6–136
- in run parameter list, 6–155
- in serial number, 5–36
- in stop statement, 6–166
- in string assignment statement, 6–46
- in string comparison, 7–6
- in string primary, 7–14
- in subroutine invocation statements, 6–167
- in take-drop functions, 7–19
- in task attribute assignment, 5–8
- translate functions, 7–22
- wait statement options, 6–184
- in path specification, 5–14, 5–15
- in print modifier phrase, 6–137
- in wait statement, 6–184
- string file attribute, 5–38
 - in string file attribute primary, 7–14
- string file attribute primary, 7–14
 - in string primary, 7–14
- string formal parameter, 3–5
 - in named parameter list, 6–162
- string function, 7–17
 - in decimal function, 7–8
 - in string primary, 7–14
- string identifier, 8–3
 - in string assignment statement, 6–46
 - in string declaration, 4–5
 - in string primary, 7–14
 - in subroutine parameters, 4–7
- string operators, in string primary, 7–13
- string parameter declaration, 3–5
- string parameters, 8–11
 - passing to a job, 8–12
 - restrictions, 8–12
- string policy
 - in mnemonic primary, 7–23
- string primary, 7–14
 - in alternategroups value, 6–160
 - in device kind assignment, 5–35
 - in option assignment, 5–18
 - in security specification, 6–160
 - in source volume name, 6–148
 - in start statement, 6–162
 - in string expression, 7–13
 - in task mnemonic primary, 7–23
 - in usercode assignment, 5–22
 - in volume attribute list, 6–174
 - in accesscode assignment, 5–11
- string print attribute
 - in print attribute phrase, 6–136
- string print modifier
 - in print modifier phrase, 6–137
- string task attribute
 - in string task attribute primary, 7–15
 - in task attribute assignment, 5–8
- string task attribute primary, 7–15
 - in string primary, 7–14
- string variables, 4–4, 4–5
 - example, 4–5
- subdirectory
 - in directory, 5–14, 5–15
- subroutine block, 4–7
- subroutine control statement, 1–9, 6–2
 - return, 6–2
 - subroutine invocation statement, 6–2
- subroutine declaration, 4–1, 4–7
- subroutine identifier, 8–3
 - in subroutine block, 4–7
 - in subroutine invocation statements, 6–167
- subroutine invocation statement, 6–167
 - in process statement, 6–141
- subroutine invocation statements, 6–167
- subroutine parameters, 4–7, 4–8
- subroutines, 4–7
 - declaring, 4–7
 - ending identifiers, 1–17
 - example, 1–9, 1–17
 - initiating, 6–167
 - nesting, 4–8
 - parameters, 4–8
 - example, 4–9
 - naming, 4–7
 - scope of declarations, 1–9, 4–1, 4–8
- substitution, family, 6–18
- subsystems, archive, 6–16
- suppresswarning assignment, 5–21
 - example, 5–22
 - in complex task attribute assignment, 5–11
- suppresswarning list, 5–21
 - in suppresswarning assignment, 5–21
- synchronous task initiation, 5–2
- syntax job disposition, 3–7
 - example, 3–7
- syntax option
 - compile or bind statement, 6–56
- system function, 7–18
 - in string primary, 7–14

system identification information, 7-18
system/backup parameters
 in PB statements, 6-136
SYSTEM/BACKUP utility, initiating, 6-136
SYSTEM/FILEDATA utility program, 6-83
SYSTEM/PATCH utility, A-2

T

tail function, 7-16, 7-27
take-drop constant functions, 7-27
 in string constant primary, 7-27
take-drop functions, 7-19
 in string primary, 7-14
tape
 copying files from, 6-95
 initiating WFL jobs from, 2-8
tape attributes
 in create libmaintdir statement, 6-116
 in restore statement, 6-148
tape directory disk file, 6-30
tape drives
 maximum blocksize, 6-81
tape name, 8-5
 in create libmaintdir statement, 6-116
 in volume name, 8-6
tape security, 6-175, 6-180
tape security file attributes, 6-180
tape volume directory, 6-175
target family name, 5-17
 in family specification, 5-16
task
 abnormal termination, effect on a job, 2-9
 discontinuing, 6-4
 status, inquiring about, 5-28
 history, inquiring about, 5-28
task assignment statement, 6-45, 6-46
task attribute assignment, 5-8
 in task declaration, 4-6
 in add statement, 6-6
 in archive task equation list, 6-36
 in compiler task equation list, 6-57
 in copy or add statement, 6-60
 in modify statement, 6-126
 in replace statement, 6-146
 in task assignment statement, 6-46
 in task equation list, 5-3
 in unwrap statement, 6-169
 in wrap statement, 6-165, 6-188
task attributes, 1-5, 5-3
 assigning value to, 5-14, 5-15

 functional groupings, 5-4
 inquiring the values of, 5-27
 keywords, B-3
 assigning, 5-24
 assigning value to, 5-13
 compiler task equation list, 6-57
 example, 1-5
task control, 1-11
task control statements, 1-11, 6-2
 abort, 6-2
 on, 6-2
 stop, 6-2
 wait, 6-2
 initialize, 6-2
 rerun, 6-2
task declaration, 4-1, 4-6
task equation, 1-5, 5-3, 6-125, 6-136, 6-141, 6-156
task equation list, 5-3
 in log statement, 6-125
 in PB statements, 6-136
 in run statement, 6-155
task identifier, 8-3
 in add statement, 6-6
 in archive merge statement, 6-37
 in archive restore statement, 6-39
 in archive rollout statement, 6-41
 in Boolean task attribute primary, 7-4
 in compile or bind statement, 6-53
 in copy or add statement, 6-60
 in family assignment, 5-16
 in initialize statement, 6-122
 in log statement, 6-125
 in PB statements, 6-136
 in real task attribute primary, 7-12
 in run statement, 6-155
 in simple task relation, 6-184
 in start statement, 6-162
 in subroutine invocation statements, 6-167
 in subroutine parameters, 4-7
 in task assignment statement, 6-46
 in task mnemonic comparison, 7-7
 in task mnemonic primary, 7-23
 in task state, 7-4
 in unwrap statement, 6-169
 in wait specification, 6-184
 in wrap statement, 6-165, 6-188
wait statement options, 6-185
 in abort statement, 6-4
 in bind statement, 6-47

- in copy or add statement, 6–60
 - in integer task attribute primary, 7–10
 - in replace statement, 6–146
 - in task declaration, 4–6
 - wait statement options, 6–185
- task identifier assignment, 4–6
- task initiation, example, 5–2
- task initiation statements, 1–3, 1–4 , 5–1 , 6–3
 - add, 5–1, 6–3
 - archive, 5–1
 - archive differential, 6–3
 - archive incremental, 6–3
 - archive restore, 6–3
 - archive rollout, 6–3
 - bind, 5–1, 6–3
 - compile, 5–1
 - copy, 5–1
 - log, 5–2, 6–3
 - PB, 5–2, 6–3
 - archive full, 6–3
 - archive merge, 6–3
 - run, 5–2, 6–3
 - start, 5–2
 - compile, 6–3
 - process, 6–3
 - restore, 6–3
 - start, 6–3
- task mnemonic
 - in task mnemonic primary, 7–23
- task mnemonic comparison, 7–7
 - in wait specification, 6–184
 - wait statement options, 6–185
- task mnemonic primary, 7–23
 - in task attribute assignment, 5–8
 - in task mnemonic comparison, 7–7
- task security statements, 6–3
 - access, 6–3
 - user, 6–3
 - password, 6–3
 - volume, 6–3
- task specifications, 1–5
- task state, 7–4
 - ABORTED, 7–5
 - ACTIVE, 7–5
 - COMPLETED, 7–5
 - in wait specification, 6–184
 - INUSE, 7–5
 - SCHEDULED, 7–5
 - STOPPED, 7–5
 - wait statement options, 6–185
- COMPILEDOK, 7–5
- COMPLETEDOK, 7–5
- task variables, 4–6, 5–9 , 5–23 , 6–62
 - example, 5–25
 - predeclared myself, 5–29
 - reusing, 5–25
 - compile statement, 6–56
 - example, 5–23, 5–25
 - predeclared myjob, 5–29
- tasks
 - using a different input file, 5–31
 - using a different output file, 5–31
- taskvalue attribute
 - with unwrap statement, 6–172
 - with wrap statement, 6–172, 6–190
- terminating a job or asynchronous subroutine, 6–166
- time, 3–11
- time information, 7–20
- time interval, 3–11
- timedate functions, 7–20
 - in string primary, 7–14
- title assignment, example, 5–10
- title file attribute, 5–38
 - in string file attribute primary, 7–14
- title print attribute
 - in print attribute phrase, 6–136
- title task attribute
 - in string task attribute primary, 7–15
 - in task attribute assignment, 5–8
- titles, 8–7
- total core, 5–12
 - in core assignment, 5–12
 - example, 5–12
- traditional security specification, 6–160
 - in security specification, 6–160
- trainid attribute
 - in alter statement, 6–15
- transfer service, 6–106
 - in add statement, 6–6
 - in copy or add statement, 6–60
 - in copy statement, 6–106
- transform attribute
 - in alter statement, 6–15
- translatable file attributes, 5–38
- translate constant function, 7–27
- translate functions, 7–22

- underscore, 8–2
 - in source volume name, 6–148
- unitno attribute, 6–86
- unitno option
 - in restore statement, 6–151
- unknown host name, 3–3
- unwrap file, 6–170
 - in simple unwrap request, 6–169
 - in unwrap request, 6–169
- unwrap group, 6–169
 - in unwrap statement, 6–169
- unwrap request, 6–169
 - in unwrap group, 6–169
- unwrap statement, 6–169
- unwrap volume, 6–170
 - in unwrap statement, 6–169
- UPPERCASE function, 7–22, 7–27
- usecatalog attribute, 6–95
 - archive tape volume attributes, 6–32
- useguardfile attribute
 - in alter statement, 6–15
- usepath option
 - in copy statement, 6–70
 - with wrap statement, 6–190
- user statement, 1–12, 6–173
 - example, 1–12
- user, synonym for usercode task attribute, 3–9
- usercode
 - changing, 6–173
 - in archive rollout statement, 6–41
 - in directory, 5–14, 5–15
 - in source volume attributes, 6–148
 - in usercode assignment, 5–22
 - in volume attribute list, 6–174
 - copying files with, 6–96
 - copying files without, 6–96
- usercode assignment, 5–22
 - example, 5–23
 - in complex task attribute assignment, 5–11
- usercode attribute, 6–86
- usercode name constant
 - in file name constant, 8–8
 - in user statement, 6–173
- usercode password, changing, 6–135
- usercode task attribute
 - inquiring value of, 5–29
- usercode task attributes
 - assigning value to, 5–22

V

Index-25

- securitylabels, 6-182
- securitymode, 6-182
- securitytype, 6-182
- securityuse, 6-183
- setgroupcode, 6-183
- setusercode, 6-183
- userguardfile, 6-183
- volume attribute list, 6-174
 - in volume statement, 6-174
- volume change option
 - in volume statement, 6-176
- volume delete option
 - in volume statement, 6-176
- volume destroyed option
 - in volume statement, 6-177
- volume directory, 6-175
- volume library, 6-175
- volume name, 8-6
 - in volume statement, 6-174
 - in volume statement, 6-175
- volume offsite option
 - in volume statement, 6-177
- volume onsite option
 - volume statement, 6-177
- volume recovered option
 - volume statement, 6-177
- volume statement, 6-174, 6-175
 - group, 6-177
 - groupr, 6-178
 - grouprwx, 6-178
 - groupw, 6-178
 - groupx, 6-178
 - guardowner, 6-178
 - other, 6-178
 - otherrwx, 6-178
 - otherw, 6-178
 - otherx, 6-178
 - ownerr, 6-178
 - ownerrwx, 6-179
 - ownerw, 6-178
 - ownerx, 6-179
 - securitylabels, 6-179
 - securitymode, 6-179
 - serialno, 6-179
 - setgroupcode, 6-179
 - setusercode, 6-179
 - userguardfile, 6-180
 - volume add, 6-176
 - volume change, 6-176
 - volume delete, 6-176
 - volume destroyed, 6-177

- volume offsite, 6-177
- volume onsite, 6-177
- volume recovered, 6-177

W

- wait specification, 6-184
 - in wait statement, 6-184
- wait statement, 1-11, 6-184
 - example, 1-12
- wait statement options
 - OK, 6-185
 - Boolean task attribute, 6-185
 - real expression, 6-185
 - resident file, 6-185
 - simple task relation, 6-185
 - string expression, 6-184
 - task identifier, 6-185
 - task mnemonic comparison, 6-185
 - task state, 6-185
- waitonerror option
 - in archive statement, 6-22
 - in copy statement, 6-72
 - in create libmaintdir statement, 6-117
 - in move statement, 6-131
 - in restore statement, 6-151
- warning number
 - in suppresswarning list, 5-21
- warnsuppress control option, 9-6
- WFL command, 2-1
 - in CANDE, 2-1
- WFL job, 1-3
 - example, 3-17
- WFLSupport library
 - compatible release level with MCP, 1-3
- while statement, 1-7, 6-187
- window size attribute, 6-86
- word file attributes, 5-39
- words, B-1
 - context-sensitive, B-2
 - predefined, B-1
 - reserved, B-1
- wrap file, 6-188
 - in simple wrap request, 6-188
 - in wrap request, 6-188
- wrap group, 6-188
 - in wrap statement, 6-165, 6-188
- wrap password, 8-6
- wrap request, 6-188
 - in wrap group, 6-188
- wrap statement, 6-165, 6-188

wrap volume, 6–188
 in wrap statement, 6–165, 6–188

X

xref control option, 9–7
xreffiles option, 9–8

Y

yourusercode attribute, 6–86
yy in date, 3–11

yyyy in date, 3–11

Special Characters

* (asterisk) option, example, 5–18
\$ (dollar sign)
 WFL control option, 3–2
(pound sign), 1–17, 8–6
*DIR, creating, 6–125
% (percent), 1–17
 comment delimiter, 3–2
? (question mark), 2–2

