Burroughs

Language Manual

B 1000 Systems Work Flow Language (WFL)

(Relative to Release Level 13.0)

Distribution Code SC

Priced Item
Printed in U.S.A.
September 1986

5025265

TABLE OF CONTENTS

TABLE OF CONTENTS (Cont.)

TABLE OF CONTENTS (Cont.)

TABLE OF CONTENTS (Cont.)

TABLE OF CONTENTS (Cont.)

TABLE OF CONTENTS (Cont.)

LIST OF TABLES

SECTION 1
INTRODUCTION

PURPOSE AND SCOPE

This manual describes the Work Flow Language (WFL) as implemented on the B 1000 Systems. It provides the information necessary to write source-code programs.

This manual is written for programmers.

WFL CONCEPTS

A task is the execution of a program. Tasks can be divided into job task, dependent task, and independent task.

A job task is the execution of a program written in the Work Flow Language. A dependent task is any task initiated by a job task or called by another task. An independent task is any task initiated by the EXECUTE ODT-command.

A job is a collection of tasks including the job task (or an independent task) and all the dependent tasks initiated by tasks in the job. Work Flow Language (WFL) is the means by which a job is described and presented to the B 1000 computer system. The language allows the user to programmatically control the execution of a set of interrelated tasks. A job can decide, at run-time, whether to run a program, which programs to run, and in what order to run them.

WFL ADVANTAGES

WFL allows a user to easily describe a complex system of programs.

WFL is more flexible than using control cards in pseudo readers and is easier to use than writing a job spawner.

B 1000 WFL is a subset of the B 5000/B 6000/B 7000/A Series WFL and as such it allows for distributed networks with common sources and allows for growth to a larger system. Refer to section 11, Portability Warnings, for a list of constructs which are not a subset of B 5000/B 6000/B 7000/A Series WFL.

RELATED DOCUMENTS

The following documents are referenced in this document:

B 1000 Systems System Software Operation Guide, Volume 1, form number 5024508.

B 1000 Systems System Software Operation Guide, Volume 2, form number 1169091.

SECTION 2 BASIC ELEMENTS AND CONSTRUCTS

GENERAL

This section describes the basic elements and constructs in the Work Flow Language. These basic elements and constructs are used throughout this manual.

WFL SOURCE FILE AND RECORD FORMAT

A WFL source program file should have a FILEKIND of JOBSYMBOL. The format of a record in a JOBSYMBOL file consists of the following information:

1. Columns 1 through 80 contain text of WFL job.
2. Columns 81 through 82 are not used.
3. Columns 83 through 90 contain the sequence number of the source record.

A WFL source program may also have a FILEKIND of DATA. The record size of the file should be 80 characters and contain only the text of a WFL job.

A WFL job entered as an ODT command or from the statements of the various programming languages is considered all one record of text; sequence numbers are not included.

CHARACTER SET

The Work Flow Language uses the EBCDIC character set. An invalid character is not allowed except in column one.

INVALID CHARACTER

Throughout this document, the symbol <i> is used to signify an invalid character in column one of a source record. An invalid character is any character that is not a valid EBCDIC character. If the source file is a storage media such as disk or tape, which cannot represent an invalid punch, then the invalid character <i> is represented by an EBCDIC question mark character (?).

The invalid character <i> is allowed in several syntax diagrams. It may also be used as a substitute for a semicolon.

Statements are separated by a semicolon or the invalid character <i>. Source records may contain more than one statement (properly separated by semicolons), except that WFL input from the ODT or from statements of the various programming languages may present only one source record. In this case, the question mark character (substituting for the invalid character <i>) may occur only in the first character of the input.

COMMENTS

Source records can be terminated by a percent sign (%), if such a character is not part of a string of characters within quotation marks. The remainder of the record is ignored and may contain comments.

Although comments are ignored, they are included in any listing of the job.

CHARACTER ELEMENTS

<letter>
Any one of the 26 upper-case English alphabet characters from A to Z.

<digit>
Any one of the 10 numerals 0 to 9.

<hyphen>
The hyphen character (-).

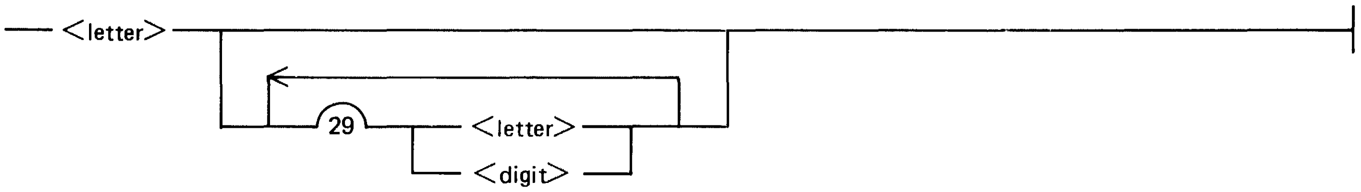
<underscore>
The underscore character (_).

<string char>
Any EBCDIC character except the quotation mark (").

IDENTIFIER

<Identifier>s are names for variables and subroutines.

<identifier> syntax:



An <identifier> is terminated by any non-alphanumeric character (including a blank), or the end of the record image. An <identifier> must not be broken across a record boundary.

An <identifier> must not be spelled the same as a reserved word. An <identifier> may be spelled the same as a special word; however, the special word loses its special meaning for the scope of the declaration.

Reserved and special words are listed in section 9.

Examples:

Valid Identifiers	**Invalid Identifiers**
A	1234
Z123	1ABC
ABC123	W-2
	A$
	BEGIN

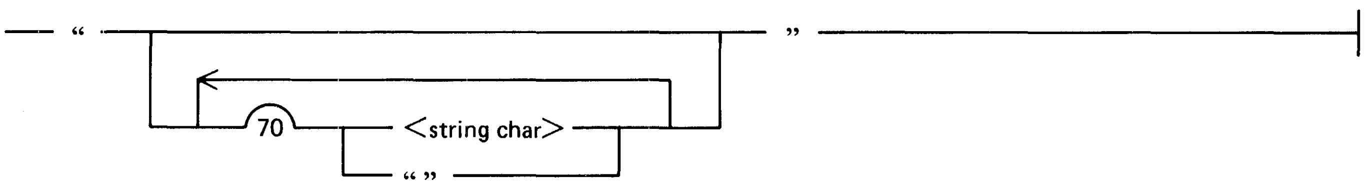
CONSTANTS

Constants are data items whose value is implied by the characters of which they are composed, or by the specification of a reserved word. There are three classes of constants: string, integer, and boolean.

String Constant

A <string constant> may be composed of any set of EBCDIC characters and represents that set of characters.

<string constant> syntax:



A pair of quotation mark characters ("") appearing alone represents a null string (a string of length zero). A pair of quotation mark characters ("") appearing in a string represents a single quotation mark character (") within the string.

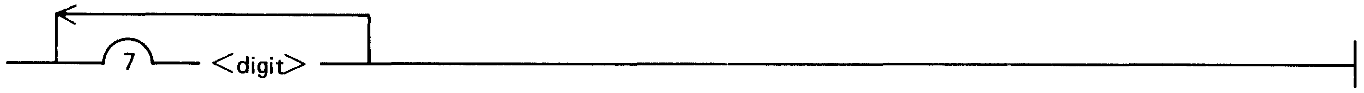
Examples:

String Constant	Value Represented
"ABC"	ABC
"defg"	defg
"12345"	12345
"-123.45"	-123.45
"?*-> "	?*->
"""WORD"""	"WORD"
""""	"
""	
"A""B"	A"B

Integer Constant

<Integer constant>s are used to represent numeric values that are whole positive numbers.

<integer constant> syntax:



An <integer constant> is terminated by any non-numeric character, including a blank, or by the end of the record image. An <integer constant> must not be broken across a record boundary. The maximum value allowed for an <integer constant> is 8388607.

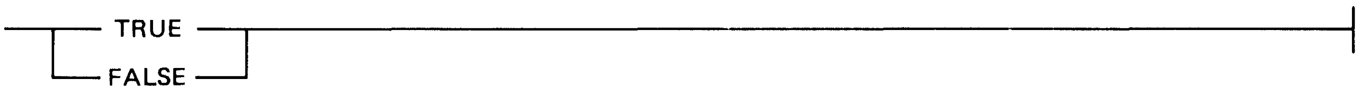
Examples:

1
1234567

Boolean Constant

<Boolean constant>s represent logical values.

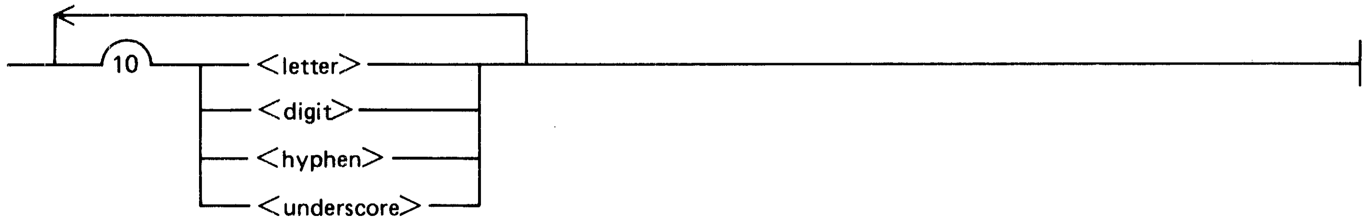
<boolean constant> syntax:



NAME

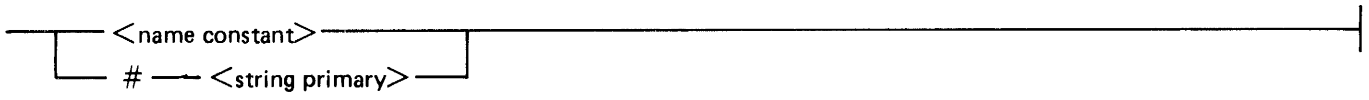
A <name> is used to identify constructs such as filenames, family names, usercodes, and so forth.

<name constant> syntax:



Some items restrict the maximum number of characters that can be used in a <name> to other than 10 characters.

<name> syntax:



The <string primary> must evaluate to a <name constant> at run-time. <String primary> is described in section 7, Expressions.

The #<string primary> syntax may be used to dynamically build constructs such as filenames, family names, usercodes, and so forth.

Examples:

```
A
ZZZ
123456789
1A2B
A1B2
W-2
2_BE
#"SPECIAL"
#PACK      % PACK is a string variable
```

<name17> syntax:

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

A <name17> can be composed of no more than 17 characters.

<name9> syntax:

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

A <name9> can be composed of no more than nine characters.

<name8> syntax:

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

A <name8> can be composed of no more than eight characters.

BASIC CONSTRUCTS

Several special purpose constructs are described in the following paragraphs.

Family Name

A <family name> is the name of a disk, pack, or tape volume.

<family name> syntax:

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

A <family name> of DISK may be used to specify the system disk. DISK overrides the default pack associated with a usercode.

Hostname

A <hostname> is the name assigned to a system for the purpose of intersystem communication.

<hostname> syntax:

```
── <name17> ────────────────────────────────────────────────────┤
```

Usercode

A <usercode> is the name assigned to a user to secure system and file access.

<usercode> syntax:

```
── <name8> ─────────────────────────────────────────────────────┤
```

Password

A <password> is a name used to authenticate a <usercode>.

<password> syntax:

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

User Specification

A <user specification> identifies and authenticates a user.

<user specification> syntax:

```
── <usercode> ─┬──────────────────────────┬─────────────────────┤
               └── / ── <password> ───────┘
```

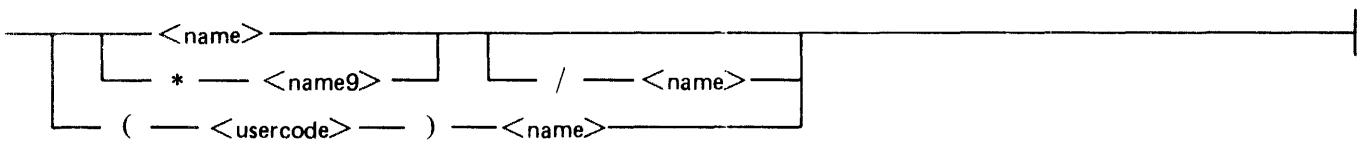
FILE NAMES AND TITLES

<Filename>s and <file title>s are used to identify physical files.

Filename

A <filename> identifies a physical file without consideration for the volume upon which it resides.

<filename> syntax:



An asterisk (*) is used in a <filename> or <directory name> to override the default usercode (and the default <family name> associated with that usercode). It is only necessary to use an asterisk in a <filename> or <directory name> if the file is not stored in a usercode directory. Use a <family name> of DISK to override the default pack.

The use of <name>/<name> without a preceding asterisk in a program run under a usercode will override the default usercode (but not the default <family name>).

NOTE
Users are strongly encouraged to use a <family name> of DISK instead of an asterisk to override the default pack. When it is necessary to override both the default usercode and the default pack, the use of both an asterisk and a <family name> of DISK is strongly encouraged.

When specifying a dynamic name or title, a <string primary> may contain the entire name or title. When a <string primary> is used for a part of the name or title, the string can contain any portion of the name or title, as long as the result conforms to the correct form.

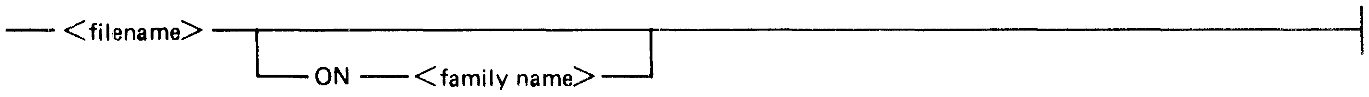
Examples:

```
XYZ
*A/B
(ZOT)XYZ
```

File Title

A <file title> identifies a physical file and the disk on which it resides.

<file title> syntax:



Examples:

```
MYNM
*A/B ON MYPACK
(ZOT)XYZ ON MYPACK
MYNM ON DISK
*COBOL ON DISK
```

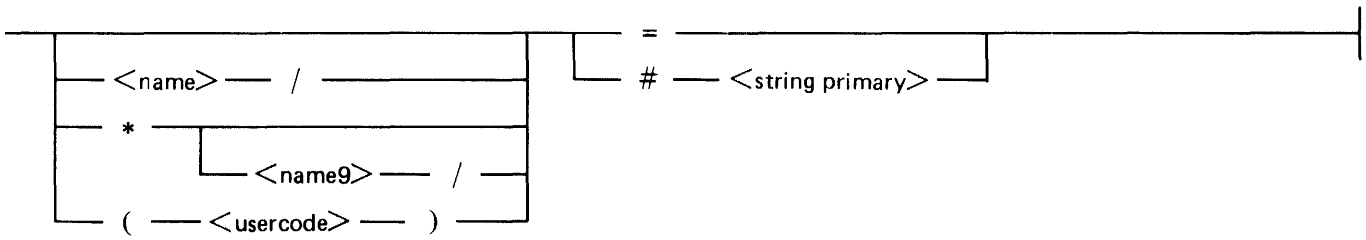
DIRECTORY NAMES AND TITLES

<Directory name>s and <directory title>s are used to identify a group of physical files that have the same first names or belong to the same user.

Directory Name

A <directory name> identifies a group of physical files without consideration for the volume upon which they reside.

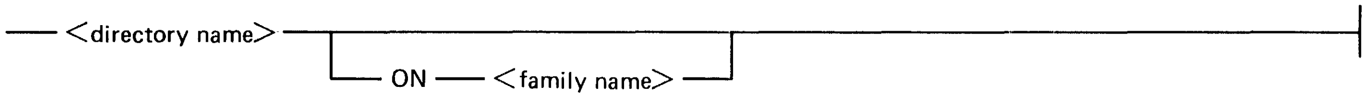
<directory name> syntax:



Directory Title

A <directory title> identifies a group of physical files and the disk on which they reside.

<directory title>



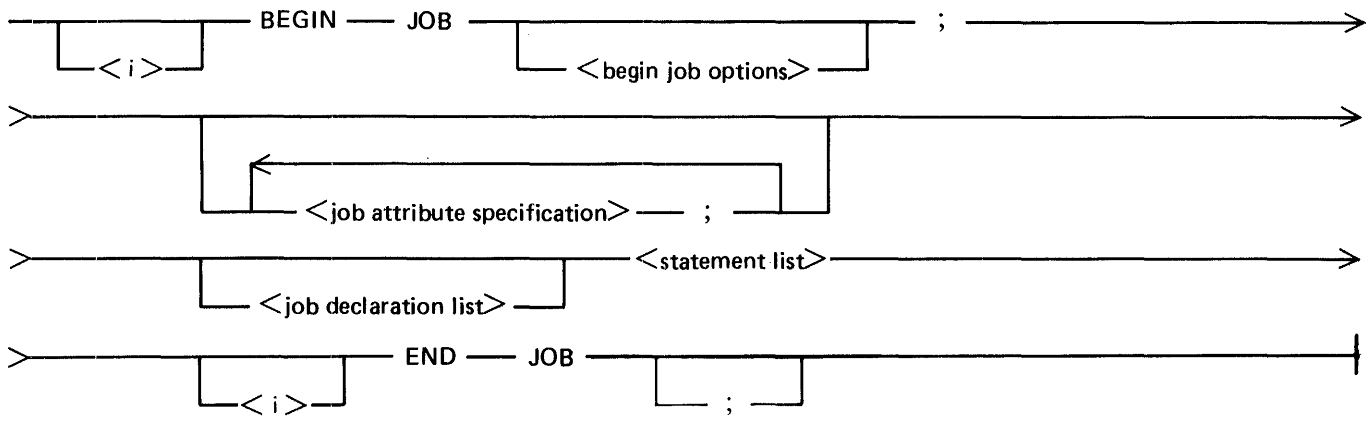
Examples:

```
=
*A/= ON 1A2B
(XYZ)= ON ZZZ
= ON DISK
*= ON DISK
```

SECTION 3
JOB

A program written in the Work Flow Language describes a job.

<job> syntax:



A job allows the user to describe sophisticated control over the initiation of tasks and system functions. The user may specify sequences of tasks.

There is a limit to the number of <statement> s in the <statement list> of a < job>, <subroutine declaration>, or <function declaration>. WFL is guaranteed to handle a minimum of 600 <statement>s. In most cases it handles 10 times that amount. An error is displayed if this limit is reached. Should this happen, the < statement>s may be split into separate subroutines. For a description of a <subroutine declaration>, refer to Subroutine Declaration in section 4. For a description of a <function declaration>, refer to Function Declaration in section 4.

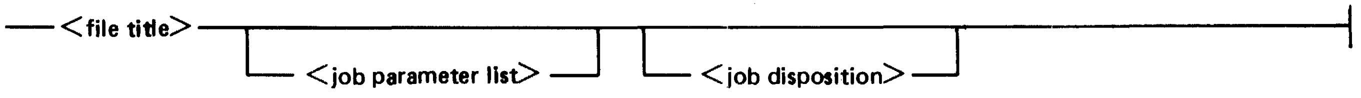
Examples:

```
1. BEGIN JOB A/JOB; % compile and execute job A/JOB
   USERCODE = C/D; % the job will run under usercode C
   FETCH = "A/JOB REQUIRES MYPACK(DISK)";
   MAXLINES=100; % the job may print a maximum of 100 lines
   COMPILE X WITH COBOL LIBRARY;
     COMPILER FILE CARD (TITLE=X/Y);
   RUN X;
   END JOB

2. <i> BEGIN JOB ANOTHER/JOB SYNTAX; % syntactically check the job
         RUN X;
         RUN Y
   <i> RUN Z
   <i> END JOB
```

BEGIN JOB OPTIONS

The <begin job options> are used to specify the title, parameters, and disposition of a job.

<begin job options> syntax:



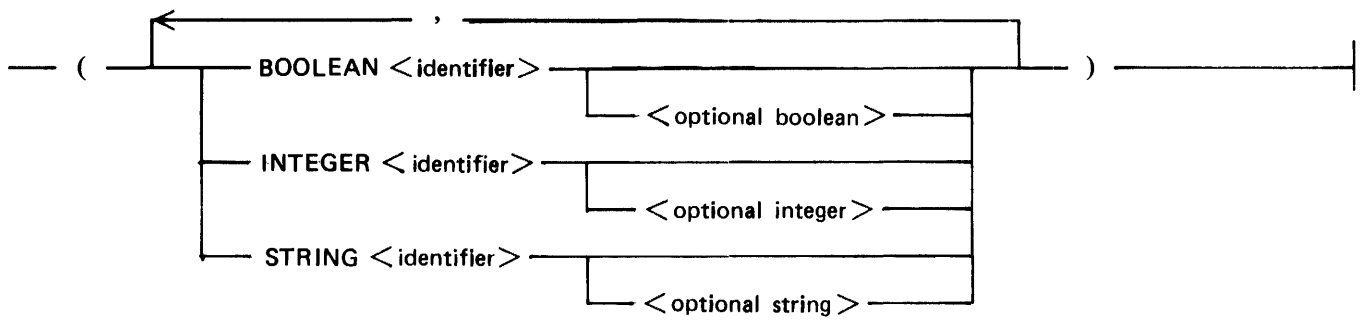
The <file title> is used to identify the job task when the job is initiated. It appears on the heading page of the job summary. (Refer to section 12, Operation, for more information about job summaries.) The <file title> has no other significance.

If the <begin job options> are omitted, the job task is titled "JOB" followed by the JOBNUMBER of the job.

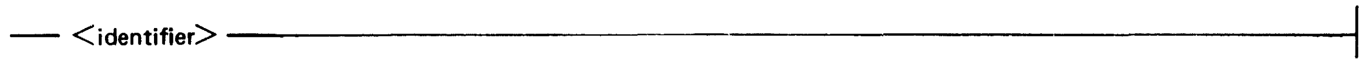
Job Parameter List

A <job parameter list> specifies the formal parameter identifiers for the job and their types.

<job parameter list> syntax:

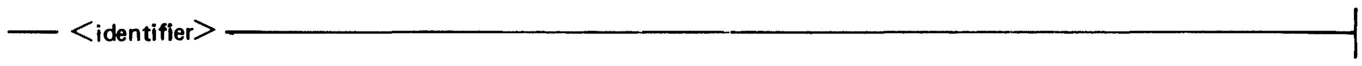


<boolean parameter id> syntax:



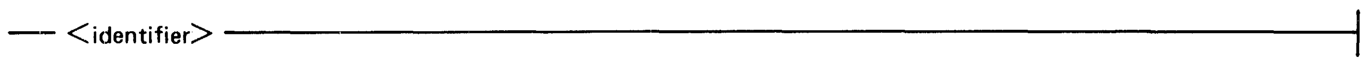
The <identifier> must have been specified in a <job parameter list> with a type of BOOLEAN.

<integer parameter id> syntax:



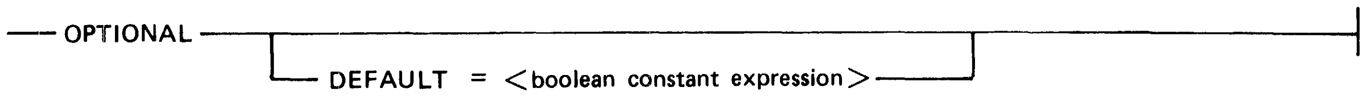
The <identifier> must have been specified in a <job parameter list> with a type of INTEGER.

<string parameter id> syntax:

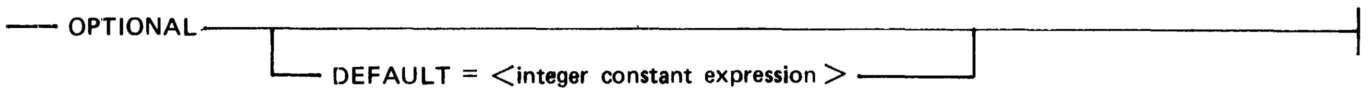


The <identifier> must have been specified in a <job parameter list> with a type of STRING.

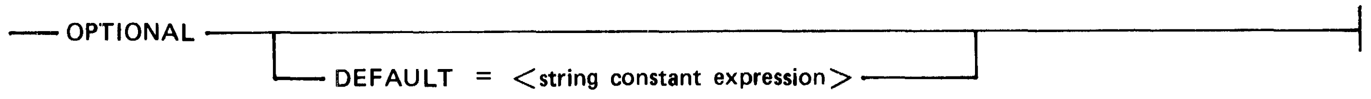
<optional boolean> syntax:



<optional integer> syntax:



<optional string> syntax:



A parameter identifier of the appropriate type may be used in any context where a <boolean constant>, <integer constant>, or <string constant> is allowed. It cannot be assigned a value by using an <assignment statement>. Its value is derived from the corresponding actual parameter in a < start parameter list>.

The keyword OPTIONAL indicates that an expression need not be passed, or provided, in the <start parameter list>. If an expression is not passed, then default values are assigned as follows: FALSE when assigned to a boolean parameter, zero when assigned to an integer parameter, and a zero length string when assigned to a string parameter.

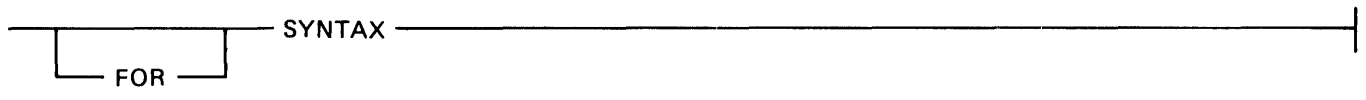
A WFL <job> with parameters must be either started with a matching <start parameter list> parameter for each non-optional parameter, or compiled for SYNTAX without providing a <start parameter list>.

Default values can also be manually specified using the DEFAULT = constant expression branch of the <optional boolean>, <optional integer>, or <optional string> syntax diagrams. The specified default value is ignored if a START parameter is provided.

Job Disposition

The <job disposition> specifies that a job is not to be executed.

<job disposition> syntax:

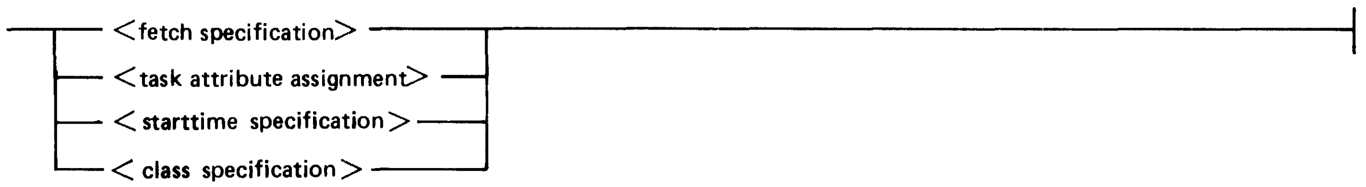


A job with a disposition of SYNTAX is checked for syntax errors. It will not be executed.

JOB ATTRIBUTE SPECIFICATION

A <job attribute specification> is used to assign task attributes to the job task.

<job attribute specification> syntax:



<Task attribute assignment> is described under Task Attribute Assignment in section 5, Task Attributes.

The <task attribute assignment> may contain only constants and must not contain file equations. (A file equation changes the attributes of a file and is described under File Equation in section 6, File Attributes.)

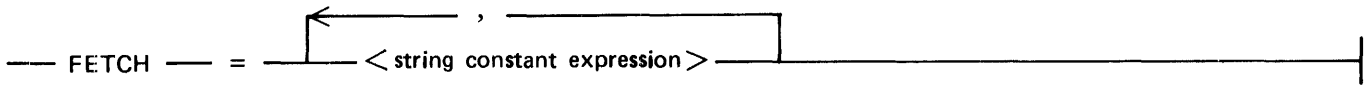
Examples:

```
FETCH = "MOUNT MASTER TAPE"

PRIORITY = 9

STARTTIME = 12:00

CLASS = 90
```

FETCH Specification

A <fetch specification> causes the job to wait for operator action before beginning execution.

<fetch specification> syntax:



A job may contain a maximum of one <fetch specification>. The aggregate length of all the strings must not exceed 1840 characters.

The <fetch specification> informs the operator which resources are required to run the job. When the job is initiated, the operator is informed that there is a job requiring operator action. The operator can display the message using the PF ODT-command. The job task is placed in the waiting schedule until the operator responds with an OK ODT-command. Refer to the *B 1000 Systems System Software Operation Guide, Volume 1,* for more information on the PF and OK commands.

Example:

Disk file ARCHIVE/INVENTORY contains the following job:

```
BEGIN JOB ARCHIVE/INVENTORY;
  FETCH = "PACK PARTS REQUIRED",
    "MOUNT SCRATCH TAPE FOR ARCHIVE";
  COPY INVENTORY/= FROM PARTS TO ARCHIVE (KIND=TAPE);
END JOB
```

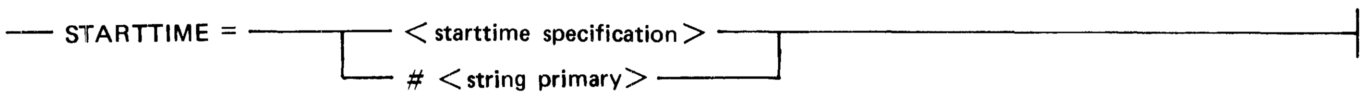
Invocation of the above job:

```
START ARCHIVE/INVENTORY
 JOB 7706 CONTAINS FETCH MESSAGE; "PF" REQUESTED
7706 PF
 7706 FETCH: PACK PARTS REQUIRED, MOUNT SCRATCH TAPE FOR ARCHIVE
7706 OK
 7706 OK-ED
 ARCHIVE/INVENTORY = 7706 BOJ PR=4 TIME=18:07:15.2
```

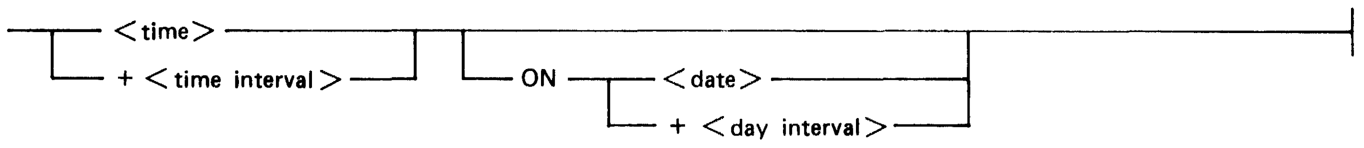
STARTTIME Specification

A <STARTTIME specification> causes the job to wait for a specified amount of time before beginning execution.

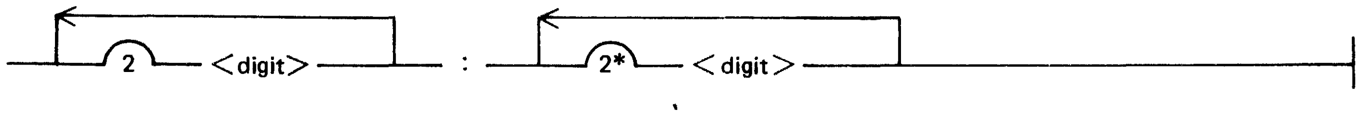
<STARTTIME specification> syntax:



<starttime spec> syntax:

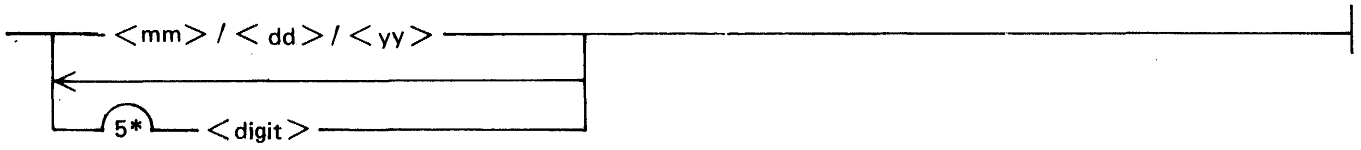


<time> syntax:



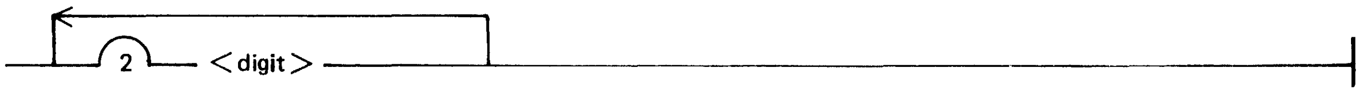
<Time> is the time of day on a 24-hour clock in the form HH:MM. The hours must be less than 24, the minutes must be less than 60.

<date> syntax:

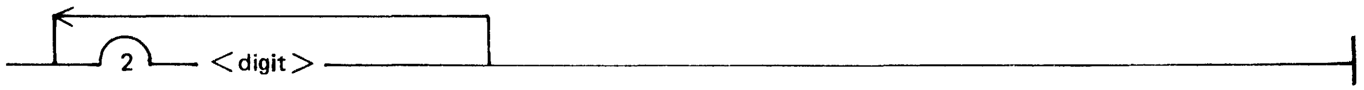


<Date> is in the form MM/DD/YY or YYDDD.

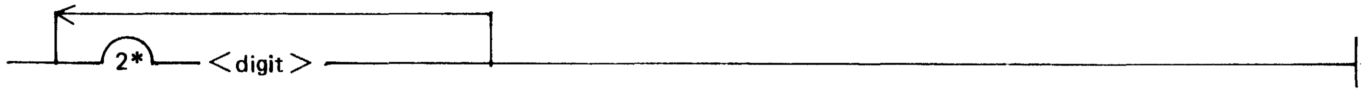
<mm> syntax:



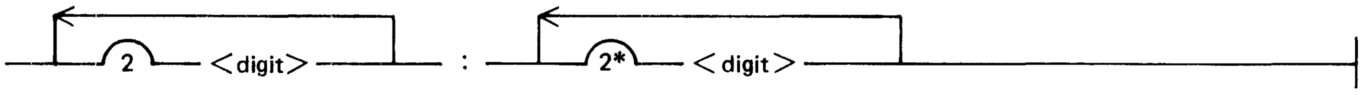
<dd> syntax:



<yy> syntax:

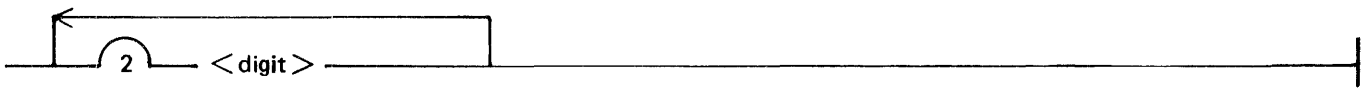


<time interval> syntax:



<Time interval> is of the form HH:MM. The hours must be less than 24, the minutes must be less than 60.

<day interval> syntax:



When a <STARTTIME specification> is included in a <job attribute specification>, the job is handled in the normal fashion, except it is not selected to run until either the current time is greater than or equal to the specified start time or the job is forced from the schedule by way of the FS system command.

The #<string primary> must evaluate to a valid <starttime spec>. #<string primary> is only valid for a START statement within a job.

WFL determines the absolute time and date at which a job should begin execution from the <starttime spec>.

If a <time interval> is specified, that <time interval> is added to the current time.

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

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

Example:

The following job begins execution after 10:00 P.M. on March 20, 1981:

```
BEGIN JOB EXAMPLE1;
  STARTTIME = 22:00 ON 03/20/81;
END JOB
```

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

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

CLASS Specification

A <class specification> assigns the job to a specific job queue.

<class specification> syntax:

```
── CLASS = <integer constant expression> ──────────────────────┤
```

The <class specification> assigns the CLASS task attribute (the number of the queue desired) for the job. All tasks initiated by the job have this class. CLASS values may range between 0 and 1022.

Example:

The following job is assigned to queue 77:

```
BEGIN JOB CLASS/EXAMPLE;
  CLASS = 77;
END JOB
```

Job Declaration List

A <job declaration list> declares variables, subroutines, or both for the entire WFL program.

<job declaration list> syntax:

```
── <declaration list> ─────────────────────────────────────────┤
```

<Declaration list> is described in section 4, Declarations.

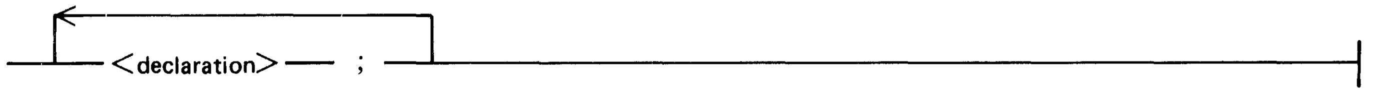
SECTION 4 DECLARATIONS

A <declaration> creates a variable and defines its type.

DECLARATION LIST

A <declaration list> is a list of one or more declarations.

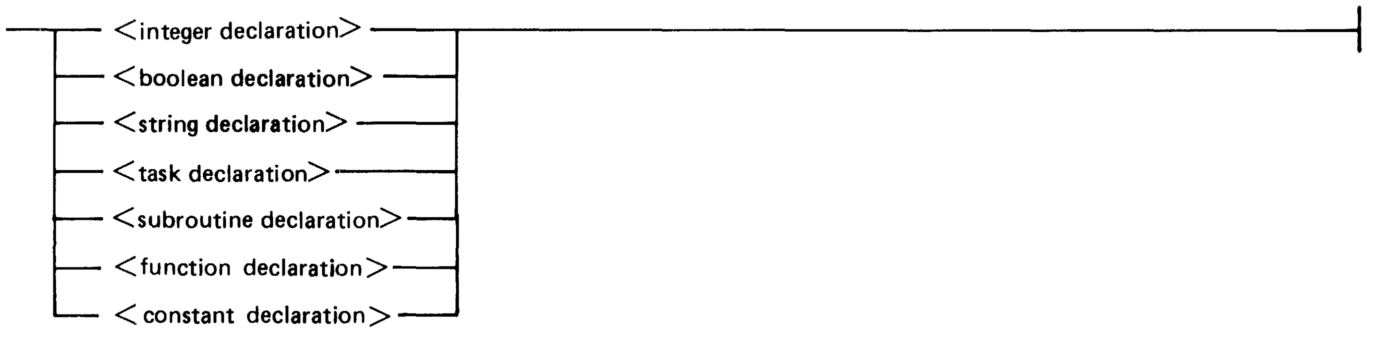
<declaration list> syntax:



DECLARATION

A <declaration> allocates a variable and associates an <identifier> with that variable.

<declaration> syntax:



All variables must be explicitly declared before they are used. The use of a variable must be consistent with its declaration.

All declarations within a job must follow the <job attribute specification> list and precede any executable statement within the job. All declarations within a WFL subroutine or function must precede any executable statement within the subroutine or function. Declarations may occur in any order.

VARIABLE DECLARATIONS

A variable declaration creates a new variable, gives it a type, and, optionally, an initial value.

Initial Value

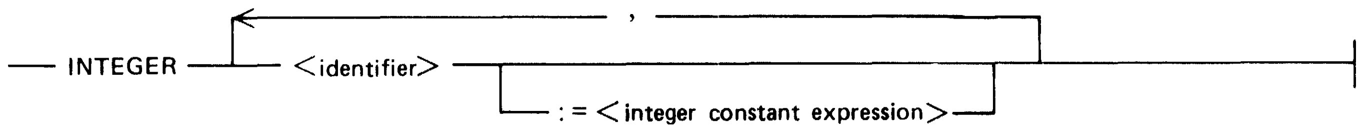
Variables can be assigned an initial value by using the assignment operator (:=) after the < identifier> which is being declared. (Task variables can be assigned an initial value by giving a list of <task attribute assignment>s or <file equation>s in parentheses.)

Variables which have not been assigned an initial value will have an unpredictable default value. Care must be taken to assign a value to variables before they are used.

Integer Declaration

Integer variables contain the value of <integer expression>s. (<Integer expression> is described in section 7, Expressions.)

<integer declaration> syntax:

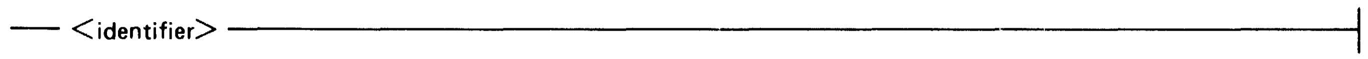


Examples:

INTEGER A2D

INTEGER X := 128, Y := 32 * 8

<integer id> syntax:

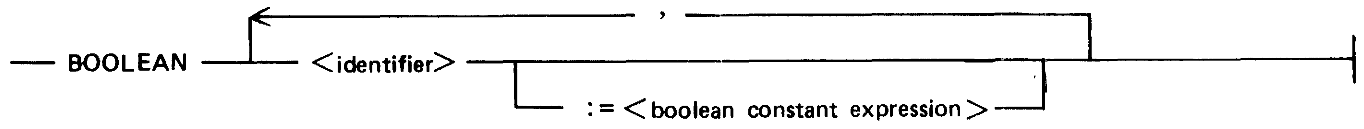


The <identifier> must have been specified in either an <integer declaration> or a <specified parameters> declaration.

Boolean Declaration

Boolean variables contain the value of <boolean expression>s. (<Boolean expression> is described in section 7, Expressions.)

<boolean declaration> syntax:

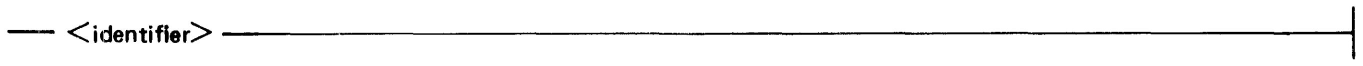


Examples:

BOOLEAN B

BOOLEAN T := TRUE, F := FALSE, B1 := TRUE OR FALSE

<boolean id> syntax:

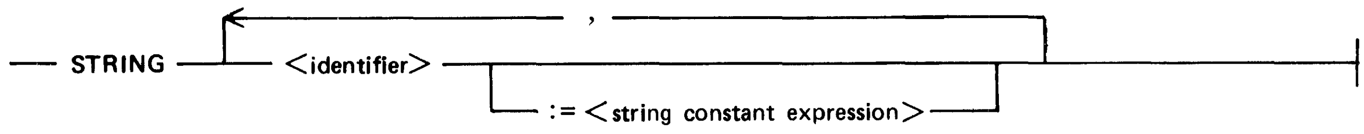


The <identifier> must have been specified in either a <boolean declaration> or a <specified parameters> declaration.

String Declaration

String variables contain the value of <string expression>s. (<String expression> is described in section 7, Expressions.)

<string declaration> syntax:



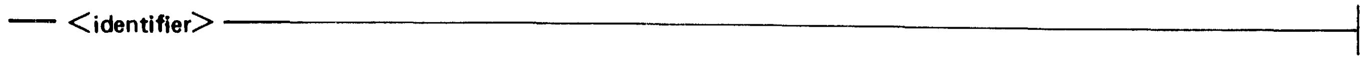
The length of the last <string expression> assigned to a string variable is retained by that variable.

Examples:

STRING S

STRING STR := "THIS IS A STRING" , S1 := "A STRING" & "CONSTANT EXPRESSION"

<string id> syntax:

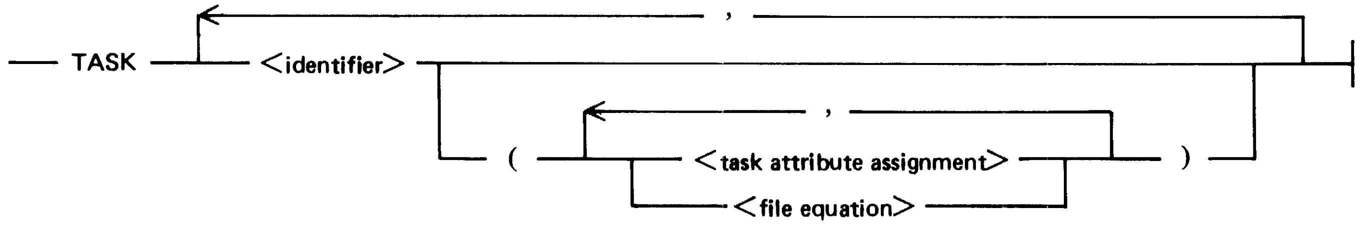


The <identifier> must have been specified in either a <string declaration> or a <specified parameters> declaration.

Task Declaration

Task variables contain the value of task and file attributes.

<task declaration> syntax:



Task variables are initialized (as though they were used in an <INITIALIZE statement>) when they are declared.

Examples of task declarations:

```
TASK T1ZX

TASK A (PRIORITY = 3, PROTECTED)

TASK A,B (PRIORITY = 7), C
```

<task id> syntax:

The <identifier> must have been specified in either a <task declaration> or a <specified parameters> declaration.

MYSELF and MYJOB are predeclared task variables. They are used in exactly the same way as any <task id> except that they may not be assigned to a task because their assignment is already implied.

MYJOB is a task variable which provides access to the values of the task attributes of the job.

Example:

```
MYJOB (PRIORITY = 9); % Sets the priority value for the job.
```

MYSELF is a task variable which provides access to the values of the task attributes of the task making the request.

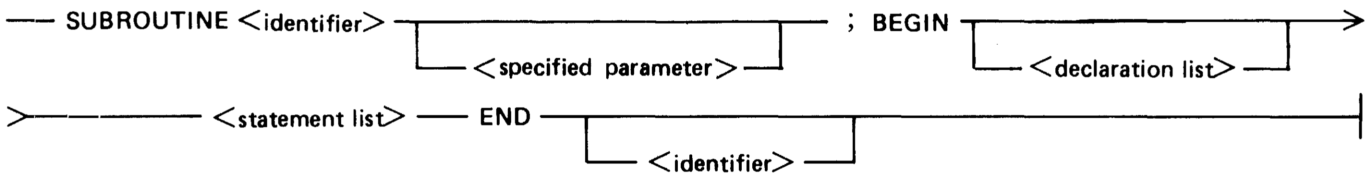
Examples:

```
I := MYSELF (MIXNUMBER)   % Stores the mix number of the task
                          % in the integer variable I.

MYSELF (PRIORITY = 9);    % Sets the priority of the task.
```

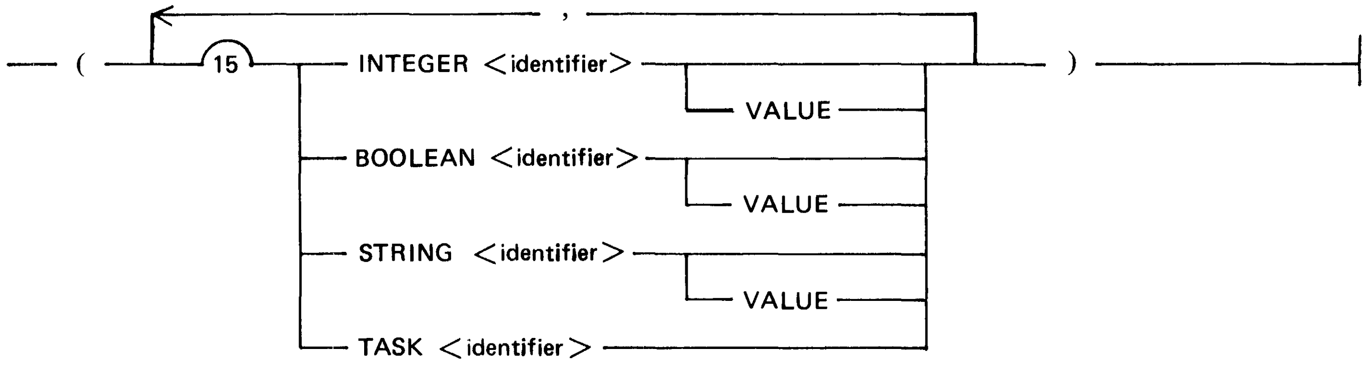
SUBROUTINE DECLARATION

A subroutine declares a block internal to the job (a procedure) which can be activated by a <subroutine invocation statement>.

<subroutine declaration> syntax:



<specified parameter> syntax:



Subroutine and function declarations may be nested to a combined depth of 10. No limit exists on non-nested subroutines.

There is a limit to the number of <statement> s in the <statement list> of a < subroutine declaration>. Refer to section 3, JOB, for a discussion of the limit.

The type of each parameter is specified by the key word preceding the name of the parameter. The key word VALUE indicates that the parameter is "call by value" rather than " call by reference."

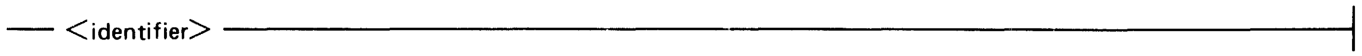
For a parameter which is "call by value," any changes made to the parameter within the subroutine are not reflected in the variable which was passed to the subroutine. For a parameter which is "call by reference," all changes made to the parameter within the subroutine also change the value of the variable which was passed to the subroutine.

The same scope rules apply to the name of parameters as apply to local variables. Note that declarations within a subroutine may not declare an identifier whose name is the same as any of the parameters to that routine.

Example subroutine declaration:

```
BEGIN JOB EXAMPLE SYNTAX;
  % declarations
  TASK T;
  BOOLEAN B;
  INTEGER I;
  SUBROUTINE SUB (INTEGER X);
  BEGIN
    RUN X; FILE F1(KIND=DISK, MAXRECSIZE=30);
      % initiates program X,
      % the file with internal name F1
      % will be assigned to a DISK
      % and have a maximum record size of 30.
    RUN Y;
    I := I + 1;
  END; % end of subroutine declaration
  %
  RUN Z; % this is the first executable statement
  I:=0; % initiates the subroutine
  SUB(I);
  B:= TRUE;
  RUN X [T];
END JOB
```

<subroutine id> syntax:

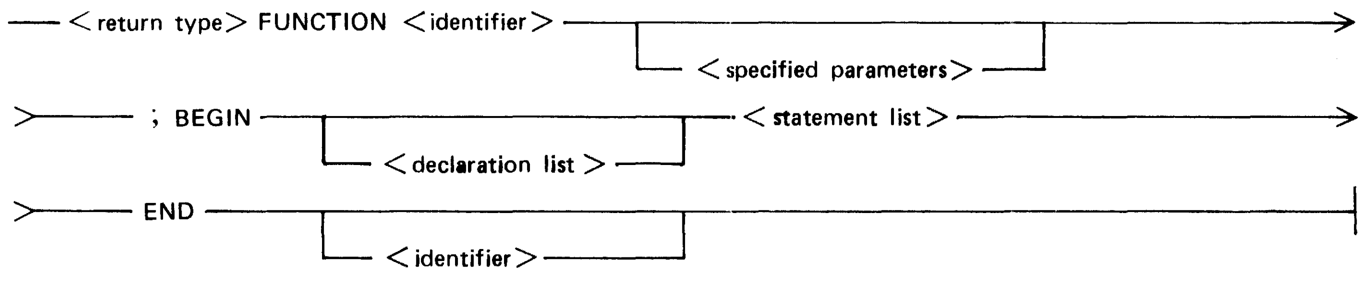


The <identifier> must have been specified in a <subroutine declaration>.

FUNCTION DECLARATION

A function declares a block internal to the job (a procedure) which can be activated by <function invocation>.

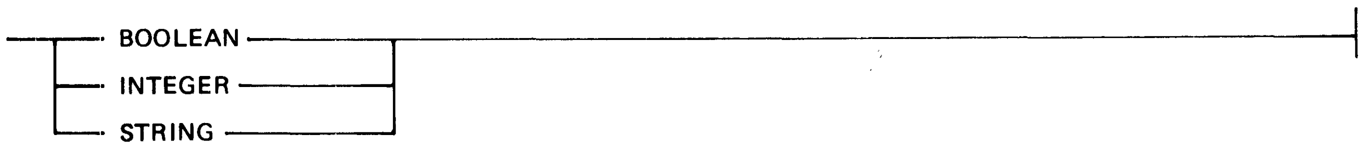
<function declaration> syntax:



Function and subroutine declarations may be nested to a combined depth of 10. No limit exists on non-nested functions.

There is a limit to the number of <statement> s in the <statement list> of a < function declaration>. Refer to section 3, JOB, for a discussion of the limit.

<return type> syntax:



The <return type> is the type of the expression returned by the function. It is a syntax error if the <RETURN statement> from a function does not contain an expression, if the expression returned has a different type from the specified <return type>, or if a <RETURN statement> has not been encountered by the time the end of <FUNCTION declaration> is reached. Not returning an expression, that is, reaching the END statement of the function during execution, results in a run time error.

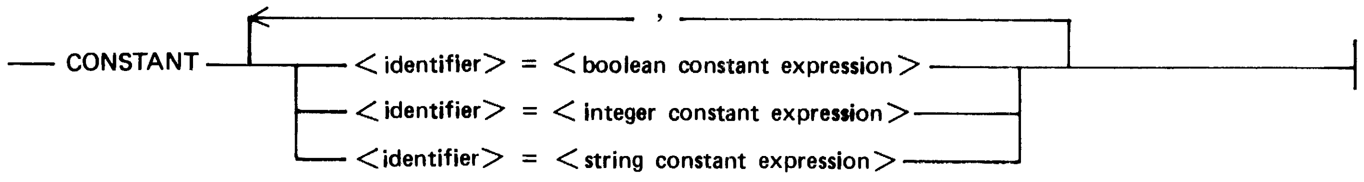
Example:

```
BOOLEAN FUNCTION FUNC1 (INTEGER LEN, STRING STR);
  BEGIN
    IF LENGTH (STR) EQL LEN THEN
      RETURN TRUE;
    ELSE
      RETURN FALSE;
  END FUNC1;

BOOLEAN FUNCTION FUNC2 (INTEGER LEN, STRING STR);
  BEGIN
    RETURN LENGTH (STR) EQL LEN
  END FUNC2;
```

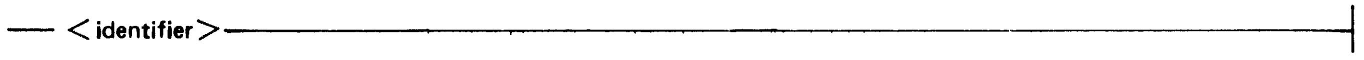
CONSTANT DECLARATION

A constant identifier contains the value of the evaluated expression, and can be used anywhere a constant of the same type may be used.

<constant declaration> syntax:

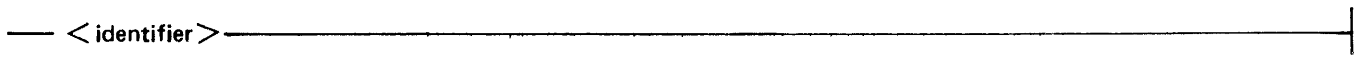


<boolean constant id> syntax:



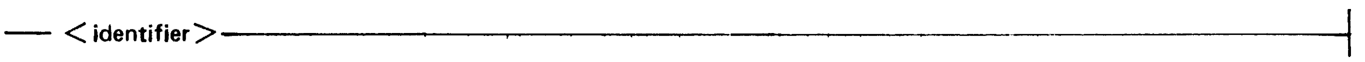
The <identifier> must have been specified in a <constant declaration> and assigned a <boolean constant expression>.

<integer constant id> syntax:



The <identifier> must have been specified in a <constant declaration> and assigned an <integer constant expression>.

<string constant id> syntax:



The <identifier> must have been specified in a <constant declaration> and assigned a <string constant expression>.

A constant identifier of the appropriate type may be used in any context where a <boolean constant>, <integer constant>, or <string constant> is allowed. It cannot be assigned a value by using an <assignment statement>.

Examples:

```
CONSTANT
  TISTURE       =    TRUE AND TRUE,
  SEVEN         =    12 - 5,
  HOSTNAME      =    "SB" & "P "
```

Scope of Variables

Declarations may occur either at the job level (globals) or within WFL subroutines. All declarations within a WFL subroutine specify variables that are local to that subroutine.

WFL is a block-structured language, meaning that blocks may be nested inside other blocks. In WFL, a block is the job (the global block) or any subroutine. A block includes any blocks declared within it. Block structure allows the same <identifier> to be used in different blocks to denote different items, even though one block is nested within another.

Consider the following example:

```
BEGIN JOB B1;
  INTEGER I,J;
  SUBROUTINE B2;
    BEGIN
      INTEGER J;
      I:=1;  %REFERS TO I INSIDE B1
      J:=2;  %REFERS TO J INSIDE B2
             %K HAS NOT BEEN DECLARED YET
             %SO IT CAN NOT BE REFERENCED
             %EVEN THOUGH IT IS DECLARED IN THE GLOBAL BLOCK
    END;
  INTEGER K;
  SUBROUTINE B3;
    BEGIN
      INTEGER I;
      SUBROUTINE B4;
        BEGIN
          INTEGER J;
          I:=1;  %REFERS TO I INSIDE B3
          J:=2;  %REFERS TO J INSIDE B4
          K:=3;  %REFERS TO K INSIDE B1
        END;
      B4
    END;
  I:=1;  %REFERS TO I INSIDE B1
  J:=2;  %REFERS TO J INSIDE B1
  K:=3;  %REFERS TO K INSIDE B1
END JOB
```

An <identifier> declared within block B1 can be used only within block B1. If another block B2 is declared within B1, then any <identifier> declared in B1 can be used inside B2, unless an <identifier> spelled the same is declared in B2.

SECTION 5 TASK ATTRIBUTES

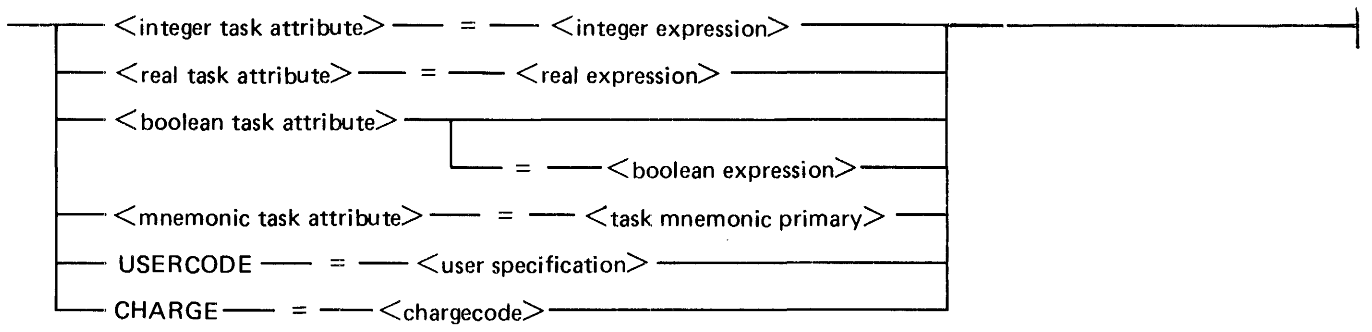
Task attributes allow the job to monitor and control the execution of tasks.

Appendix A lists all attributes implemented in B 1000 WFL, the allowed use (Read only, Read/Write), and contains a description of the function of each attribute.

TASK ATTRIBUTE ASSIGNMENT

A <task attribute assignment> sets the value of an attribute within a task variable or a task.

<task attribute assignment> syntax:


<integer task attribute> = <integer expression>
<real task attribute> = <real expression>
<boolean task attribute>
= <boolean expression>
<mnemonic task attribute> = <task mnemonic primary>
USERCODE = <user specification>
CHARGE = <chargecode>

Misusing an attribute (for example, attempting to set USERCODE to an invalid <user specification>) results in a fatal run-time error.

<Integer expression>, <real expression>, and <boolean expression> are described in section 7, Expressions.

Semantics:

<integer task attribute>
Any task attribute of type Integer.

<real task attribute>
Any task attribute of type Real.

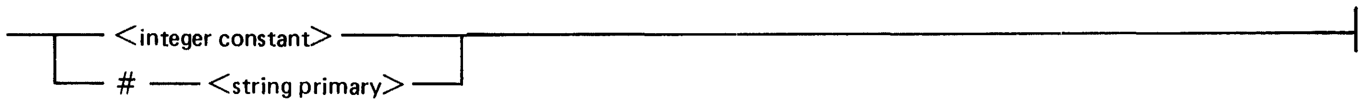
<boolean task attribute>
Any task attribute of type Boolean.

<mnemonic task attribute>
Any task attribute of type Mnemonic.

CHARGECODE

<Chargecode> is a user assigned code and is used by the logging function of the MCP.

<chargecode> syntax:



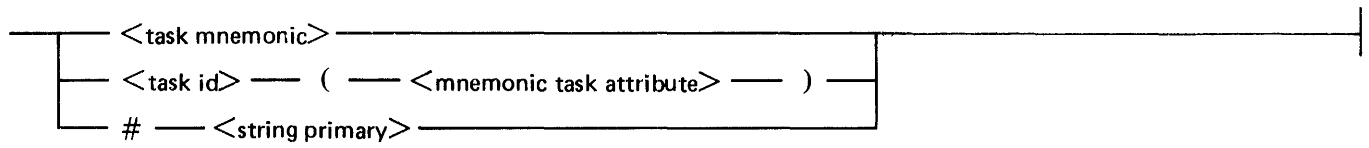
A <chargecode> must be numeric and no greater than seven digits in length, otherwise a run-time error will occur.

<String primary> is described in section 7, Expressions.

TASK MNEMONIC PRIMARY

A <task mnemonic primary> is used to compute the value of <mnemonic task attribute>s.

<task mnemonic primary> syntax:



Semantics:

<task mnemonic>
Any member of the set of named constants associated with a mnemonic task attribute.

<String primary> is described in section 7, Expressions.

<Task mnemonic primary> allows mnemonic valued attributes to be used in comparisons and assignments. It is important to note that the specific mnemonics must be compatible. Two attributes are compatible if any of their mnemonic values are the same. An attribute and a mnemonic are compatible if the mnemonic is a valid value for that attribute.

Example:

The manner in which task attributes are used depends on the particular application. The following job includes examples of how task attributes might be used to control the execution of tasks within the job.

```
BEGIN JOB SAMPLE;
%
  TASK A, B;
  A (MAXTIME = 30);
  B (PRIORITY = 8);
%
  RUN TEST1 [A];
% Will be DSed if it exceeds 30 seconds elapsed time
%
  IF A IS COMPLETEDOK THEN % if TEST1 completed normally
    BEGIN
      COMPILE BNOR WITH COBOL [B];
        COMPILER FILE CARDS (TITLE = COBFILE, DISK);
        COMPILER MAXTIME = 600; % elapsed time limit for COBOL
      RUN TEST2 [A]; FILE TINPUT (TITLE = X/Y);
    END;
%
END JOB
```

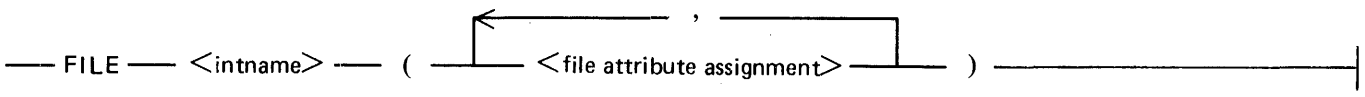
SECTION 6 FILE ATTRIBUTES

Each file attribute defines a characteristic of the file. Changing the state of a single attribute changes the characteristic of the file defined by that attribute and may cause the states of other attributes to be changed. Further explanation of each individual attribute is contained in the attribute descriptions in appendix B.

FILE EQUATION

A <file equation> specifies changes to the attributes of a file declared in a program to which it is applied.

<file equation> syntax:



The <file equation> is applied to the file that is declared with an identifier (in the object program) the same as the <intname>. If the object program to which the <file equation> is applied did not declare a file with an identifier the same as <intname>, the program will fail initiation.

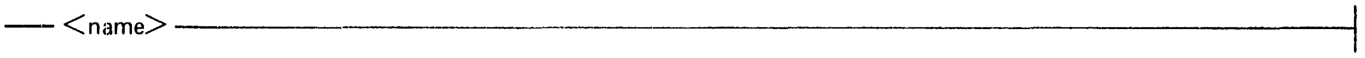
If more than one <file equation> is specified for the same <intname>, the file attributes are merged and the last value for a given attribute is used.

When a file is opened, file attributes are set in the same order as task attributes. Refer to Task Attributes and File Equation in section 8, Statements, for additional information.

INTNAME

An <intname> is the name given to a logical file when it was declared in a program (COBOL FD file-name, RPG F specification filename, and so forth).

<intname> syntax:



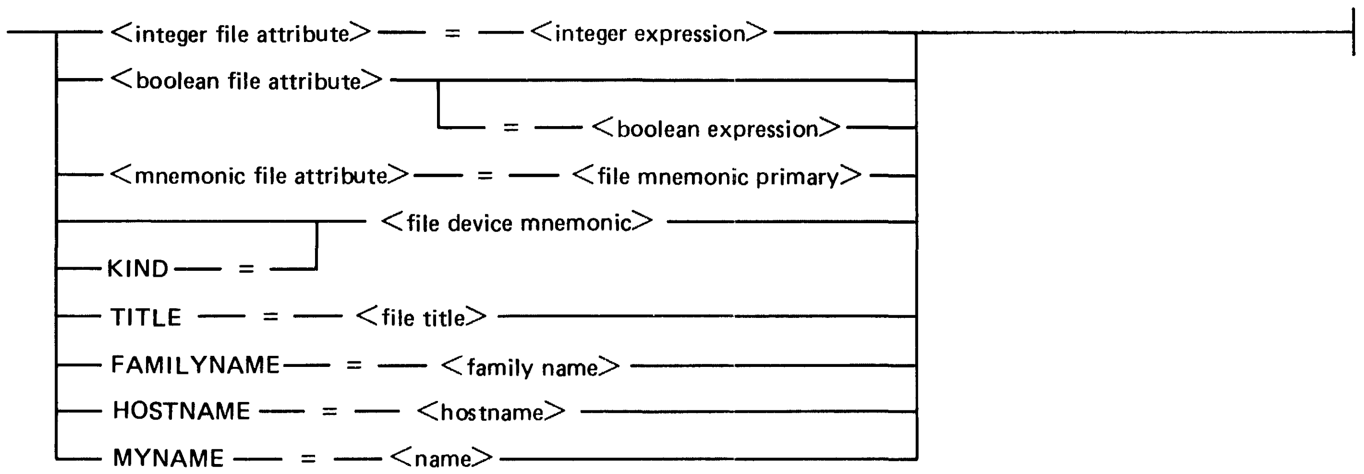
An <intname> may have an unrestricted number of characters, but only the first ten (10) characters are used to match the logical file as declared in the program.

NOTE
Users are strongly encouraged to use all the characters of the logical file name when forming an <intname> for compatibility with other systems.

FILE ATTRIBUTE ASSIGNMENT

A <file attribute assignment> is used to assign a value to a file attribute.

<file attribute assignment> syntax:



Semantics:

<integer file attribute>
    Any file attribute of type Integer.

<boolean file attribute>
    Any file attribute of type Boolean.

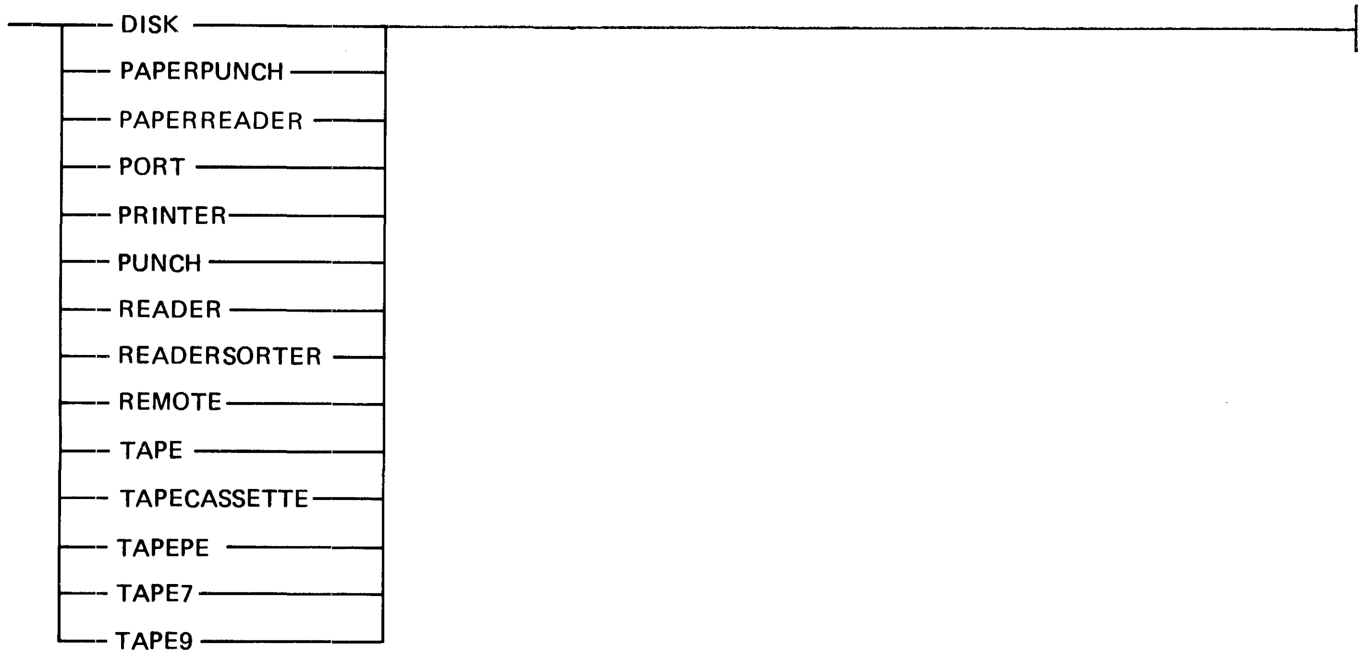
<mnemonic file attribute>
    Any file attribute of type Mnemonic.

Misusing an attribute (for example, attempting to set TITLE to an invalid <title> using a <string expression>) results in a fatal run-time error.

FILE DEVICE MNEMONICS

A <file device mnemonic> is a value which is compatible with the KIND mnemonic file attribute.

<file device mnemonic> syntax:

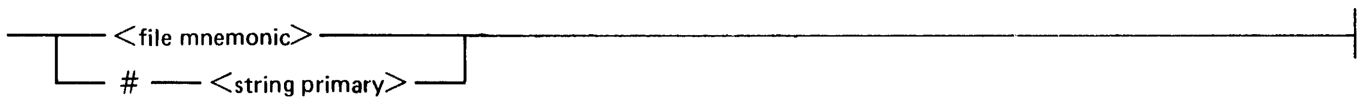


For more information see the description of the KIND file attribute in appendix B.

FILE MNEMONIC PRIMARY

A <file mnemonic primary> is used to compute the value of <mnemonic file attribute>s.

<file mnemonic primary> syntax:



Semantics:

<file mnemonic>
Any member of the set of named constants associated with a mnemonic file attribute.

<String primary> is described in section 7, Expressions.

<File mnemonic primary> allows mnemonic valued attributes to be used in comparisons and assignments. It is important to note that in mnemonic attribute comparisons and assignments, the specific attributes and mnemonics must be compatible. Two attributes are compatible if any of their mnemonic values are the same. For example, the MYUSE attribute of one file (mnemonics IN, OUT, or IO) may be compared with or assigned to the MYUSE attribute or the OTHERUSE attribute (mnemonics SECURED, IN, OUT, IO) of the same or some other file identifier. The MYUSE and KIND file attributes are not compatible because none of their mnemonic values are the same. An attribute and a mnemonic are compatible if the mnemonic is a valid value for that attribute.

Examples:

```
RUN P;
  F1 (KIND = DISK)      % Syntactically correct.
  F1 (KIND = DONTCARE)  % Incorrect because DONTCARE is
                        % not a valid mnemonic for the file
                        % attribute KIND.

RUN X;
 FILE F1 (TITLE = X/Y, KIND = DISK);
```

In this example, when program X opens file F1, the physical file associated with F1 will be a disk file called "X/Y".

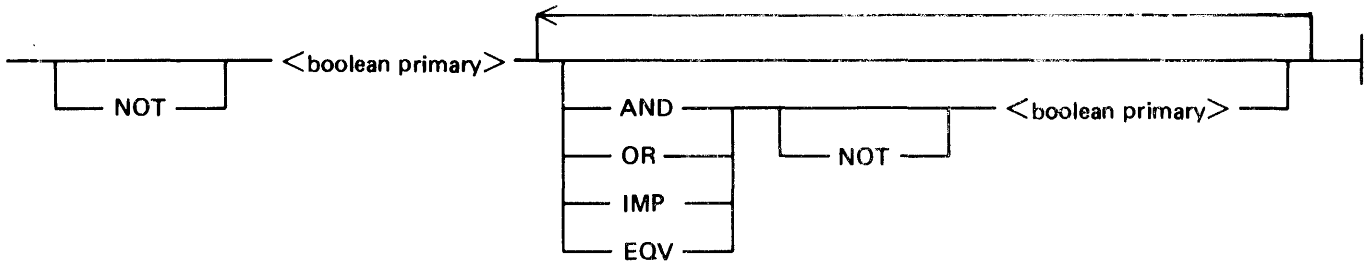
SECTION 7 EXPRESSIONS

Expressions are used to compute values by applying various operators to variables, functions, and subexpressions (primaries).

BOOLEAN EXPRESSION

A <boolean expression> is used to compute logical values.

<boolean expression> syntax:



Examples:

NOT B1
B1 AND NOT B2
P IMP Q AND B1 EQV NOT B2 OR B3

Logical Operators

Truth Table

A	B	NOT A	A AND B	A OR B	A IMP B	A EQV B
FALSE	FALSE	TRUE	FALSE	FALSE	TRUE	TRUE
FALSE	TRUE	TRUE	FALSE	TRUE	TRUE	FALSE
TRUE	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE
TRUE	TRUE	FALSE	TRUE	TRUE	TRUE	TRUE

Order of Evaluation

The order of precedence (highest first) for the execution of logical operators is as follows:

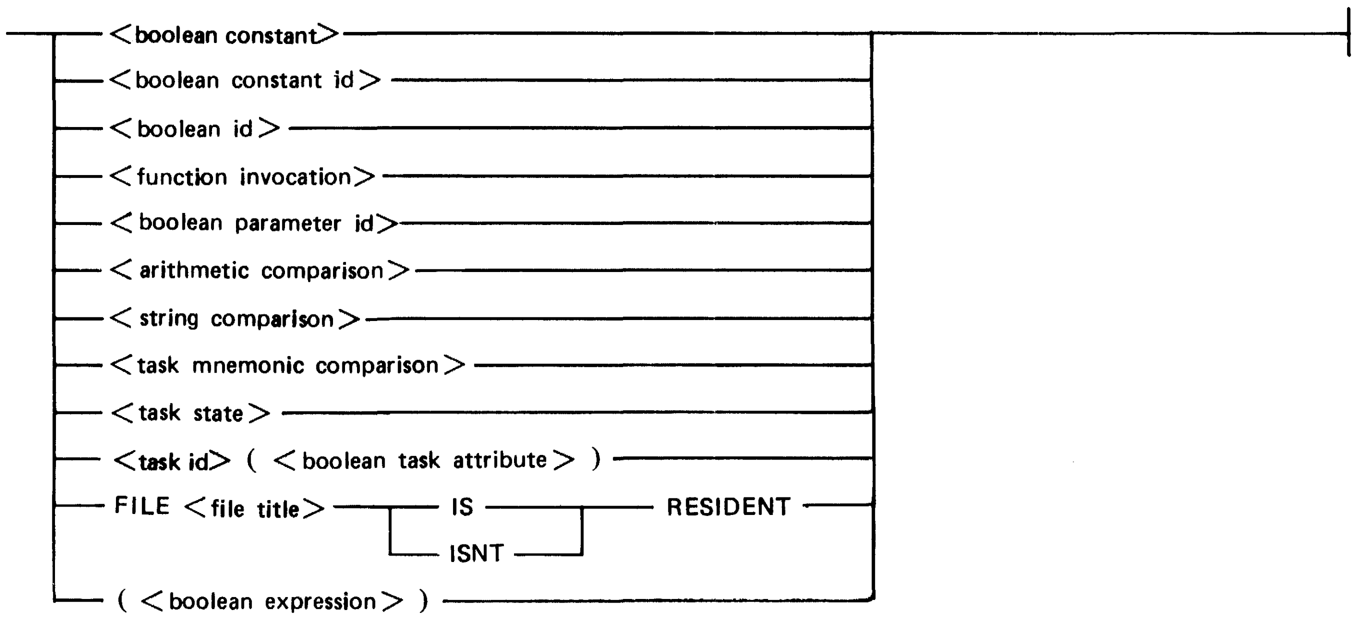
1. <boolean primary>
2. NOT
3. AND
4. OR
5. IMP
6. EQV

First, all <boolean primary>s are evaluated. Second, the NOT operators are applied to the <boolean primary> that they precede. Finally, the operations are performed in the appropriate order of priority. If two operations have the same priority, the leftmost operation is performed first. Note that a <boolean expression> enclosed in parentheses becomes a <boolean primary>.

Boolean Primary

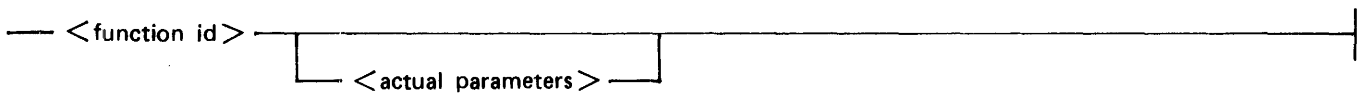
A <boolean primary> represents a logical value.

<boolean primary> syntax:



FUNCTION INVOCATION

<function invocation> syntax:



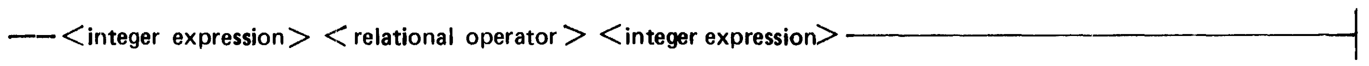
The <function invocation> causes the function denoted by <function id> to be executed. See Subroutine Invocation Statement in section 8 for a description of <actual parameters>.

If a <function invocation> is used as a <boolean primary>, the function must have a <return type> of BOOLEAN.

Arithmetic Comparison

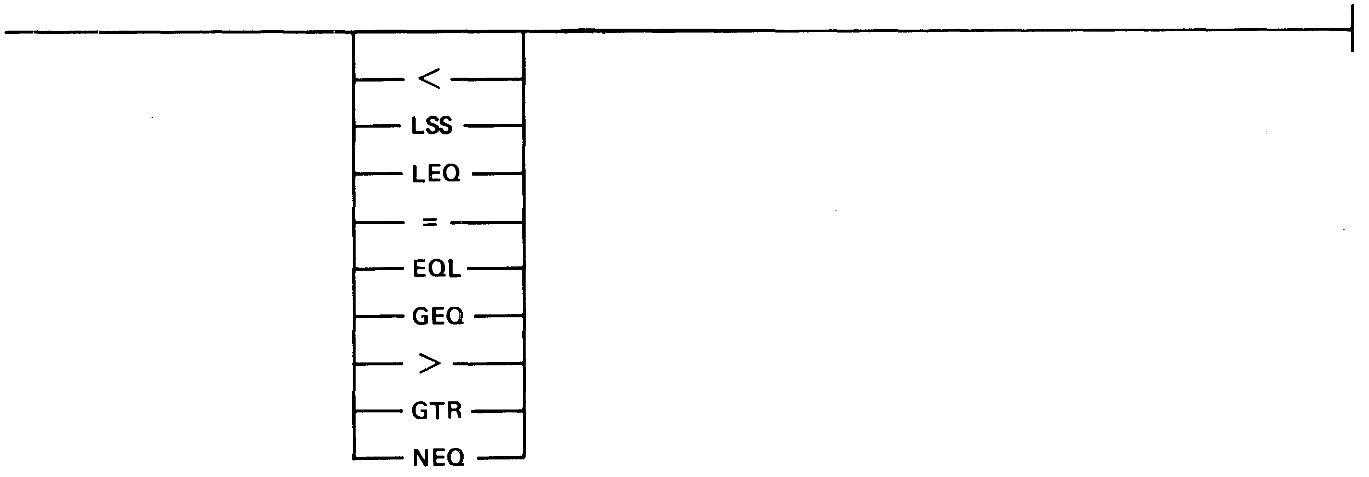
An <arithmetic comparison> allows the values of two <integer expressions> to be compared.

<arithmetic comparison> syntax:



Relational Operators

<relational operator> syntax:



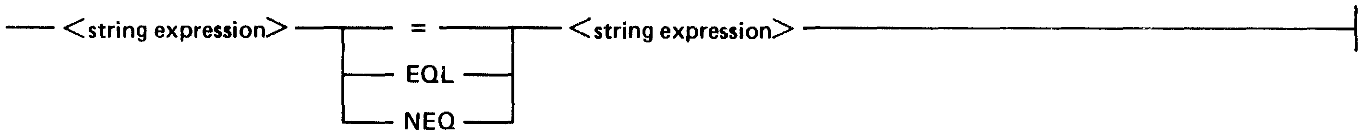
Relational operators perform a comparison between two operands and produce a boolean result.

Operator	Function
<	less than
LSS	less than
LEQ	less than or equal
=	equal
EQL	equal
GEQ	greater than or equal
>	greater than
GTR	greater than
NEQ	not equal

String Comparison

A <string comparison> allows the value of two <string expression>s to be compared.

<string comparison> syntax:

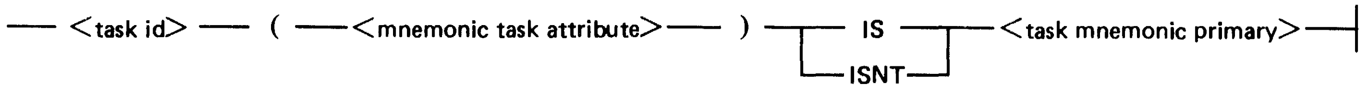


Two strings are equal only if all characters in the first string occur in the same order in the second string and the lengths of the two strings are equal.

Task Mnemonic Comparison

A <task mnemonic comparison> allows the comparison of a <mnemonic task attribute> to a <task mnemonic primary>.

<task mnemonic comparison> syntax:



Examples:

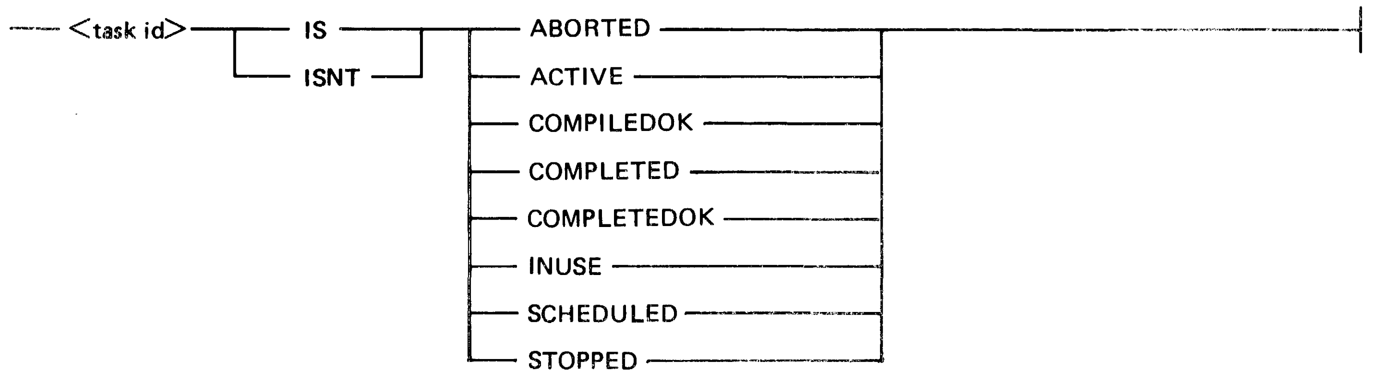
T(JOBSUMMARY) IS SUPPRESSED

T1(JOBSUMMARY) ISNT T2(JOBSUMMARY)

Task State

<Task state> allows the status of a task to be monitored.

<task state> syntax:



Semantics:

ABORTED
: The task failed initiation or was abnormally terminated with a DP or DS system command.

ACTIVE
: The task is currently running.

COMPILEDOK
: A compiler task completed without detecting syntax errors.

COMPLETED
: The task was initiated and has terminated. A task id that fails initiation is in the COMPLETED state. A task id not used in a task initiation statement since it was declared or used in an INITIALIZE statement, is not in any of the <task state> syntax states.

COMPLETEDOK
: The task completed and was terminated without faulting or being aborted.

INUSE
: The task is in the SCHEDULED, ACTIVE, or STOPPED state.

SCHEDULED
: The task has not yet been entered in the mix.

STOPPED
: The task has been stopped by the operator, suspended by the system, or programmatically suspended.

File Residency Test

The test for file resident returns TRUE if the permanent file exists for the current user. It does not cause the logical file to be opened or the job to be suspended.

Examples:

```
FILE *COBOL IS RESIDENT

FILE (ZOT)XYZ ON MYPACK ISNT RESIDENT
```

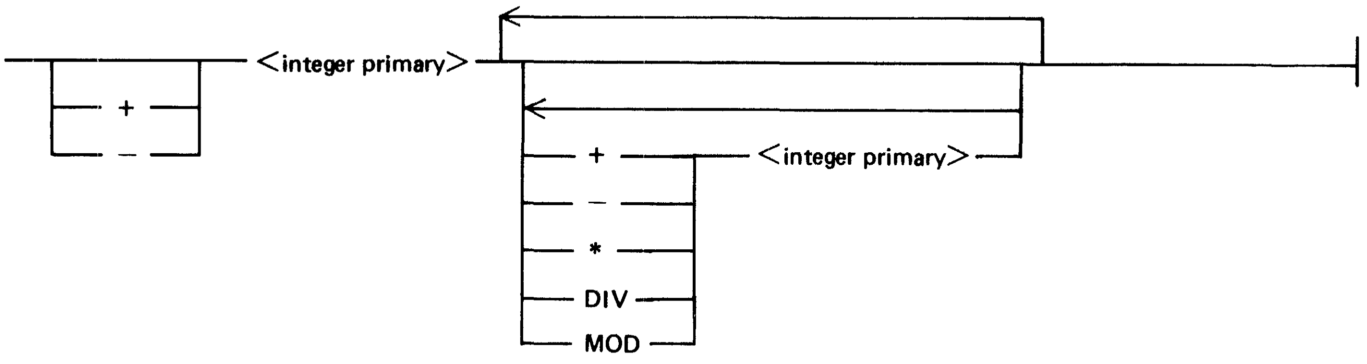
INTEGER AND REAL EXPRESSIONS

Integer and Real expressions yield numerical values by combining primaries with arithmetic operators.

Integer Expression

An <integer expression> is used to compute an integer value.

<integer expression> syntax:



Arithmetic Operators

The operators +, −, and * have the conventional mathematical meanings of addition, subtraction, and multiplication, respectively. The real division operator (/) is not implemented. The DIV operator produces a quotient with a truncated fractional part. The MOD operator returns the remainder of a divide operation.

Order of Evaluation

The order of precedence (highest first) for the execution of arithmetic operators is as follows:

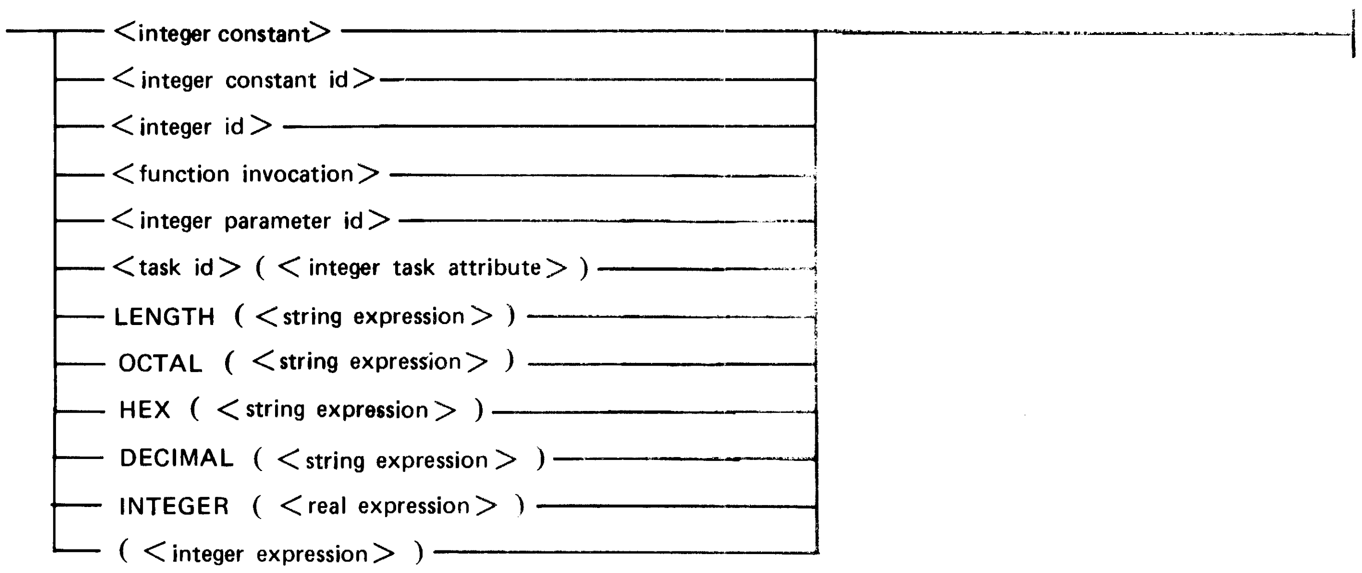
1. <Integer primary>
2. Prefix + or -
3. *, DIV, or MOD
4. Infix + or -

First, all primaries are evaluated. Second, the prefix operator + or –, if any, is applied to the primary that it precedes. Finally, the operations are performed in the order of priority. If two operations are of the same priority, the left operation is performed first. Note that when an expression is enclosed in parentheses, it becomes a primary.

Integer Primary

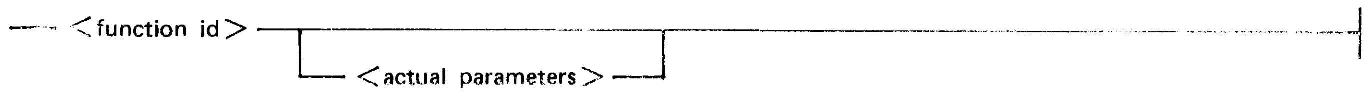
An <integer primary> represents an integer number.

<integer primary> syntax:


<integer constant>
<integer constant id>
<integer id>
<function invocation>
<integer parameter id>
<task id> (<integer task attribute>)
LENGTH (<string expression>)
OCTAL (<string expression>)
HEX (<string expression>)
DECIMAL (<string expression>)
INTEGER (<real expression>)
(<integer expression>)

Function Invocation

<function invocation> syntax:



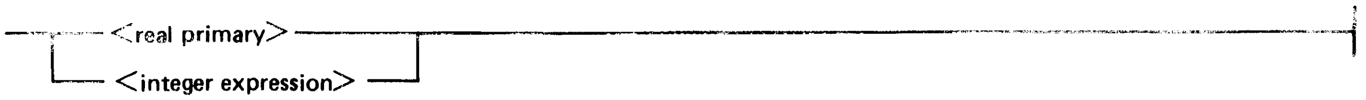
The <function invocation> causes the function denoted by <function id> to be executed. See <subroutine invocation statement> for a description of <actual parameters>.

If a <function invocation> is used as an <integer primary>, the function must have a <return type> equal to INTEGER.

Real Expression

B 1000 WFL supports only integer arithmetic, it does not support real arithmetic. Certain task attributes are of type real and may be used as a real expression. In order to perform arithmetic on a real number, the INTEGER function must be used to convert the real number to an <integer primary>.

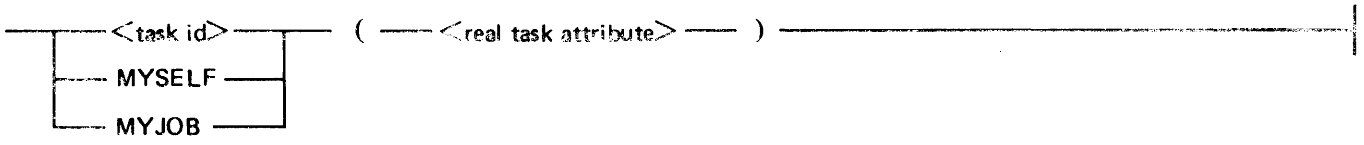
<real expression> syntax:



Real Primary

A <real primary> represents a real number.

<Real primary> syntax:



LENGTH Function

The LENGTH function returns the number of characters within the value of <string expression>.

OCTAL Function

The OCTAL function returns an integer value equal to the octal (base 8) number represented by the value of <string expression>. <String expression> must contain at least one character and not more than eight characters and must not be larger than "37777777". All the characters within the value of <string expression> must be within the set of characters "01234567". A run-time error is given if <string expression> does not satisfy these requirements.

HEX Function

The HEX function returns an integer value equal to the hexadecimal (base 16) number represented by the value of <string expression>. <Sting expression> must contain at least one character and not more than six characters and must not be larger than "7FFFFF". All the characters within the value of <string expression> must be within the set of characters "0123456789ABCDEF". A run-time error is given if <string expression> does not satisfy these requirements.

DECIMAL Function

The DECIMAL function returns an integer value equal to the decimal (base 10) number represented by the value of <string expression>. <String expression> must contain at least one character and not more than seven characters and must not be larger than "8388607". All the characters within the value of <string expression> must be within the set of characters "0123456789". A run-time error is given if <string expression> does not satisfy these requirements.

INTEGER Function

The INTEGER function returns the <real expression> without a fractional part.

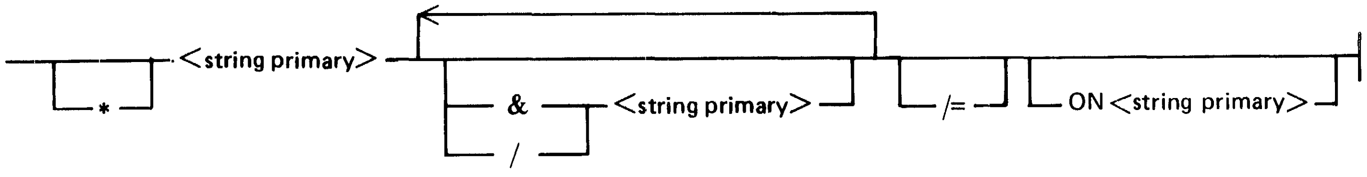
Examples:

Function	Result
INTEGER (MYSELF (ELAPSEDTIME))	The number of whole seconds the WFL program has been running.
LENGTH ("ABCDEF")	6
OCTAL ("10")	8
OCTAL ("377")	255
HEX ("10")	16
HEX ("FF")	255
DECIMAL ("10")	10
DECIMAL ("255")	255

STRING EXPRESSION

A <string expression> is used to compute a string value.

<string expression> syntax:



Concatenation Operation

The concatenate operator (&) is applied to two strings to produce a new string. The length of the new string is the sum of the lengths of the two original strings. The value of the new string is formed by joining a copy of the second string immediately onto the end of a copy of the first string.

Other Concatenation Operators

Four other string concatenation operators are provided. These operators make it easier to build <file title>s from strings, although they are not limited to this purpose. The operators are:

*	unary prefix asterisk
/	binary infix slash
ON	binary infix ON
/=	unary postfix slash-equal

The action of these operators is described in terms of the string concatenation operator (&); s1 and s2 represent any possible <string primary>s:

*s1	is equivalent to	"*" & s1
s1/s2	is equivalent to	s1 & "/" & s2
s1 ON s2	is equivalent to	s1 & " ON " & s2
s1/=	is equivalent to	s1 & "/="

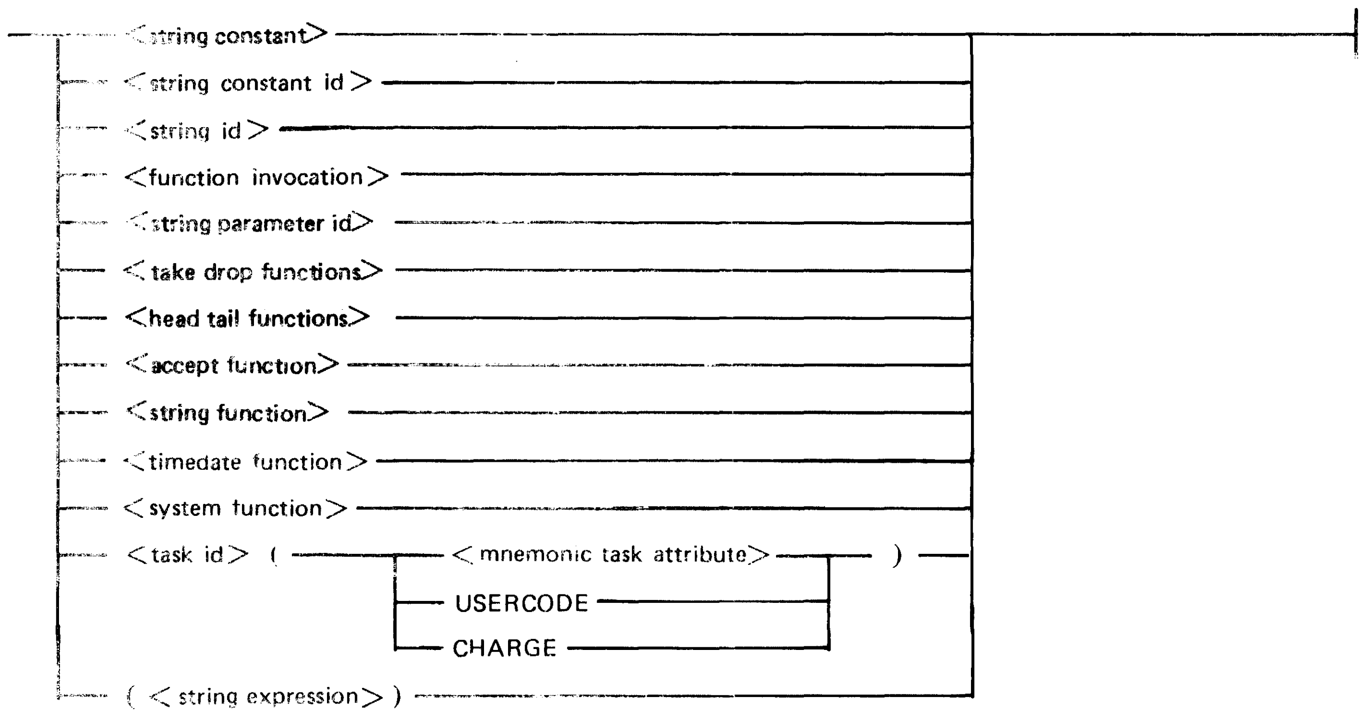
String Overflow

An attempt to evaluate a <string expression> with a length greater then 255 characters causes a run-time error.

String Primary

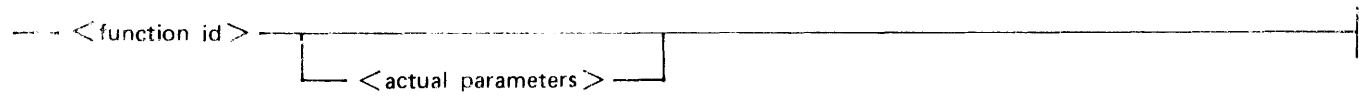
A <string primary> represents a string.

<string primary> syntax:



FUNCTION INVOCATION

<function invocation> syntax:



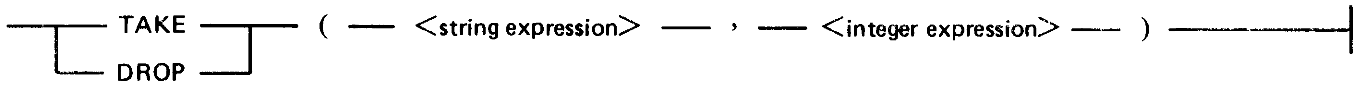
The <function invocation> causes the function denoted by <function id> to be executed. See <subroutine invocation statement> in section 8 for a description of <actual parameters>.

If a <function invocation> is used as a <string primary>, the function must have a <return type> equal to STRING.

TAKE and DROP Functions

The TAKE and DROP functions return a string whose value is some number of characters from the beginning or end of another string expression.

<take drop functions> syntax:



The TAKE function returns a new string whose value is a copy of the first <integer expression> number of characters taken from the <string expression> . If the value of <integer expression> is greater than the number of characters in <string expression> or <integer expression> is less than zero, a run-time error occurs.

The DROP function returns a new string whose value is a copy of the characters remaining in <string expression> after the first <integer expression> number of characters have been discarded. The value of <integer expression> is limited as in the TAKE function.

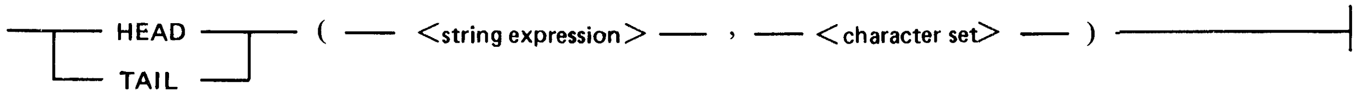
For any <string expression> S and any <integer expression> I in the range 0 LEQ I LEQ LENGTH(S), the following relation is always true:

S = TAKE(S,I) & DROP(S,I)

HEAD and TAIL Functions

The HEAD and TAIL functions return a string whose value is some number of characters from the beginning or end of another string expression. The number of characters to use is determined by scanning the string expression for a <character set>.

<head tail functions> syntax:



The HEAD function returns a new string consisting of a copy of all the leading characters in <string expression> that belong to the set of characters specified by <character set>. If the first character in <string expression> is not a member of the <character set>, a null (zero length) string is returned.

The TAIL function returns a new string consisting of a copy of all the characters in <string expression> that remain after the removal of all the leading characters that belong to <character set>. If all characters in <string expression> are members of the specified <character set>, a null string is returned.

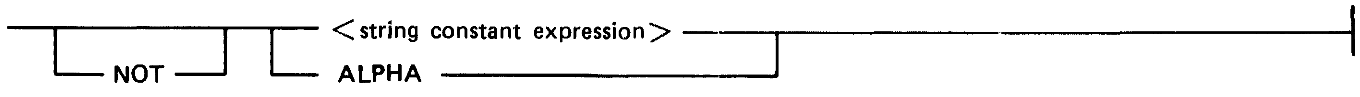
For any <string expression> S and any <character set> C, the following relation is always true:

S = HEAD(S, C) & TAIL(S, C)

Character Set

A <character set> is a collection of characters used to control the action of the HEAD and TAIL functions.

<character set> syntax:

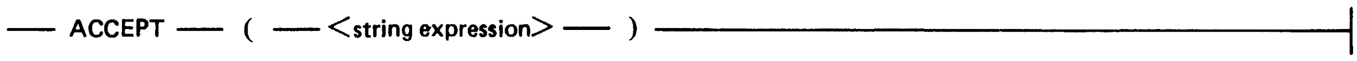


The built-in <character set> ALPHA consists of all the upper-case English alphabet letters A through Z and the digits 0 through 9. The <string constant expression> may have characters in any order. The NOT keyword indicates a character set consisting of all EBCDIC characters except those specified.

ACCEPT Function

The <accept function> returns a string entered by the operator.

<accept function> syntax:



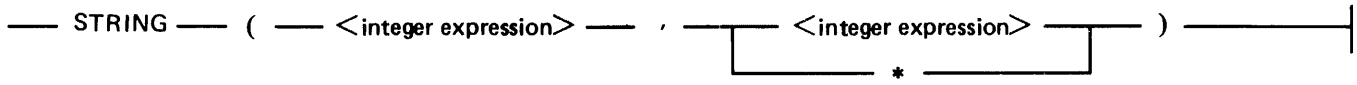
The ACCEPT function displays its <string expression> parameter on the system ODT and waits for the operator to respond with an AX ODT-command. Refer to the *B 1000 Systems System Software Operation Guide, Volume 1,* for more information on the AX ODT-command.

The first 255 characters of the operator response are returned as the value of the ACCEPT function.

STRING Function

The <string function> converts an <integer expression> into a < string expression>.

<string function> syntax:



The <string function> generates a new string whose value is the decimal representation of the absolute value of the first <integer expression>. 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 this second <integer expression> is greater than the minimum number of characters needed to represent the first argument, a sufficient number of leading zero characters are provided. If the value of the second <integer expression> is less than the number of characters needed to represent the first <integer expression> then the rightmost characters are returned. If the second parameter of the STRING function is an asterisk (*) the string will be just long enough to contain all digits of the integer character representation of the first argument with no leading zero characters.

Examples:

String Declaration

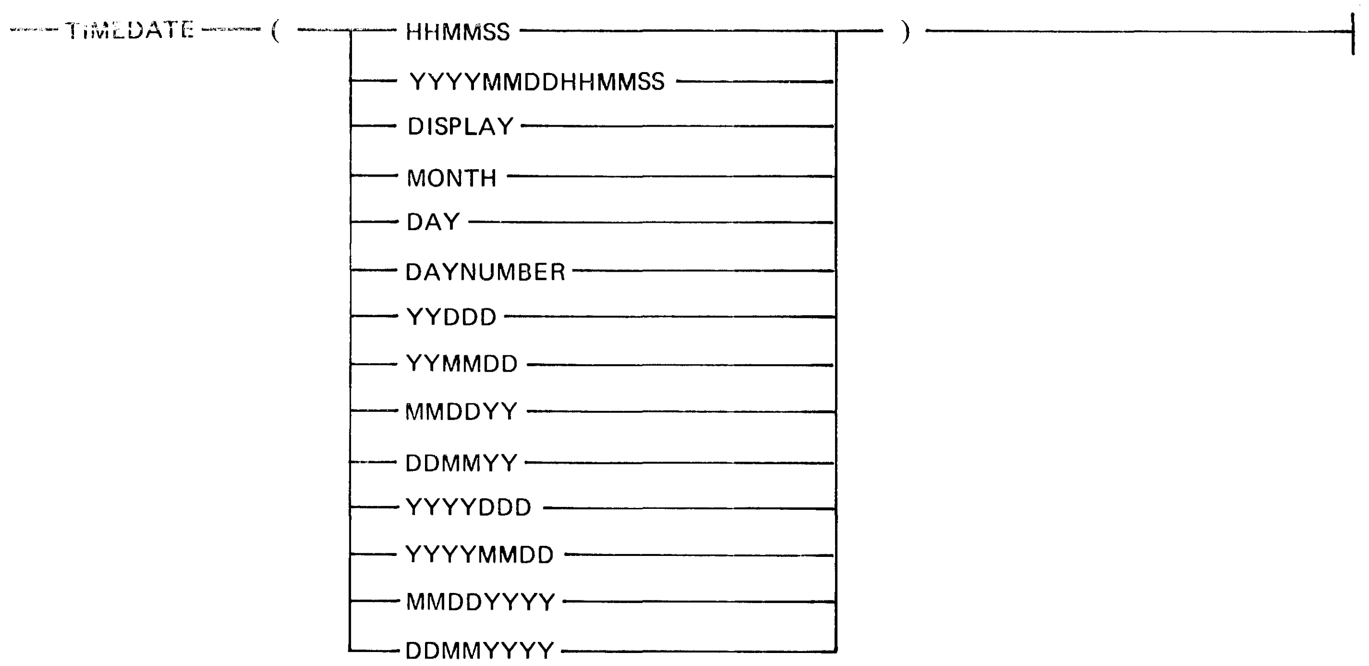
```
STRING STR1, STR2, PGMNAME;
```

String Expressions and Assignment	Result
STR1 := "ABCDEF";	
STR2 := TAKE(STR1, 2);	"AB"
STR2 := DROP(STR1, 2);	"CDEF"
STR1 := " A B C " ;	
STR2 := HEAD(STR1, " ");	" "
STR2 := TAIL(STR1, " ");	"A B C "
STR1 := "FILE/NAME ";	
STR2 := HEAD(STR1,NOT "/ ");	"FILE"
STR2 := TAIL(STR1,NOT "/ ");	"/NAME"
STR1 := STRING(123, *);	"123"
STR2 := STRING(123, 6);	"000123"
STR2 := STRING(1234, 3);	"234"
STR1 := ACCEPT("CONTINUE? ");	The phrase "CONTINUE?" is displayed on the system ODT. The program then waits for operator response by means of the AX ODT-command. This response becomes the value of STR1.
STR1 := "X/";	
STR2 := "Y";	
PGMNAME := STR1 & STR2 ;	"X/Y"

TIMEDATE Function

The TIMEDATE function returns various time-related items as strings. The function has one parameter which is a mnemonic description of the function being requested.

<timedate function> syntax:



All strings returned by the <timedate function> are in upper case. The forms available are described below. Each description includes an example that indicates the result that would be returned at 5:09 PM Friday, March 4, 1983. In all cases, the name of the month and the name of the day are returned in English.

HHMMSS
: Returns the time as a string of six characters. The first two characters are the hours on a 24 hour clock, the next two are the minutes, and the last two are the seconds.

  Example: "170900"

YYYYMMDDHHMMSS
: Returns the time and date as a string of fourteen characters. The first eight are the date as described for YYYYMMDD. The last six characters are the time as described for HHMMSS.

  Example: "19830304170900"

DISPLAY
: Returns the time, day of week, and date in an English-like format. The length of the string varies from 27 to 38 characters.

  Example: "5:09 PM FRIDAY, MARCH 4, 1983"

MONTH

Returns the name of the month as a string. The length of the string varies from 3 to 9 characters.

Example: "MARCH"

DAY

Returns the name of the day of the week as a string. The length of the string varies from 6 to 9 characters.

Example: "FRIDAY"

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.

Example: "5"

YYDDD

Returns the date as a string of five characters. The first two characters are the year modulo 100, and the last three are the day of the year.

Example: "83063"

YYMMDD

Returns the date as a string of six characters. The first two characters are the year modulo 100, the next two are the month, and the last two are the day of the month.

Example: "830304"

MMDDYY

Returns the date as a string of six characters. The first two characters are the month, the next two are the day of the month, and the last two are the year modulo 100.

Example: "030483"

DDMMYY

Returns the date as a string of six characters. The first two characters are the day of the month, the next two are the month, and the last two are the year modulo 100.

Example: "040383"

YYYYDDD

Returns the date as a string of seven characters. The first four characters are the year, and the last three are the day of the year.

Example: "1983063"

YYYYMMDD
Returns the date as a string of eight characters. The first four characters are the year, the next two are the month, and the last two are the day of the month.

Example: "19830304"

MMDDYYYY
Returns the date as a string of eight characters. The first two characters are the month, the next two are the day of the month, and the last four are the year.

Example: "03041983"

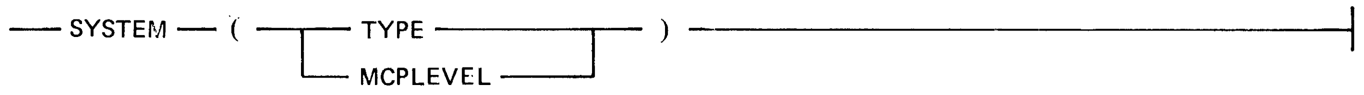
DDMMYYYY
Returns the date as a string of eight characters. The first two characters are the day of the month, the next two are the month, and the last four are the year.

Example: "04031983"

SYSTEM Function

The <system function> is a string function that returns system identification information.

<system function> syntax:



Semantics:

TYPE	Returns the machine type as a string of five characters. Example: "B1900"
MCPLEVEL	Returns the MCP release, level and version as a string of eight characters. Example: "12.0.078"

The format and length of the values returned are as described for B 1000 systems. These descriptions might not be accurate should the WFL <job> containing them be run on other Burroughs systems.

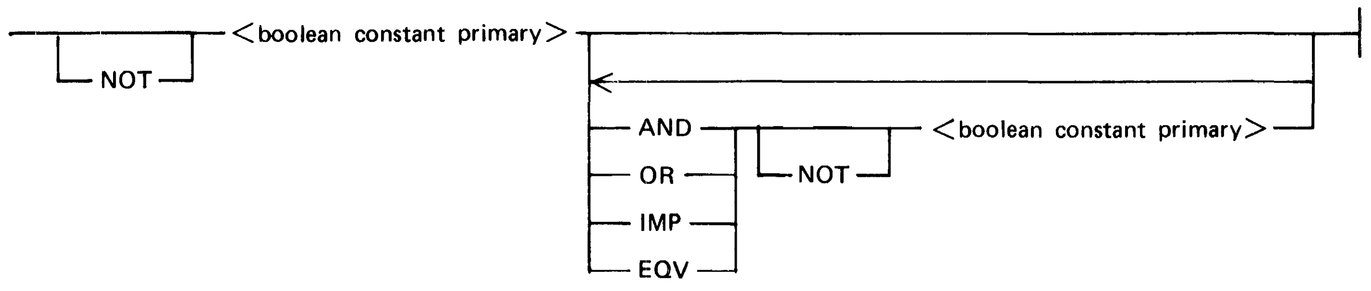
CONSTANT EXPRESSIONS

A constant expression is a combination of basic elements whose value can be determined at compile time. These basic elements can be boolean constants, integer constants, string constants, boolean constant ids, integer constant ids, or string constant ids.

Boolean Constant Expression

A <boolean constant expression> computes logical values at compile time.

<boolean constant expression> syntax:

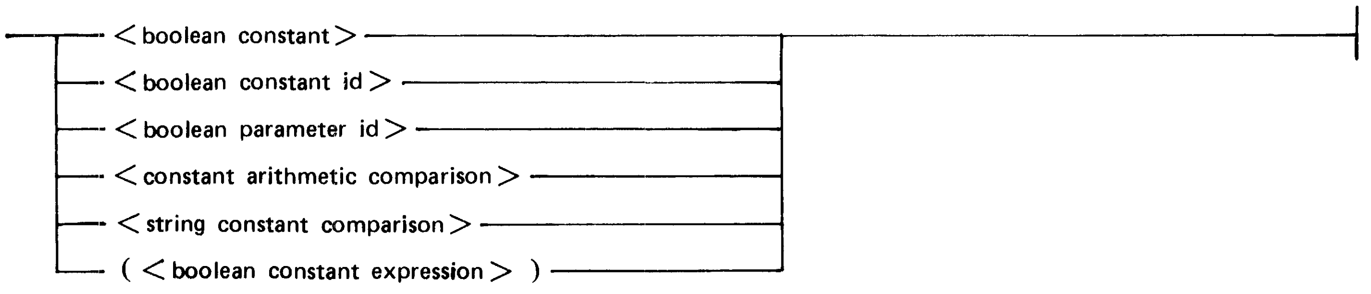


The logical operators have the same meaning as those for boolean expressions.

Boolean Constant Primary

A <boolean constant primary> represents a logical value.

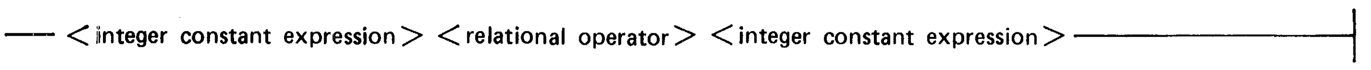
<boolean constant primary> syntax:



Constant Arithmetic Comparison

A <constant arithmetic comparison> compares the values of two <integer constant expressions>.

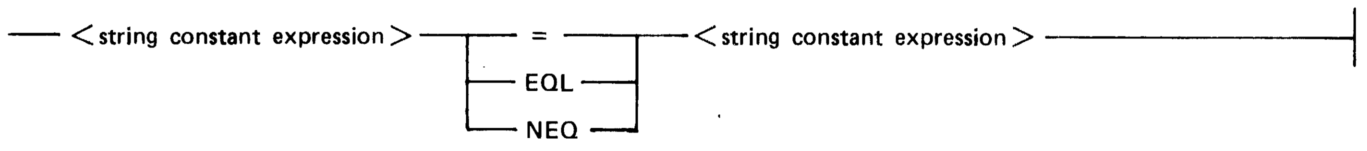
<constant arithmetic comparison> syntax:



String Constant Comparison

A <string constant comparison> compares the value of two <string constant expression>s.

<string constant comparison> syntax:

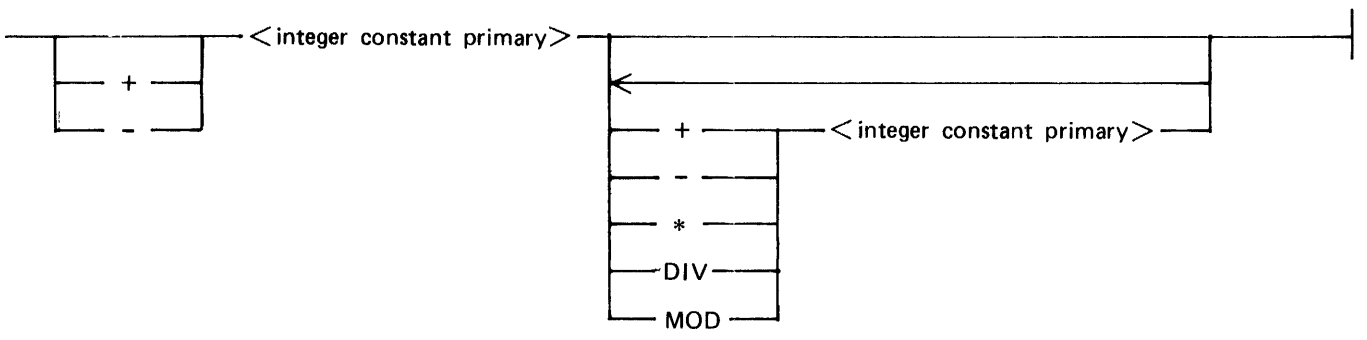


The two string constant expressions are equal only if, once evaluated, all characters in the first string occur in the same order in the second string and the lengths of the two strings are equal.

INTEGER CONSTANT EXPRESSION

An <integer constant expression> computes an integer value at compile time.

<integer constant expression> syntax:

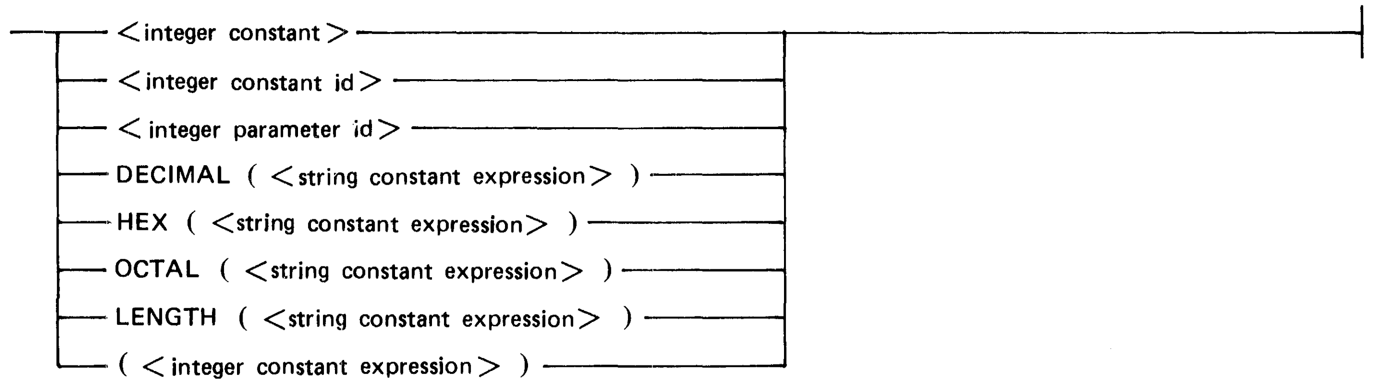


The arithmetic operators have the same meaning as those for integer expressions. Refer to Arithmetic Operators in this section for detailed information.

Integer Constant Primary

An <integer constant primary> represents an integer number.

<integer constant primary> syntax:

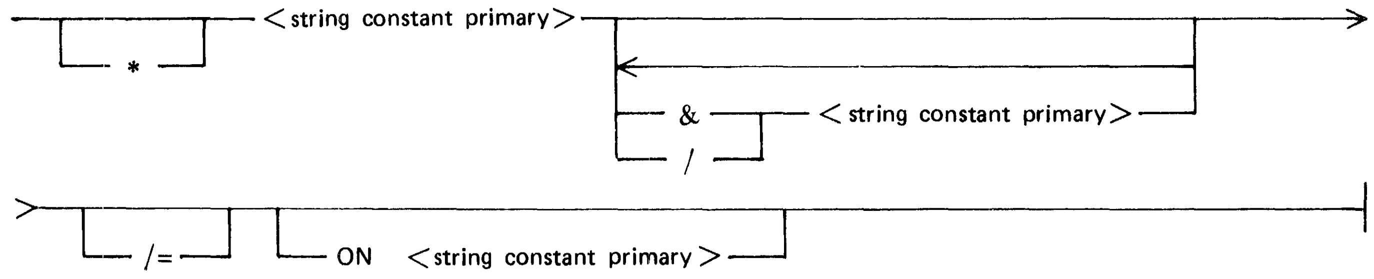

<integer constant>
<integer constant id>
<integer parameter id>
DECIMAL (<string constant expression>)
HEX (<string constant expression>)
OCTAL (<string constant expression>)
LENGTH (<string constant expression>)
(<integer constant expression>)

Refer to Integer Expression in this section for an explanation of the various integer functions.

STRING CONSTANT EXPRESSION

A <string constant expression> computes a string value at compile time.

<string constant expression> syntax:

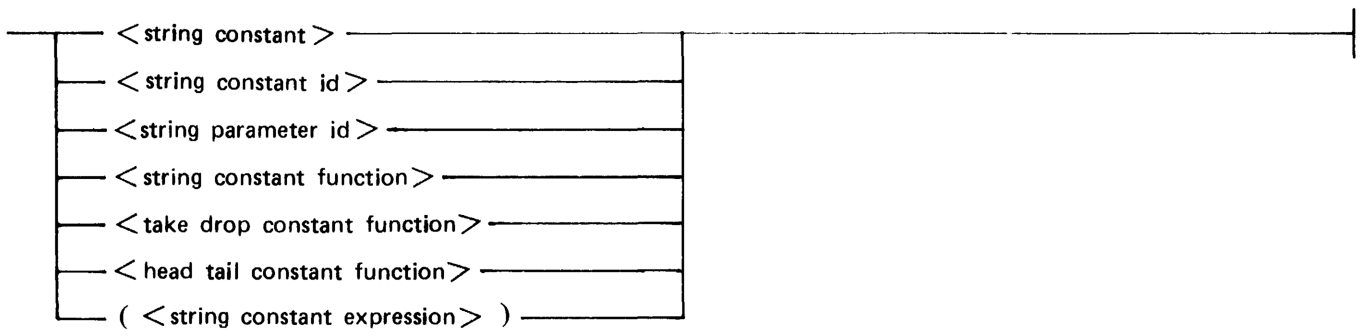


The concatenation operators have the same meaning as those for string expressions. Refer to String Expression in this section for detailed information.

String Constant Primary

A <string constant primary> represents a string.

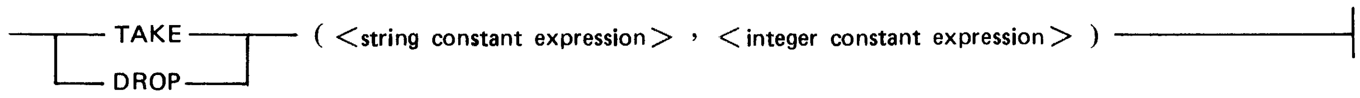
<string constant primary> syntax:



TAKE and DROP Constant Functions

The TAKE and DROP constant functions return a string whose value is some number of characters from the beginning or end of another string constant expression.

<take drop constant function> syntax:

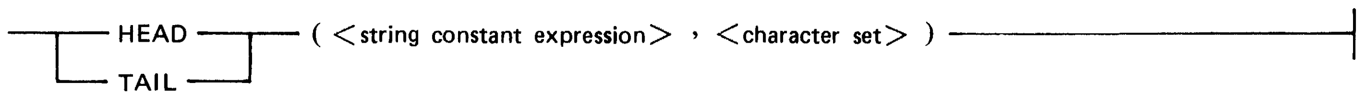


The TAKE and DROP constant functions operate the same as the TAKE and DROP functions. Refer to TAKE and DROP Functions in this section.

HEAD and TAIL Constant Functions

The HEAD and TAIL constant functions return a string whose value is some number of characters from the beginning or end of another string expression. The number of characters to use is determined by scanning the string expression for a <character set>.

<head tail constant function> syntax:

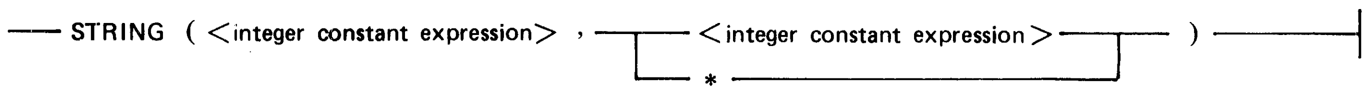


The HEAD and TAIL constant functions operate the same as the HEAD and TAIL functions. Refer to HEAD and TAIL Functions in this section.

STRING Constant Function

The <string constant function> converts an <integer constant expression> into a string.

<string constant function> syntax:



The <string constant function> operates the same as the <string function>. Refer to STRING Function in this section for additional information.

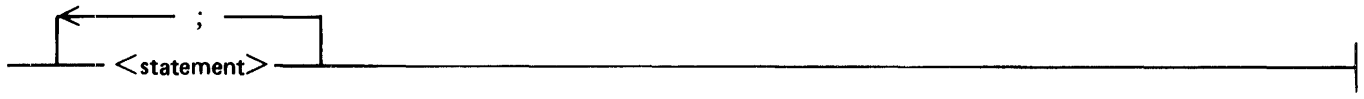
SECTION 8 STATEMENTS

This section describes the statements used in the Work Flow Language. Information which is common to several statements is described at the beginning of this section. The individual statements are described in alphabetical order after the common information.

STATEMENT LIST

<Statement>s are normally executed sequentially in the order in which they are written.

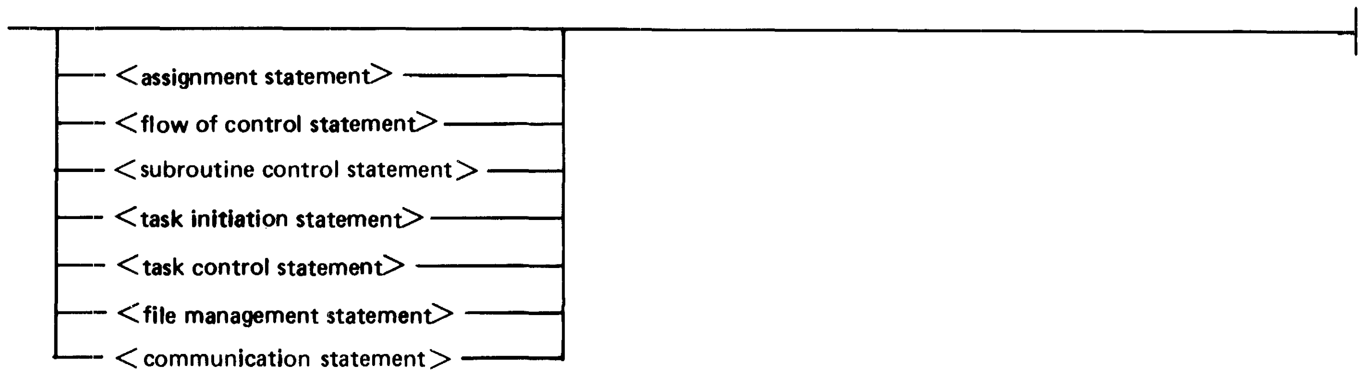
<statement list> syntax:



STATEMENT

A <statement> indicates some type of action to be performed.

<statement> syntax:

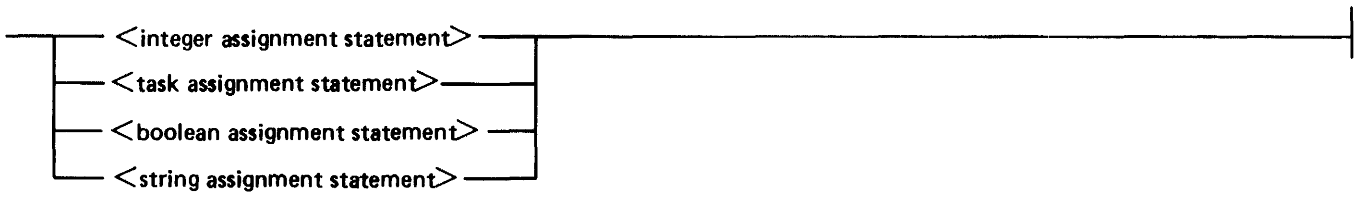


The path through the syntax diagram that does not cross any token is an empty statement.

ASSIGNMENT STATEMENT

The <assignment statement> is used to replace the current value of a variable (or attribute) by a new value.

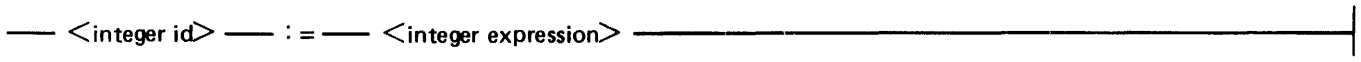
<assignment statement> syntax:



Integer Assignment Statement

An <integer assignment statement> is used to replace the current value of an integer variable by a new value.

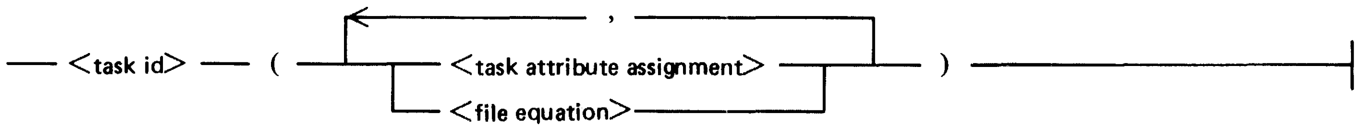
<integer assignment statement> syntax:



Task Assignment Statement

A <task assignment statement> is used to replace the current value of an attribute by a new value.

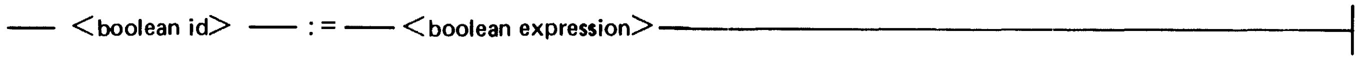
<task assignment statement> syntax:



Boolean Assignment Statement

A <boolean assignment statement> is used to replace the current value of a boolean variable by a new value.

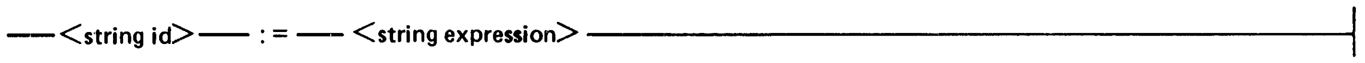
<boolean assignment statement> syntax:



String Assignment Statement

A <string assignment statement> is used to replace the value of a string variable by a new value.

<string assignment statement>



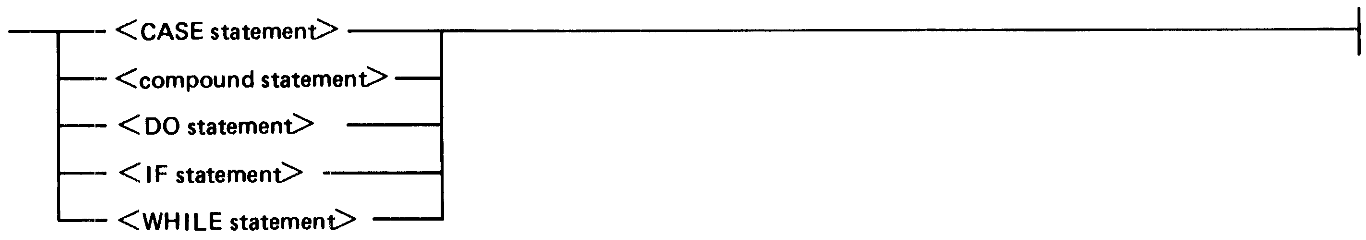
Examples:

```
INTEGER X, Y;
TASK A, B;
BOOLEAN C;
STRING S;

X := 128; Y := 256;
A (PRIORITY=7); B (PROTECTED);
C := TRUE;
S := "JOBSUMMARY=" & MYSELF (JOBSUMMARY);
```

FLOW OF CONTROL STATEMENT

The sequential flow of control can be altered by a < statement> which indicates that its successor is to be a <statement> other than the one which follows it in the program.

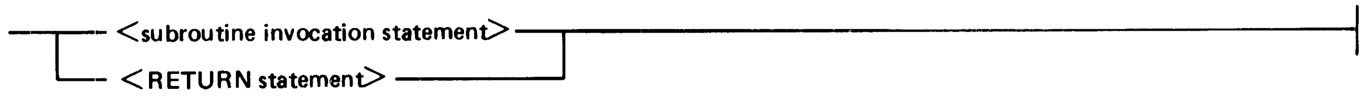
<flow of control statement> syntax:



SUBROUTINE CONTROL STATEMENT

The subroutine control statements consist of an invocation and a return statement.

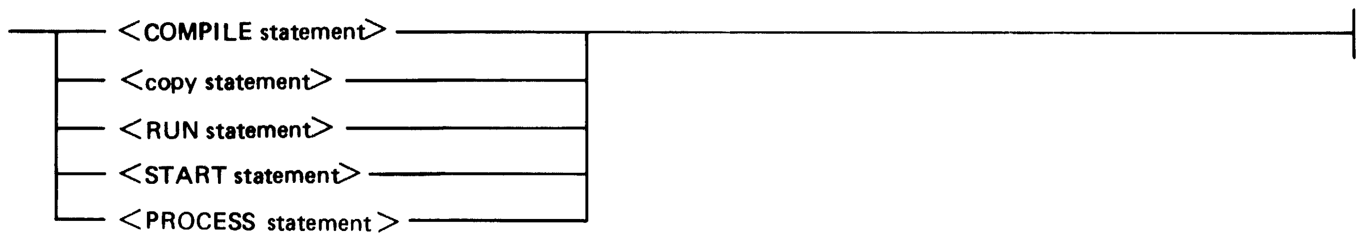
<subroutine control statement> syntax:



TASK INITIATION STATEMENT

A <task initiation statement> starts application programs and system functions as separate dependent tasks.

<task initiation statement> syntax:



The COMPILE, copy, RUN, and START statements run synchronously, that is, the execution of the job task (WFL program) is suspended until the dependent task is completed. For example,

```
RUN X;
RUN Y;
```

Program X runs first. When it is finished, program Y runs.

The <PROCESS statement> initiates asynchronous tasks. For more information refer to PROCESS Statement in this section.

The <START statement> initiates a synchronous dependent task and then an independent job task. For more information refer to START Statement in this section.

The job task becomes the parent task of the dependent task started with a task initiation statement. A job task does not have a parent. The independent task initiated by a <START statement> does not have a parent task. (A program which is called by a task initiate, IPC call, program call, or sort has a parent which is the calling task. A program initiated by the EXECUTE ODT-command, including a ZIP, does not have a parent.)

A task variable can be attached to a task by placing the < task id>, enclosed in brackets, after the title of the task. The task variable can then be used to set attributes before the task is initiated or to inquire about the value of attributes after the task completes.

In the following examples, T and T2 have been declared as TASKs.

Examples:

```
COMPILE ARO [T] WITH COBOL [T2];
  % T is attached to the object program ARO
  % T2 is attached to COBOL.

RUN X [T];
  % T is attached to the program X.

COPY A TO B [T];
  % T is attached to the SYSTEM/COPY program.

START A/JOB [T];
  % T is attached to the SYSTEM/WFL program.

PROCESS RUN Y [T];
  % T is attached to the program Y.
  % Y runs asynchronously with the job.
```

Task Attributes and File Equation

Tasks have properties, called task attributes, that contain the information needed to find, run, and keep track of the task. Misusing an attribute (for example, setting USERCODE to an invalid <user specification>) results in a fatal run-time error. For specifics on task attributes refer to section 5, Task Attributes.

A <file equation> is used to specify changes to the various files declared within the program being initiated. For specifics on file attributes refer to section 6, File Attributes.

Task attributes and <file equation>s can be specified in several ways:

1. System or user defaults.
2. Inherited from parent task.
3. Language source (such as WFL <job attribute specification>).
4. <Compile task equation list> without COMPILER.
5. MODIFY ODT-command.
6. <Task equation list> as part of a task initiation statement.
7. ODT-commands (such as PR or DY), <task assignment statement> after task initiated (such as MYSELF (PRIORITY=9)).

Attributes are assigned values in the order listed above starting with 1 and proceeding to 7. If the values conflict, the value specified in the highest number assignment is used.

Example:

```
COMPILE X COBOL LIBRARY;
   COMPILER PRIORITY = 5;  %priority of compilation
   PRIORITY =6;  % sets priority in code file X

RUN X;    % runs at PRIORITY 6

RUN X;
   PRIORITY = 7;  % overrides compiled-in PRIORITY
```

Reuse of Task Variable

WFL saves all the information associated with a task variable until the job is completed or until the task is initialized. Special care must be taken when reusing a task variable for a new task.

In the following example, there is one task declared, T, which is used for two <RUN statement>s. In the second <RUN statement>, the PRIORITY information, file-equation information, and the setting of CHARGE are all still in effect for the execution of Y.

Example 1:

```
BEGIN JOB EXAMPLE1;
   TASK T;
   T(PRIORITY=5);
   RUN X[T];
      FILE F(KIND=DISK);
   T(CHARGE=123);
   RUN Y[T];
END JOB
```

There may also be less-obvious side effects of reusing T in the manner shown in example 1. For example, the operator may have changed the PRIORITY of program X or X may have changed its MAXWAIT. The changed PRIORITY and MAXWAIT would still be in effect when program Y is executed, and could have an undesired effect on Y.

Retaining the information about the previous task is normally not desired when reusing a task variable.

In the next example, T is initialized. After that point, the PRIORITY information and the previous file-equation information are no longer in effect.

Example 2:

```
BEGIN JOB EXAMPLE2;
   TASK T;
   T(PRIORITY=5);
   RUN X[T];
      FILE F(KIND=DISK);
   INITIALIZE (T);
   RUN Y[T];
END JOB
```

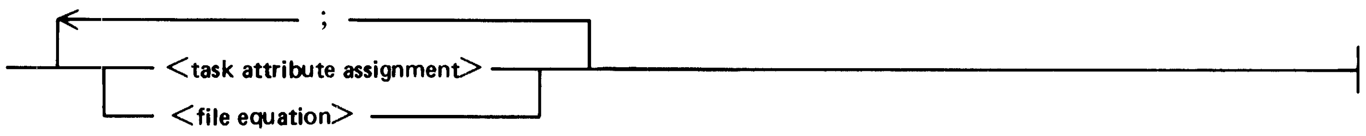
It is not possible for WFL to automatically clear the information associated with a task variable. If the information were cleared just before the task was initiated, information such as the PRIORITY in the above example would be lost. If the information were cleared just after the task is completed, important result information such as TASK STATE would be lost.

The safest way to reuse a task variable is to always INITIALIZE it before associating the variable with another task, as is shown in example 2.

Task Equation List

All task initiation statements allow an optional <task equation list>. A <task equation list> is used to specify task and file attribute values at task initiation time.

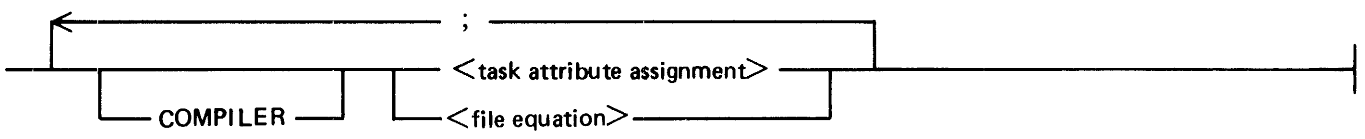
<task equation list> syntax:



Compile Task Equation List

A <compile task equation list> is used to specify task and file attribute values for either the compiler or the object program when a compile is initiated.

<compile task equation list> syntax:



<Task attribute assignment> and < file equation> specifications preceded by the word COMPILER and those not preceded by the word COMPILER may be intermixed.

Task attributes preceded by the word COMPILER are assigned to the compiler.

Example:

```
COMPILE ARO WITH COBOL LIBRARY;
  COMPILER FILE CARDS (TITLE=AR, DISK);•
%   COBOL source is in file AR
  COMPILER MAXPROCTIME = 100;
%   the compiler will have
%   a maximum processor time of 100 seconds
  COMPILER MAXLINES = 130
%   the compiler will have
%   a maximum of 130 printed lines
```

Attributes not preceded by the word COMPILER override any attributes specified in the programming language file and in a task variable if it is attached to the object code file. The attributes are assigned to the task whenever the compiled program is executed, unless the attribute values are overriden by run-time task attribute values (refer to the <RUN statement>).

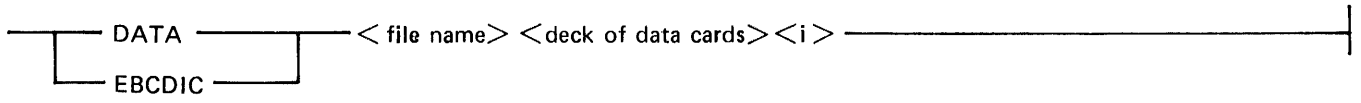
Example:

```
COMPILE ARO WITH COBOL LIBRARY;
  COMPILER FILE CARDS (TITLE=AR, DISK);
  PRIORITY = 5;
%   the compiled program will run at priority 5
  FILE F (TITLE = Y/Z)
%   file F of the compiled program
%   will have a title of Y/Z
```

DATA SPECIFICATION

A <data specification> is used in a <COMPILE statement> or a <RUN statement> to give input data to a running object program.

<data specification> syntax:



The <file name> must appear on the same source record as DATA or EBCDIC. The remainder of that record must be blank. <Data specification> is not valid in an ODT-command.

A deck of data cards can contain a maximum of 94500 records.

A <data specification> specifies the external recording mode (EXTMODE) and external file name (FILENAME) file attributes of the file. An EXTMODE declared by DATA or EBCDIC specifies that the data deck is recorded in EBCDIC.

When a task tries to open a card file, it is assigned the first unread deck with the correct name located between its task invocation and the next task invocation. A data deck is read only once during an invocation of the task it follows.

The statement following a data deck must start with an invalid character <i> in column 1. The < i> indicates the physical end of the data card deck and any semicolon that follows the data specification. In the following example (JOB J), a semicolon between the <i> and the IF is not required, and a semicolon between the < i> and the ELSE is not allowed.

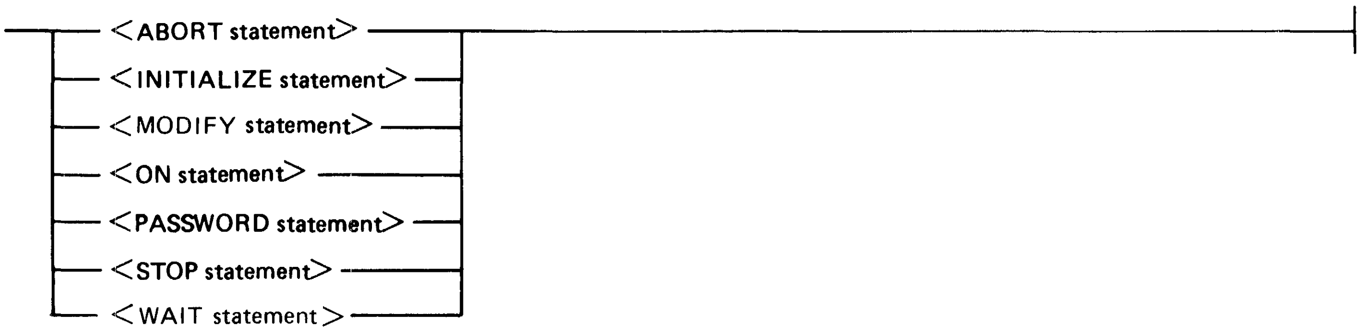
Example:

```
         BEGIN JOB J (BOOLEAN B);
           RUN P1;
           DATA D1
              ...
     <i> IF B THEN
           RUN P2;
           DATA D2
              ...
     <i> ELSE
           RUN P3;

         RUN TEST/PROGRAM;
         DATA INPUT/FILE
              .
              .
              .
     <i> EBCDIC DECK1
              .
              .
              .
     <i> END JOB
```

TASK CONTROL STATEMENT

A <task control statement> is used to control a task.

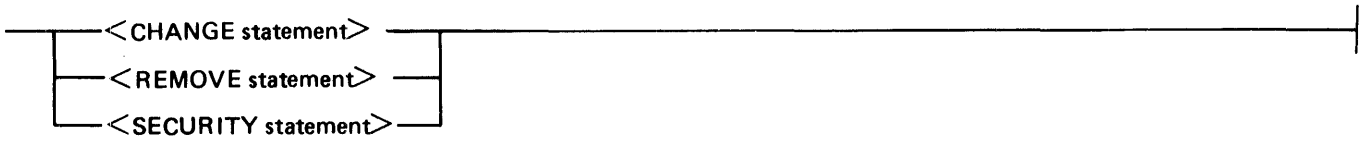
<task control statement> syntax:



FILE MANAGEMENT STATEMENT

File management statements are used to change the name, attributes, or residence of disk files.

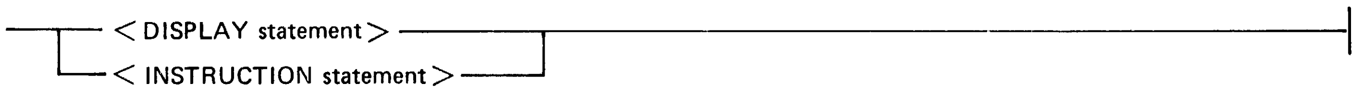
<file management statement> syntax:



COMMUNICATION STATEMENT

A communication statement is used to communicate with the person running a <job>.

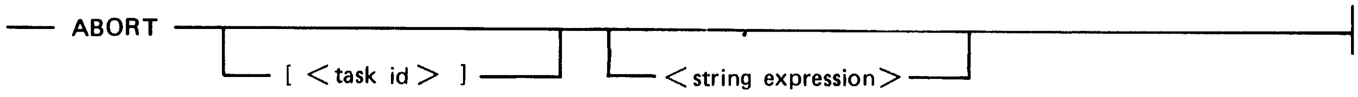
<communication statement> syntax:



ABORT STATEMENT

The <ABORT statement> discontinues tasks.

<ABORT statement> syntax:



If a <task id> is specified for an <ABORT statement>, the task associated with that <task id> is terminated. If a <task id> is not specified, the job and any tasks initiated by the job are terminated. The <string expression> is displayed prior to the abort.

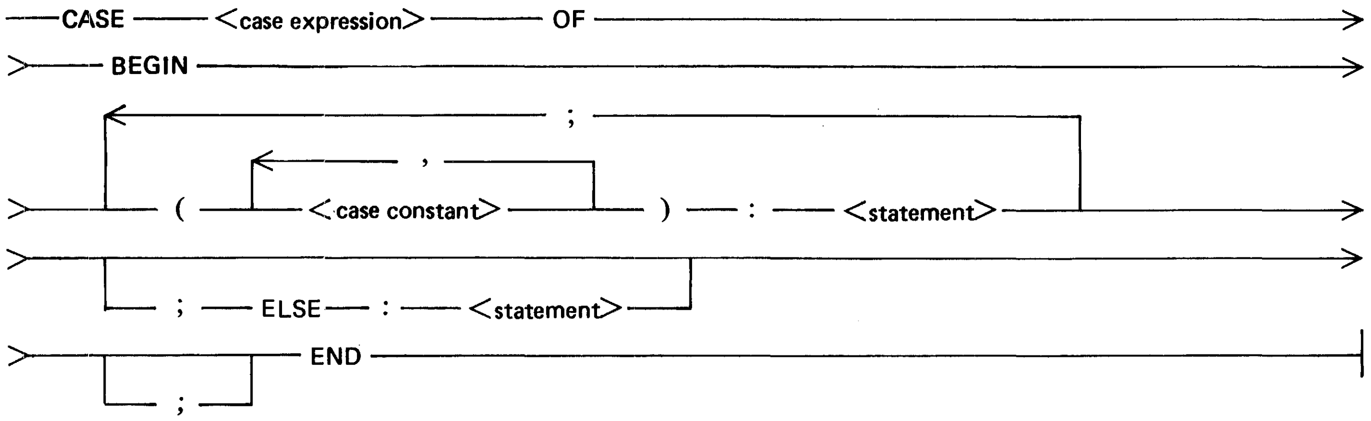
Examples:

```
IF T ISNT COMPILEDOK THEN ABORT;

ABORT "THIS JOB HAS BEEN ABORTED";

IF T2(TASKVALUE) = 3 THEN ABORT [T1] "SUB TASK ABORTED";
```

CASE STATEMENT

The <CASE statement> dynamically selects one of several alternative <statement>s for execution.

<CASE statement> syntax:



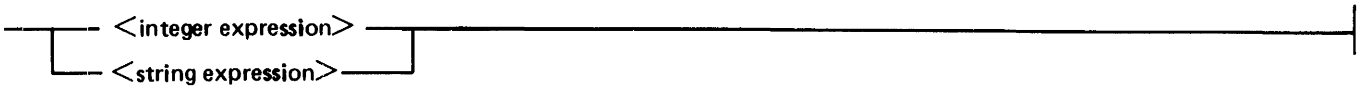
The <CASE statement> executes only the <statement> associated with the < case constant> that is equal to the value of the <case expression>.

If no matching <case constant> is found and an ELSE specification is given for the < CASE statement>, the statement associated with the ELSE is executed. If no matching <case constant> is found and no ELSE specification appears for the <CASE statement>, a run-time error occurs.

Case Expression

A <case expression> is used to select which <statement> is executed.

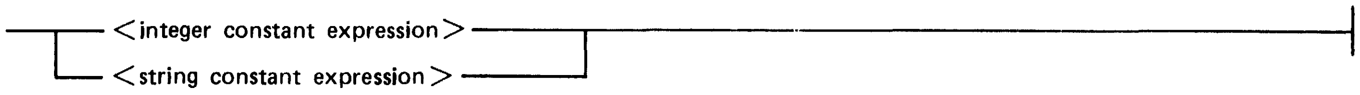
<case expression> syntax:



Case Constant

A <case constant> is used to label the <statement>s.

<case constant> syntax:



All <case constant>s must be of the same type as the <case expression>. In addition, no two <case constants>s within the same <CASE statement> may have the same value.

A maximum of 256 <case constant>s are allowed.

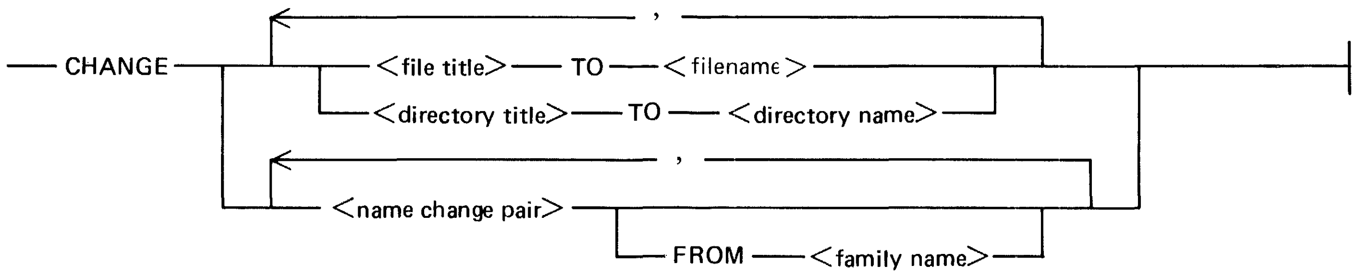
Example:

```
CASE I OF BEGIN
  (0):
    ;     % NO ACTION TO BE TAKEN
  (1):
    RUN X;
  ELSE:
    ABORT "INVALID VALUE FOR I"
END
```

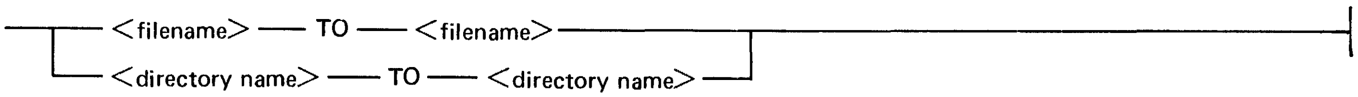
CHANGE STATEMENT

The <CHANGE statement> changes the name of files on disk.

<CHANGE statement> syntax:


CHANGE
,
<file title> TO <filename>
<directory title> TO <directory name>
,
<name change pair>
FROM <family name>

<name change pair> syntax:



If a directory is specified, the names of all files in that directory are changed. The new directory must not exist prior to the change.

The FROM clause indicates that the list of files preceding it (but not any files prior to a preceding FROM clause) reside on the disk specified by <family name>.

If a <family name> is not specified and the file is present on the default family, the file on the default family will be changed. If the file is not present on the default family but is present on DISK, the file on DISK will be changed. If the file is present on both the default family and DISK, only the file on the default family will be changed. A directory is treated as a whole in the same manner.

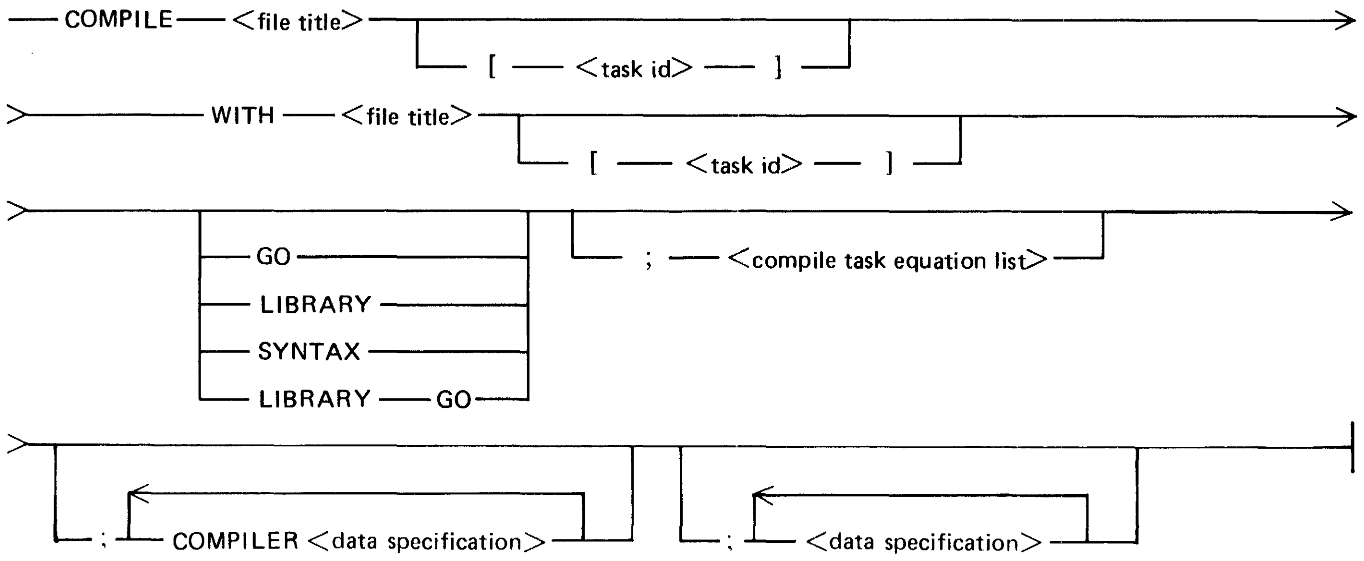
Examples:

```
STRING S1, S2, S3;

CHANGE X TO Y;
%    Change the name of file X to Y.

CHANGE A/B ON USERS TO C/D;
%    Change the name of file A/B on the disk USERS to C/D.

CHANGE X TO Y, X/X TO Y/Y FROM MYPACK,
XX TO YY FROM USERS, Z TO ZZ;
%    Change the name of file X on disk MYPACK to Y.
%    Change the name of file X/X on the disk MYPACK to Y/Y.
%    Change the name of file XX on the disk USERS to YY.
%    Change the name of file Z on the default family to ZZ.

CHANGE X/= TO Y/=;
%    Change all the files in the X directory to be in the Y
%    directory.  The  Y  directory must not exist prior to
%    the change.

S1 := "A/=";
S2 := "INVENTORY";
S3 := "B/=";
CHANGE #S1 ON #S2 TO #S3;
%    Change  all the files in the A directory on  the  disk
%    INVENTORY to be in the B directory.
```

COMPILE STATEMENT

The <COMPILE statement> is used to initiate a compiler to compile an object program.

<COMPILE statement> syntax:



The first <file title> is the name of the resulting code file. The second <file title> is the name of the compiler to be used. If a disposition for the code file is not specified, GO is assumed. The semantics of the dispositions are:

GO
: Execute the code file if it compiles without errors. The object code file is not entered into the disk directory and must be compiled again to be used again.

LIBRARY
: Enter the code file in the directory if it compiles without errors.

SYNTAX
: Compile for syntax only.

LIBRARY GO
: If no syntax errors are present, the code file is entered in the directory and executed.

The <COMPILE statement> initiates the compiler task (and object task if a disposition of GO was specified) as synchronous dependent tasks. The job waits for the compiler task (and the object task if it is initiated) before continuing to the next statement.

If a <task id> is specified following the code <file title> and the GO disposition has been specified, the task variable is attached to the execution of the object code file. The task variable specified following the compiler name is attached to the compiler.

If a <task id> is specified for the object code file and the LIBRARY disposition has been specified, the non-default values in the task variable are permanently attached to the object code file.

The word COMPILER must precede the DATA or EBCIDIC specification on decks to be read by a compiler. This prevents a compiler from reading data decks intended for the 'go' part and prevents the 'go' from reading decks intended for a compiler. All <data specifications>s preceded by the word COMPILER must appear prior to those not preceded by the word COMPILER, and all attribute specifications must precede any data decks.

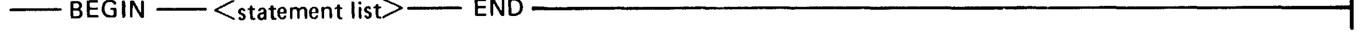
Example:

```
TASK T1, T2;

COMPILE X/Y WITH COBOL LIBRARY GO;
  COMPILER PRIORITY = 5; % COBOL runs at priority 5
  FILE F(TITLE=Y/Z); % file F in X/Y has a title of Y/Z

COMPILE A WITH RPG SYNTAX;
COMPILE SOX WITH FORTRAN;

T1 (PRIORITY = 3);
T2 (PRIORITY = 5);
COMPILE X/Y [T1] WITH COBOL [T2] LIBRARY GO; PRIORITY = 7;
  % The priority attribute of T1 is overridden
  % by the object compile task equation list,
  % (that is, PRIORITY = 7).
  %
  % COBOL runs at a priority of 5.
  %
  % When the GO part of the LIBRARY GO runs,
  % T1 is attached to the task running X/Y.
  %
  % T2 is attached to the compile task (COBOL).

RUN X/Y;
  % X/Y will run at a priority of 7.

T1 (PRIORITY = 6);
COMPILE X/Y WITH COBOL [T1] LIBRARY;
  % COBOL will run at a priority of 6.


COMPILE X WITH COBOL74 GO;
COMPILER MAXLINES = 1000;
MAXLINES = 2000;
COMPILER DATA CARD
       ....
       ....
<i>DATA DECK1
       ....
       ....
<i>
    ....
    ....
```

COMPOUND STATEMENT

A <compound statement> specifies that the statements in the <statement list> are to be executed as a group.

<compound statement> syntax:



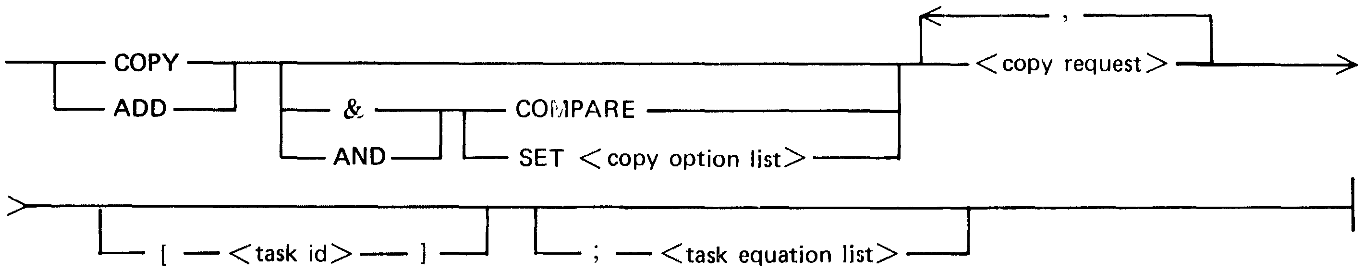
Example:

```
BEGIN
  RUN X;
  I :=1;
END
```

COPY STATEMENT

The <copy statement> is used to copy disk and tape files.

<copy statement> syntax:



If the verb ADD is used instead of COPY, the files are copied to each outout volume where they are not already resident. ADD cannot be requested if an output volume is a tape or if a HOSTNAME is specified for an input or output volume.

The AND COMPARE phrase sets the COMPARE option to TRUE. For more information about the COMPARE option, refer to Copy Options List in this section.

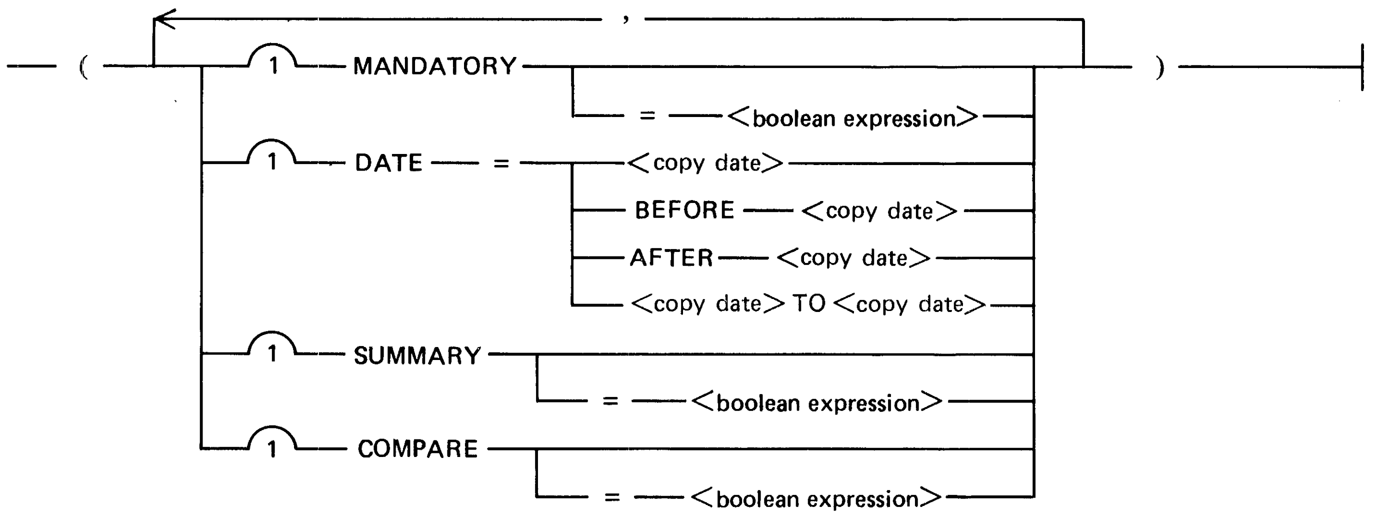
A <copy statement> in a job which is not running under a privileged usercode can copy only files that may be accessed by the usercode of the job.

The <copy statement> initiates a RUN of the program in the CPY entry of the Name Table, normally the SYSTEM/COPY program, as a synchronous dependent task. For more information about the SYSTEM/COPY program, refer to the *B 1000 Systems System Software Operation Guide, Volume 2.*

Copy Options List

A <copy options list> is used to specify the options that apply to the entire <copy statement>.

<copy options list> syntax:



All options are assumed to be FALSE unless explicitly set in the options list. If a boolean option appears in the option list, it is assigned the value of the <boolean expression> . If the <boolean expression> is omitted, the option is assigned the value TRUE.

The MANDATORY option in a COPY or ADD statement means the SYSTEM/COPY program will either successfully complete the entire statement or it will be abnormally terminated. For errors that an operator can correct, such as missing files or packs, the SYSTEM/COPY program will request operator assistance. The operator will have only two choices: correct the situation so that the request can continue or abnormally terminate (DS or DP) the program. For errors that the operator cannot correct, such as hard I/O errors or comparison errors, SYSTEM/COPY will abnormally terminate without operator intervention.

The DATE copy option refers to the date of the last file update (as nearly as the host system can determine it). Only those files whose update date is within the specified range are copied. The BEFORE and AFTER clauses are exclusive of the specified date and the date range is inclusive of the delimiting dates. The first date of a date range must be less than or equal to the second date.

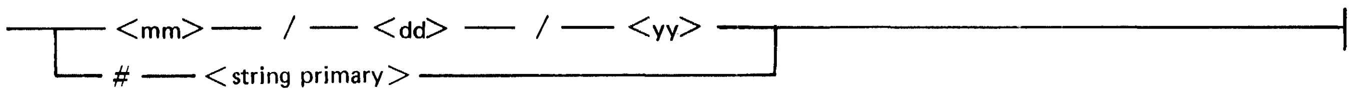
The SUMMARY copy option causes the SYSTEM/COPY program to produce a printer listing of the results.

The COMPARE copy option causes each file to be compared immediately after it is copied. The COMPARE option must not be set if either the input or output volume specifies a HOSTNAME.

Copy Date

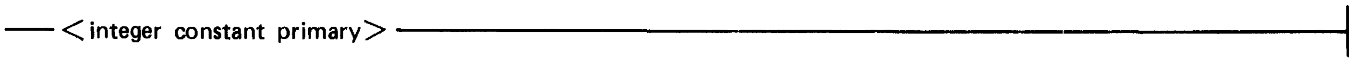
A <copy date> is used to specify the month, day, and year an event occurred.

<copy date> syntax:



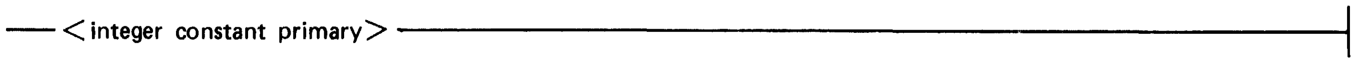
A <copy date> must specify a valid date. If the #<string primary> form is used, the <string primary> is evaluated at run-time and must contain a string in the form <mm> /<dd>/<yy>.

<mm> syntax:



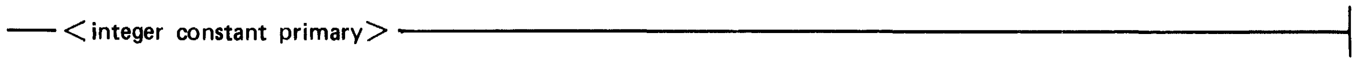
The <integer constant> must be only one or two digits and must be in the range 1 through 12. It represents the month of the year.

<dd> syntax:



The <integer constant> must be only one or two digits and must be in the range 1 through 31. It represents the day of the month.

<yy> syntax:

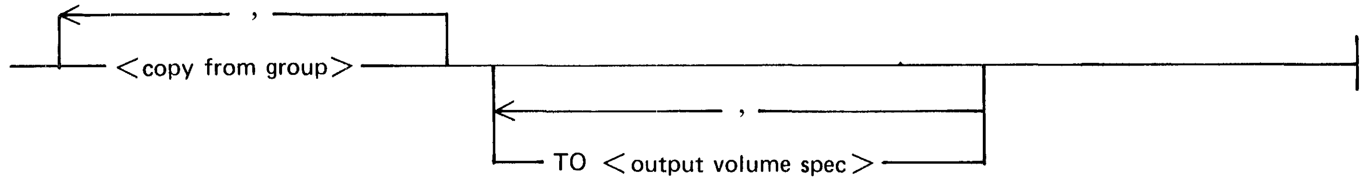


The <integer constant>, a MOD 100 representation of the year, must be exactly two digits.

Copy Request

A <copy request> identifies the files (including the volume on which they reside) that are to be copied and the volume or volumes to which they are to be copied.

<copy request> syntax:



Either a FROM phrase or a TO phrase must appear in each < copy request>, unless all <file spec>s contain an AS phrase.

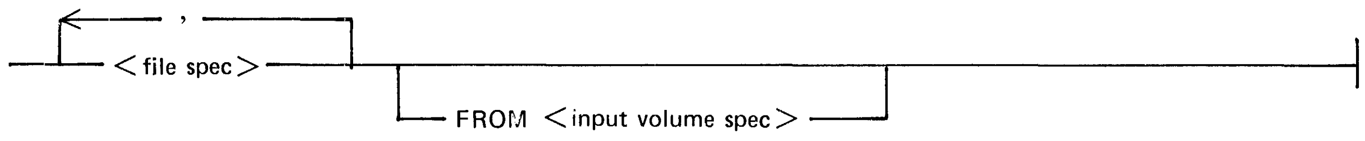
Each TO phrase in a list of TO phrases (one or more adjacent TO phrases) applies to the <copy from group>s (list of <file spec>s and FROM phrases) preceding it in the same <copy request> (but not the <copy from group>s separated by another list of TO phrases). If a <copy request> does not contain a TO phrase, the output volume is the default family (KIND=DISK).

When more than one destination volume is associated with a list of files, the TO clauses must be adjacent. A <copy request> can have a maximum of eight TO phrases. Files associated with more than one destination volume are copied, at the same time and in the same order, to all the associated destination volumes.

Copy From Group

A <copy from group> identifies the files and the volume on which they reside that are to be copied to a given volume or volumes.

<copy from group> syntax:

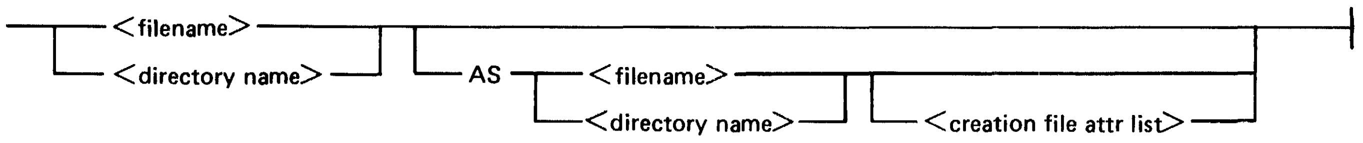


Each FROM phrase applies to the list of <file spec>s preceding it in the same <copy from group>, but not those <file spec> s separated by a TO phrase or another FROM phrase. If a < copy from group> does not contain a FROM phrase, (a FROM phrase does not occur between the list of <file spec>s and the following TO phrase or the end of the statement), the input volume is the default family, that is, the user's default pack, (KIND=DISK). A copy statement can have a maximum of 32 FROM phrases.

File Spec

A <file spec> is used to specify the name of a file or directory to be copied and, optionally, a different name for the new file or directory.

<file spec> syntax:

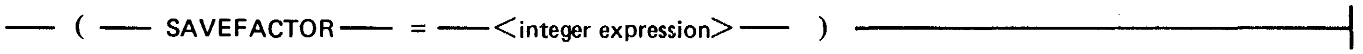


If a file specification is of the form name AS name, then either both names must be <filename>s, or both names must be <directory name>s.

Creation File Attr List

A <creation file attr list> is used to specify file attributes for the new file.

<creation file attr list> syntax:

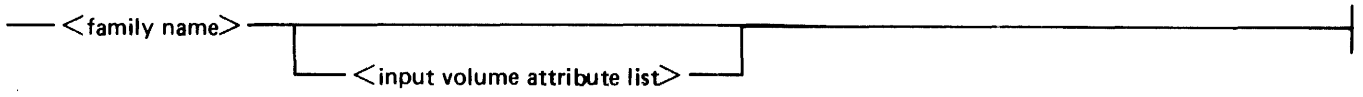


SAVEFACTOR may only be specified if the output volume is a disk.

Input Volume Spec

An <input volume spec> is used to specify the disk or tape from which files are copied.

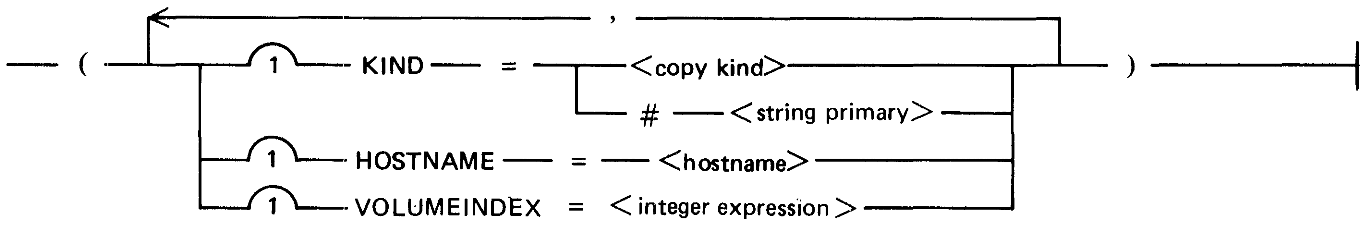
<input volume spec> syntax:



Input Volume Attribute List

An <input volume attribute list> specifies the file attributes used to select the disk or tape from which the files are copied.

<input volume attribute list> syntax:



The <string primary> in the KIND assignment must evaluate to a <copy kind>.

HOSTNAME may only be specified if the input volume is a disk and the COMPARE copy option is FALSE. HOSTNAME must not be specified with the ADD verb. If HOSTNAME is specified, all <file spec>s for that volume must contain only <filename>s, that is, they must not contain <directory name>s.

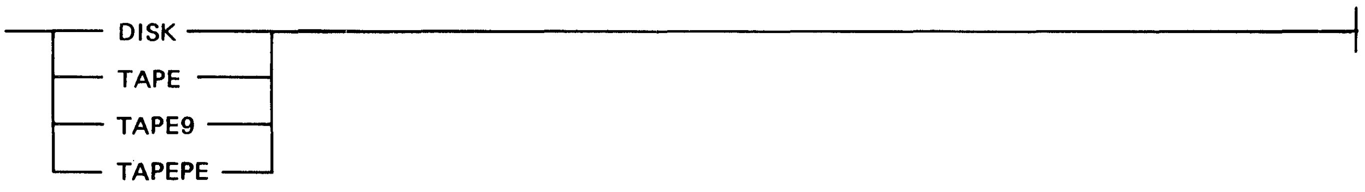
VOLUMEINDEX may only be specified if the input volume is a tape.

For more information about KIND, HOSTNAME, or VOLUMEINDEX, refer to appendix B.

Copy Kind

A <copy kind> is used to specify whether the input or output volume is a disk or tape.

<copy kind> syntax:



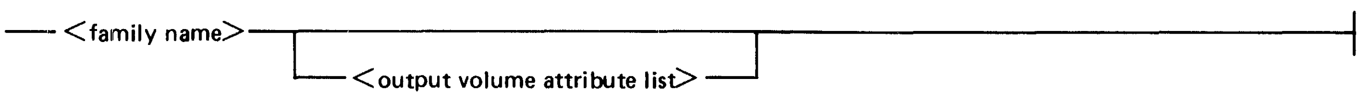
The default KIND is DISK.

For further information, refer to the KIND file attribute in appendix B.

Output Volume Spec

An <output volume spec> is used to specify the disk or tape to which the new files are copied.

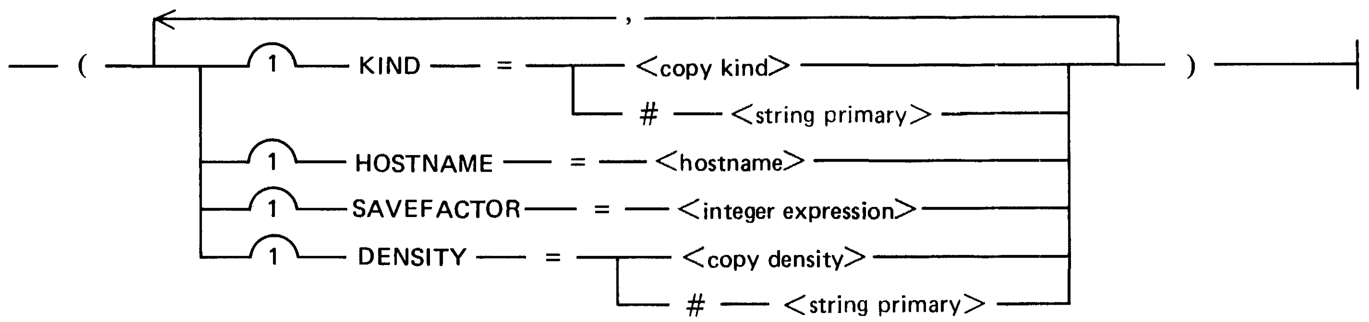
<output volume spec> syntax:



Output Volume Attribute List

An <output volume attribute list> is used to specify the file attributes used to create the disk or tape to which the new files are copied.

<output volume attribute list> syntax:



HOSTNAME may only be specified if the output volume is a disk and the COMPARE copy option is FALSE. HOSTNAME must not be specified with the ADD verb.

SAVEFACTOR may only be specified if the output volume is a tape.

DENSITY may only be specified if the output volume is a tape. If both DENSITY and a specific KIND are specified, they must be compatible, that is, BPI800 may be specified for only TAPE or TAPE9 and BPI1600 may be specified for only TAPE or TAPEPE.

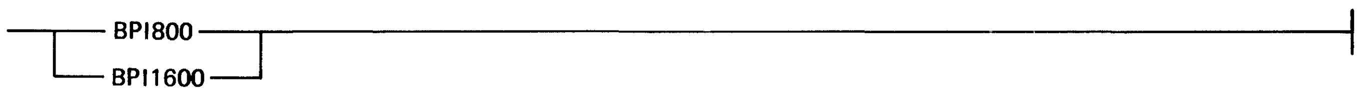
The KIND and DENSITY <string primary>s must evaluate to a <copy kind> or < copy density>, respectively.

For more information about KIND, HOSTNAME, or SAVEFACTOR refer to appendix B.

Copy Density

A <copy density> is used to specify the recording density for an output tape volume.

<copy density> syntax:



Copy Order

All files are copied in the same order as they are specified in the <copy statement> except that (1) all copies from tape are arranged to correspond to the order in which files appear in the tape directory, and (2) all copies from the same input tape volume that are specified in adjacent FROM clauses are performed as an indivisible group, that is, the tape volume is copied in one pass.

Examples:

```
    COPY A, B FROM T1 (KIND=TAPE), C, E FROM T2 (KIND=TAPE)
    TO P1 (KIND=DISK), D, F FROM T2 (KIND=TAPE) TO P2 (KIND=DISK);
```

Note that two of the FROM CLAUSES specify the same input tape volume. Files A and B are copied from the tape T1. Files C and E are copied to the disk P1. Files D and F are copied to the disk P2.

```
    COPY I, J TO P1 (KIND=DISK), K, L FROM T1 (KIND=TAPE) TO P1
    (KIND=DISK), TO T2 (KIND=TAPE);
```

Files I and J are copied from the default disk family. The FROM T1 does not apply to the file I and J because of the intervening TO P1 clause. Files K and L are copied from the tape T1. Files I and J are copied to the disk P1. Files K and L are copied to both the disk P1 and the tape T2.

Examples:

COPY X TO Y (KIND = DISK) [T];
    Copy file X from the user's default pack to the disk labelled Y, attaching task variable T to the copying task.

COPY X FROM Y (KIND = TAPE);
    Copies file X from tape Y to the user's default pack.

COPY X/Y, Z/= FROM P (KIND=DISK) TO T1 (KIND = TAPE),
TO T2 (KIND = TAPE);
    Copy file X/Y and all files under directory Z from the disk P to both tapes T1 and T2.

COPY X AS Y;
    Copy X from disk to disk, changing the name of the new file to Y.

COPY X FROM DISK TO DISK (HOSTNAME = HOSTB);
    Copy X on the system disk at own host to system disk at host HOSTB.

COPY X/= AS Y/= FROM DISK, A FROM B (KIND =DISK)
TO T (KIND = TAPE);
    Copy all files in the X directory from the system disk to tape T, changing all the names to be in the Y directory. Also, copy file A (without changing its name) from disk B to tape T.

COPY = FROM DISK TO T (KIND=TAPE);
    If the job is run with a usercode, copy all files in the user's directory from the system disk to tape T. If the job is run without a usercode, copy all files on the system disk to tape T.

COPY AND SET(COMPARE) = FROM T (KIND = TAPE)
TO D (KIND = DISK);
    Copy all files from tape T to disk D. Immediately after each file is copied to disk D, it is compared with the original file on tape T.

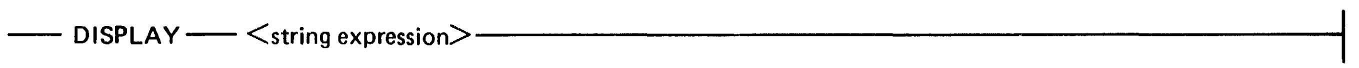
ADD X/Y FROM T (KIND = TAPE);
Copy file X/Y from tape T to disk, only if there is not already a resident disk file named X/Y.

ADD Z/= FROM T(KIND = TAPE) TO R (KIND = DISK), TO DISK;
May copy different files to R and to the system disk, depending on what is resident on each destination volume.

DISPLAY STATEMENT

The <DISPLAY statement> displays a <string expression> on the Operator Display Terminal (ODT).

<DISPLAY statement> syntax:



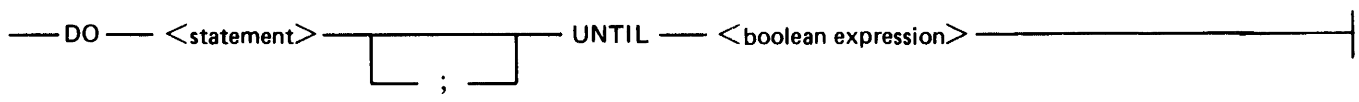
Example:

DISPLAY "BEGIN UPDATE RUN";

DO STATEMENT

The <DO statement> allows the user to execute a <statement> until a condition is TRUE.

<DO statement> syntax:



The <statement> following the keyword DO is executed, then the <boolean expression> is evaluated. If the expression is FALSE, the < statement> is executed again and the <boolean expression> is re-evaluated. This sequence continues until the value of the <boolean expression> is TRUE, at which time control passes to the next statement.

The <statement> is executed at least once.

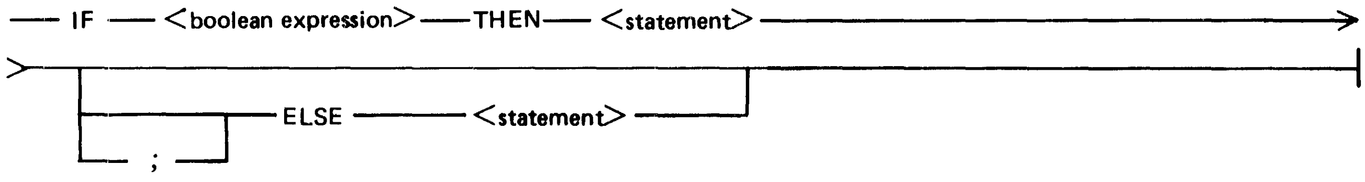
Example:

```
DO
  BEGIN
    A := A + 1;
    RUN X [T];
  END
UNTIL A = X;
```

IF STATEMENT

The <IF statement> allows the user to execute a <statement> only if a condition is TRUE.

<IF statement> syntax:



Each time the reserved-word ELSE appears in an <IF statement>, it is paired with the nearest preceding unpaired reserved-word THEN.

The <statement> following the reserved-word THEN is executed if the <boolean expression> is TRUE. In this case, the optional second < statement> is not executed.

If the <boolean expression> is FALSE, the <statement> following the reserved-word THEN is not executed. In this case, if the optional second < statement> is specified, it is executed; otherwise, control passes to the next executable statement.

Examples:

```
IF B THEN
  ABORT "B IS TRUE"

IF T IS COMPLETED THEN
  I := I+1;
ELSE
  I := 1

IF FILE X/Y ISNT RESIDENT THEN
  DISPLAY "NO FILE X/Y"
ELSE
  RUN X/Y

IF I=J THEN
  BEGIN
    RUN X;
    RUN Y
  END
ELSE
  RUN Z
```

INITIALIZE STATEMENT

The <INITIALIZE statement> causes all attributes in a <task id> to be set to their default value.

<INITIALIZE statement> syntax:

```
——INITIALIZE—— ( —— <task id> —— ) ————————————————————————————————————————|
```

The specified <task id> is reinitialized to a status of uninitiated and all other attributes are set to their default values.

NOTE

The default values are system defaults and not the values originally specified by the user. The default values are given in appendix A.

Example:

INITIALIZE (TI);

INSTRUCTION STATEMENT

The INSTRUCTION statement is used to supply job instructions to operators.

<INSTRUCTION statement> syntax:

```
                                          ┌←——————————————————————————————————————————————————┐
—— INSTRUCTION <instruction number> ——┴—/1500\—— <any EBCDIC character except semicolon (;)> ——┴————|
```

<instruction number> syntax:

```
——<integer constant primary> ——————————————————————————————————————————————|
```

The <instruction number> must be in the range 1 to 63 inclusive.

Each <INSTRUCTION statement> in a WFL job must have a unique <instruction number> and must be followed by a semicolon (;) character. Also, there must be at least one non-blank character between the <instruction number> and the semicolon.

At any point during a job execution, the operator can display an individual instruction by number. When an <INSTRUCTION statement> is encountered during execution of the job, it is marked as the most recently executed instruction. If the operator enters an IB ODT-command without an instruction number, the most recently executed instruction is displayed. If an < INSTRUCTION statement> was not executed, a message is displayed informing the operator of that fact.

Examples:

```
BEGIN JOB COMPILE/TESTS;

  INSTRUCTION 1 TESTTAPE IS IN TAPE RACK 3.;
  COPY & SET (COMPARE) =
    FROM TESTTAPE (KIND=TAPE) TO USERS (KIND=DISK);

  INSTRUCTION 2 IF T17 OR T17A WERE NOT COPIED FROM
    TESTTAPE TO USERS, PLEASE DS THIS JOB AND LEAVE
    JK A NOTE.;

  COMPILE TEST/17 WITH COBOL LIBRARY;
    COMPILER FILE CARD (TITLE=T17, KIND=DISK);
    FILE F (TITLE=T17A);

  IF FILE TEST/17 ISNT RESIDENT THEN
    ABORT "BAD COMPILE";
END JOB
```

During execution of the above job, the system will need tape TESTTAPE. If the operator asks for the most recent instruction, instruction 1 will be displayed, indicating where TESTTAPE can be found. Later, the job will need files T17 and T17A. An instruction request will display instruction 2, specifying what to do if T17 and T17A are not present.

MODIFY STATEMENT

The <MODIFY statement> is used to permanently change attributes within a program.

<MODIFY statement> syntax:

```
—— MODIFY ——<file title>—— ; ——<task equation list>————————————————|
```

The <task equation list> can contain only permanent attributes. (Refer to appendix A for information about the attributes that are permanently stored with the code file.)

A TASKFAULT condition occurs for any of the following errors:

File not present
Attempt to modify a non-executable file
Intname not present in code file

All other errors cause the job to terminate abnormally.

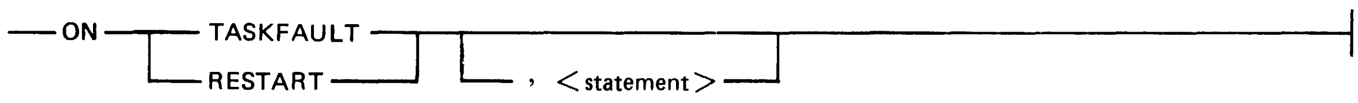
Examples:

```
MODIFY PAYROLL; PRIORITY = 5; PROTECTED

MODIFY ACCTSRCV;
  FILE INPUT (TITLE = TODAY/TASKS);
  PRIORITY = 7; SW1;
```

ON STATEMENT

The <ON statement> causes the job task to branch to the statement specified when a taskfault or restart occurs.

<ON statement> syntax:



A taskfault occurs when a task is abnormally terminated or a compilation terminates having detected syntax errors. A restart occurs when the system is clear/started and the job is automatically restarted by the operating system.

The statement ON TASKFAULT, <statement>; enables the taskfault condition. Subsequently, the abnormal termination of any task initiated by the job causes the statement to be executed. This includes operator or programmatic discontinuation as well as arithmetic faults. The statement ON TASKFAULT; disables the condition so that an abnormal task termination does not have any effect on the job.

Similarly, the statement ON RESTART, < statement>; enables the restart condition. Subsequently, the system being clear/started and the job automatically restarting causes the statement to be executed. The statement ON RESTART; disables the condition.

If a condition is enabled, then when that condition occurs, the statement part is executed as a subroutine. The ON condition which caused the statement to be executed is disabled immediately before executing the statement and returned to what it was when the condition occurred immediately after executing the statement. The job resumes execution at the point where it would have been if the condition had been disabled.

A "snapshot" of the job is taken at specific places during the execution of the job. These "snapshots" are taken: 1) before every task initiate, 2) after every synchronous task initiate, and 3) at the end of an ON RESTART, <statement>;. A "snapshot" is not taken if there are any tasks active which were initiated by the job (asynchronous tasks running). If the system is clear/started during the execution of a job, the job restarts at the most recent "snapshot." The values of all boolean, integer, and string variables are reset to their values at the point of restart (prior to invoking the statement of any enabled RESTART condition). Task variables are initialized to default values.

Only one ON TASKFAULT, <statement>; can be enabled at any given time. Also, only one ON RESTART, < statement>; can be enabled at any given time. Thus, any ON statement disables any previous ON statement. If a subroutine executes an ON statement, it disables any previous ON statement until the subroutine is finished or the ON condition is disabled. Exiting the subroutine or executing a disabling ON statement returns the condition to what it was before the subroutine was called.

Example 1:

```
BEGIN JOB J;
  SUBROUTINE SUB;
  BEGIN
    ON TASKFAULT,
        ABORT "SUB FAULT TAKEN";

    RUN X;   % if X aborts, "SUB FAULT TAKEN" is displayed
    ON TASKFAULT;
    RUN Y;   % if Y aborts, "JOB FAULT" is displayed
  END SUB;

  RUN A;   % if A aborts, no taskfault is executed
  ON TASKFAULT,
     ABORT "JOB FAULT";
  RUN B;   % if B aborts, "JOB FAULT" is displayed
  SUB;
END JOB
```

In example 2, "** SOMETHING ABORTED **" is displayed and C is run after either A or B aborts.

Example 2:

```
BEGIN JOB K;
  ON TASKFAULT,
    BEGIN
      DISPLAY "** SOMETHING ABORTED  **";

      RUN C;
    END;

  RUN A;
  RUN B;
END JOB
```

Example 3:

```
BEGIN JOB DRIVER;
  TASK T;
  INTEGER COUNTER;

  ON RESTART,
    BEGIN
      IF COUNTER = 1 THEN
        T (FILE CARD (TITLE = SOURCE/FILE, DISK));
      ELSE IF COUNTER = 2 THEN
        T (PRIORITY = 6);
    END;

  COUNTER := 1;
  T (FILE CARD (TITLE = SOURCE/FILE, DISK));
  COMPILE OBJECT/FILE WITH COBOL74 [T] LIBRARY;
  INITIALIZE (T);
  COUNTER := 2;
  T (PRIORITY = 6);
  RUN OBJECT/FILE [T];
END JOB
```

In example 4, the DUMMY/PROGRAM program waits for an accept message and then goes to EOT. The ZIP/IT program accepts a message, zips the string, and then goes to EOT.

Example 4:

```
BEGIN JOB RESTART/EXAMPLE;
  TASK T;
  STRING S;

  ON RESTART, DISPLAY "DUMMY/PROGRAM NOT RUNNING";

  RUN ONE;
  RUN TWO;

  ON RESTART, DISPLAY "DUMMY/PROGRAM RUNNING";

  PROCESS RUN DUMMY/PROGRAM [T]; % DUMMY/PROGRAM RUNS ASYNCHRONOUSLY

  RUN THREE;
  RUN FOUR;

  S := STRING ( T (MIXNUMBER), *) & "AX "; % GET DUMMY/PROGRAM'S
                                           % MIXNUMBER & APPEND "AX"

  RUN ZIP/IT (S); % TERMINATE DUMMY/PROGRAM NORMALLY

  WAIT (T); % WAIT FOR DUMMY/PROGRAM TO TERMINATE

  ON RESTART, DISPLAY "DUMMY/PROGRAM NOT RUNNING";

  RUN FIVE;
  RUN SIX;
END JOB
```

If the system is clear/started while ONE, TWO, FIVE, or SIX are executing, the job is restarted, the string DUMMY/PROGRAM NOT RUNNING is displayed, and the program which was executing is initiated again. If the system is clear/started while THREE, FOUR, or ZIP/IT are executing, the job is restarted, the string DUMMY/PROGRAM RUNNING is displayed, and the program named DUMMY/PROGRAM is initiated. This is because, before the WAIT statement, the initiation of DUMMY/PROGRAM is the last place where no other tasks are running.

PASSWORD STATEMENT

The <PASSWORD statement> is used to change the password associated with the usercode of the job.

<PASSWORD statement> syntax:

```
—— PASSWORD —— = —— <password> —— / —— <password> ——————————————————|
```

The first <password> must be the password for the current usercode of the job. The password associated with the current usercode of the job in the (SYSTEM)USERCODE file is changed to the second <password>.

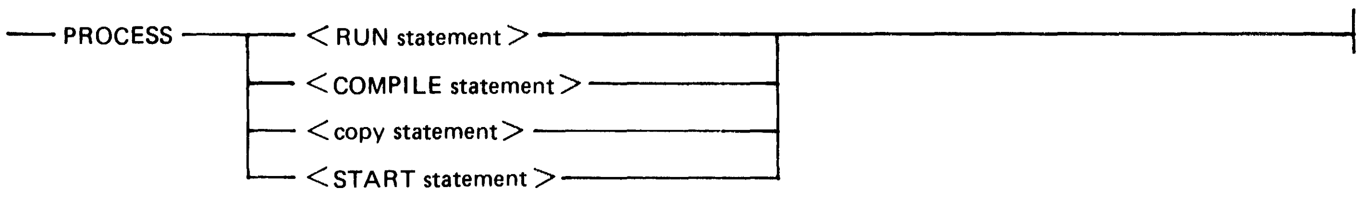
Example:

```
BEGIN JOB CHANGE/PASSWORD;
  USERCODE = ME/OLD;
%   job runs under usercode ME
  PASSWORD = OLD/NEW;
%   changes ME's password from OLD to NEW
END JOB
```

PROCESS STATEMENT

The PROCESS statement initiates tasks asynchronously.

<PROCESS statement> syntax:



If a task variable is attached, the job can subsequently wait for the process to terminate (see WAIT statement), monitor its execution (see IF statement), or alter its execution (see task assignment statement). The job does not terminate until all asynchronous tasks have terminated.

NOTE
The use of implied task variables requires more memory per job; therefore, it is recommended that a task variable be attached to all PROCESS statements.

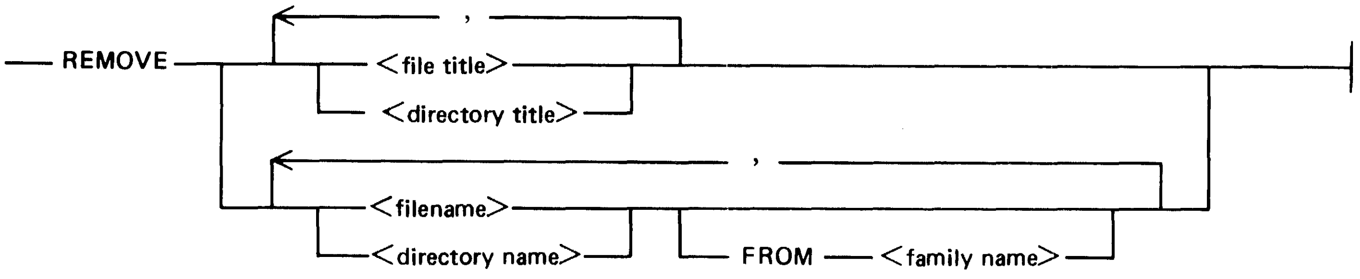
Example:

PROCESS COPY A TO B [T];

REMOVE STATEMENT

The <REMOVE statement> removes files from disk.

<REMOVE statement> syntax:



If a <directory title> or <directory name> is specified, all files in that directory are removed.

The FROM clause indicates that the list of files preceding it (but not any files prior to a preceding FROM clause) reside on the disk specified by <family name>.

If a <family name> is not specified and the file is present on the default family, the file on the default family will be removed. If the file is not present on the default family but is present on DISK, the file on DISK will be removed. If the file is present on both the default family and DISK, only the file on the default family will be removed. A directory is treated as a whole in the same manner.

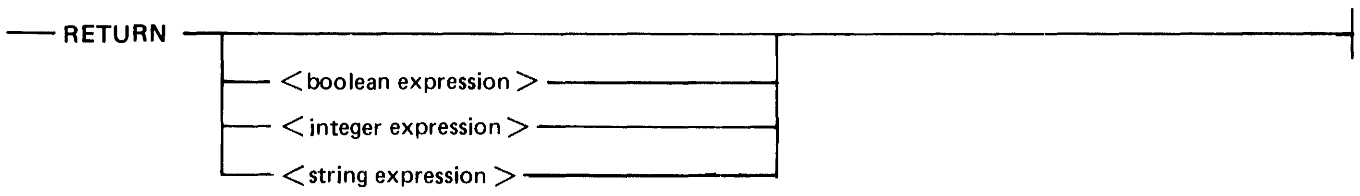
Examples:

```
STRING S1, S2;

REMOVE X;
%    Removes the file X.

REMOVE A/B ON USERS;
%    Removes the file A/B from the disk USERS.

REMOVE X,Y FROM MYPACK, X/Y FROM INVENTORY, Z;
%    Removes the files X and Y from the disk MYPACK.
%    Removes the file X/Y from the disk INVENTORY.
%    Removes the file Z from the default family.

REMOVE X, Y ON USERS FROM MYPACK; % Invalid syntax
%    This  <REMOVE statement> is not valid because  it  uses
%    both ON USERS and FROM MYPACK.

REMOVE X/=;
%    Removes all files in the X directory.

S1 := "A/=";
S2 := "MYPACK";
REMOVE #S1, #S1 ON #S2;
%    Removes  all files in the A directory from the  default
%    family  and all files in the A directory from the  disk
%    MYPACK.
```

RETURN STATEMENT

The <RETURN statement> terminates the execution of a subroutine.

<RETURN statement> syntax:



Execution control is returned to the statement following the <function invocation> or < subroutine invocation statement>.

An expression may not be supplied on a return from an ON statement, or from a subroutine. If returning from a function, the expression type being returned must be the same type as in the <return type> of the function. It is an error not to return an expression when returning from a function.

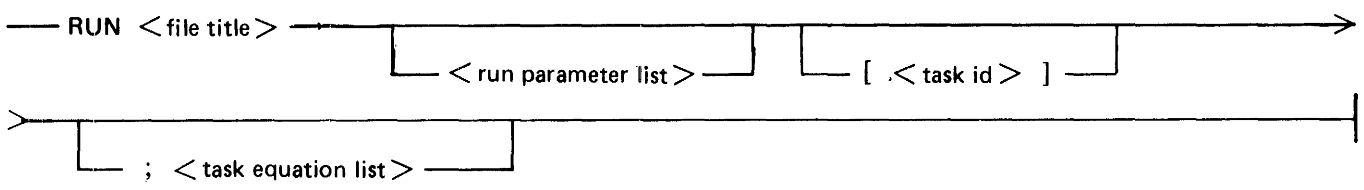
Example:

```
BEGIN JOB RETURN/EXAMPLE;
  BOOLEAN B;
  SUBROUTINE SUB;
    BEGIN
      RUN PROG/1;
      IF B THEN RETURN;
      RUN PROG/2;
    END;
  B := FALSE;
  SUB; % runs PROG/1 and PROG/2
  B := TRUE;
  SUB; % runs just PROG/1
END JOB


BEGIN JOB RETURN/EXPRESSION;
  STRING STR1, STR2;
  INTEGER LEN;
  STRING FUNCTION GETSTRING;
    BEGIN
      RETURN ACCEPT ("WAITING FOR AN INPUT STRING", OK);
    END GETSTRING;
  INTEGER FUNCTION MAX (INTEGER I, INTEGER J);
    BEGIN
      IF I GEQ J THEN
        RETURN I;
      ELSE
        RETURN J;
    END MAX;
  STR1 := GETSTRING;
  STR2 := GETSTRING;
  LEN := MAX (LENGTH (STR1), LENGTH (STR2));
  DISPLAY "THE MAXI LENGTH = " & STRING (LEN,*);
END JOB
```

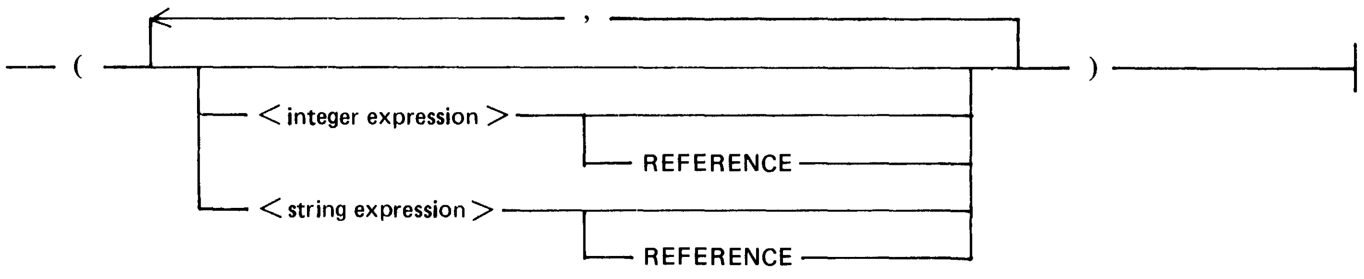
RUN STATEMENT

The <RUN statement> initiates a previously created code file.

<RUN statement> syntax:



<run parameter list> syntax:



The path through the <run parameter list> diagram that does not cross any token is an empty expression. The value of an empty expression is zero when assigned to an integer, a zero length string when assigned to a string variable, and blanks when assigned to a character variable.

NOTE

Empty expressions may only be passed to optional parameters declared within the code file.

When a task is initiated, the parameters in the <run parameter list> are compared to those in the code file denoted by <file title>. The number of parameters in <run parameter list> must be the same as, or less than, the number of parameters expected by the code file. If the <run parameter list> contains fewer parameters than expected by the code file, the <run parameter list> is treated as though there were sufficient empty expressions before the right parenthesis to match the number of parameters expected by the code file. There is a correspondence between parameters in <run parameter list> and those in the code file which is established by the position of the parameters in the lists. Each expression in <run parameter list> must be of a similar data type as its corresponding parameter in the code file. If the number of parameters in <run parameter list> is greater than the number of parameters in the code file, or any of the corresponding parameters do not have a similar type, or an empty expression is given and the code file does not allow for an optional parameter, the job is discontinued. By default, all parameters are passed "by value." The keyword REFERENCE indicates that the parameter is passed "by value/result" rather than "by value," and is valid only if the preceding expression is an identifier.

Parameters may be passed only to UPL, UPL2, SDL, SDL2, and Mark 13.0 COBOL74B code files. Parameters of type BOOLEAN may not be passed. A syntax error occurs if a BOOLEAN parameter is passed. Parameters to UPL, UPL2, SDL, and SDL2 code files must be a <string expression> and may not be passed "by value/result" (REFERENCE may not be specified). If both of these conditions are not met, a run-time error occurs. The parameters are passed as early AC messages. Parameters may only be passed to Mark 13.0 COBOL74B code files that contain the PROCEDURE DIVISION USING ... phrase. The parameters to Mark 13.0 COBOL74B programs are not passed as early AC messages. Valid parameter passes to Mark 13.0 COBOL74B programs are as follows: 1) parameters of type INTEGER may be passed to a PIC 9 field with USAGE of DISPLAY or COMPUTATIONAL (if a SIGN is present, it must be LEADING), and 2) parameters of type STRING may be passed to a PIC X field.

The <RUN statement> initiates a synchronous dependent task. The job waits for that task to complete before continuing to the next statement.

A list of task attribute assignments can be given to set attributes in the task. These values override any attributes contained in the code file. Refer to Task Attributes and File Equation in this section.

For parameters which are passed "by value," the expression is evaluated and the value is passed to the initiated task. When the task is completed, no value is passed back to the job by means of the run parameter. The value of the expression, or more likely identifier, is not changed.

For parameters which are passed "by value/result," as described above, the expression must be an identifier. The value of the identifier is passed to the initiated task. When the task completes, the value is copied from the task back to the job's run parameter. Thus, the run parameter of the job is updated when the task completes (EOT). For asynchronous tasks, two "sync points" have been provided where the value of parameters may be copied back from the task to the job. These syncpoints are: 1) when the job suspends the initiated task (that is, setting STATUS = SUSPENDED), and 2) at task EOT. This allows for information to be passed back and forth from a job to an asynchronous task. Parameter values are copied from the job to the asynchronous task when the job resumes the task; that is, setting STATUS = ACTIVE.

NOTE

Any attempt to SUSPEND a task which is currently waiting for a CALLed program to complete will be ignored.

Example:

```
TASK T, T1;

RUN X;

RUN A/B [T];  MAXCARDS = 500;

RUN A/B [T1];
  FILE F(BLOCKSIZE = 10, KIND=TAPEPE);
  FILE G(KIND=DISK);

RUN A/B; PRIORITY = 8;

RUN A/B [T]; PRIORITY = 9;
  % In this example, if task T contains a priority
  % attribute, it is overridden by the run-time priority
  % attribute (that is, PRIORITY = 9).

RUN DMPALL ("LIST USER4/PAYROLL/  A SKIP 50", "");
  % DMPALL is initiated and each string is passed as an
  % early AC message.

RUN COBOL74B/PROGRAM (24, MYSELF(TITLE));
  % Initiates COBOL74B/PROGRAM passing an integer 24
  % and a string containing the TITLE of the job.
```

The following example of a job initiating an asynchronous task and the passing of information back and forth from the job to the task. The task ADD/PARAMS is a COBOL74B program which requires three parameters (PIC 9(8)). The first two parameters are added together and the result put in the third parameter. The COBOL74B program notifies the job that the numbers have been added together by setting its own TASKVALUE to 1, CHANGE ATTRIBUTE TASKVALUE OF MYSELF TO 1, and executing a CAUSE AND RESET ATTRIBUTE EXCEPTIONEVENT OF MYJOB statement. The job notifies the COBOL74B task that another set of numbers is available to be added by setting TASKVALUE to 0 for the task. Thus the COBOL74B program must wait for its own TASKVALUE to be changed to 0 before continuing.

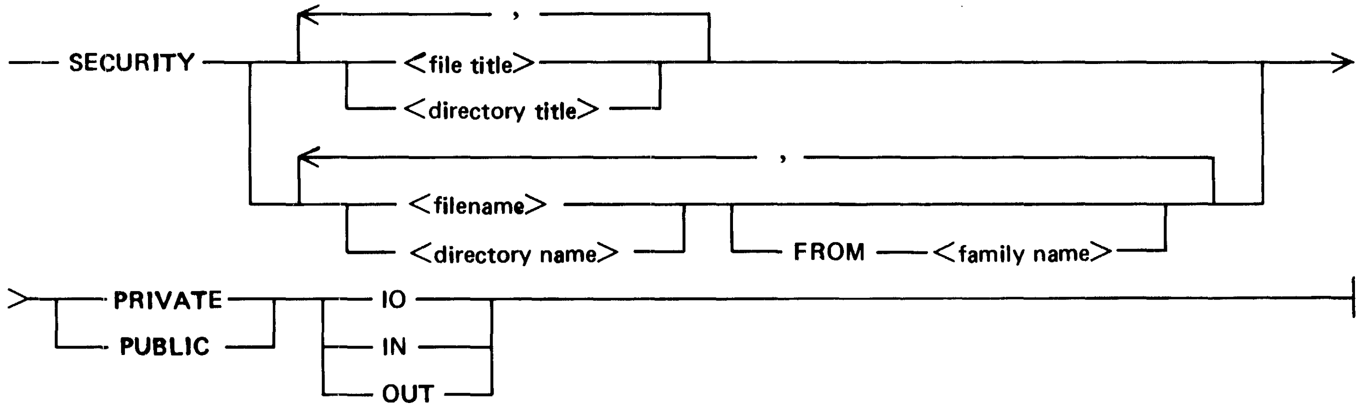
Example:

```
BEGIN JOB ASYNC/EXAMPLE;
  TASK T;
  INTEGER I1 := 1, I2 := 2, I3 := 44;

  PROCESS RUN ADD/PARAMS (I1 REFERENCE, I2 REFERENCE,
                         I3 REFERENCE) [T];

  WAIT (T (TASKVALUE) = 1); % WAIT FOR TASK TO ADD PARAMS
  T (STATUS = SUSPENDED); % SUSPEND TASK SO PARAMETER VALUES
                         % WILL BE RETURNED TO THE JOB
  IF I3 = 3 THEN
    DISPLAY "SUCCESS";
  ELSE
    DISPLAY "FAILURE";

  I1 := 14; % ASSIGN NEW VALUES
  I2 := 10; % TO BE PASSED TO THE TASK

  T (TASKVALUE = 0); % NOTIFY TASK TO ADD PARAMETERS AGAIN
  T (STATUS = ACTIVE); % ALLOW PROGRAM TO RUN AGAIN

  WAIT (T (TASKVALUE) = 1); % WAIT FOR TASK TO ADD PARAMS
  T (STATUS = SUSPENDED); % GET THE RESULTS BACK

  IF I3 = 24 THEN
    DISPLAY "SUCCESS AGAIN";
  ELSE
    DISPLAY "FAILURE AGAIN";

  T (STATUS = TERMINATED); % DS THE TASK

END JOB
```

SECURITY STATEMENT

The <SECURITY statement> changes the SECURITYTYPE and SECURITYUSE of files on disk.

<SECURITY statement> syntax:



If a directory is specified, the security of all files in that directory is changed.

The FROM clause indicates that the list of files preceding it (but not any files prior to a preceding FROM clause) reside on the disk specified by <family name>.

For a description of PUBLIC and PRIVATE refer to the description of the SECURITYTYPE file attribute in appendix B. For a description of IO, IN, or OUT, refer to the description of the SECURITYUSE file attribute in appendix B.

If a <family name> is not specified and the file is present on the default family, the security of the file on the default family is changed. If the file is not present on the default family but is present on DISK, the security of the file on DISK is changed. If the file is present on both the default family and DISK, the security of only the file on the default family is changed. A directory is treated as a whole in the same manner.

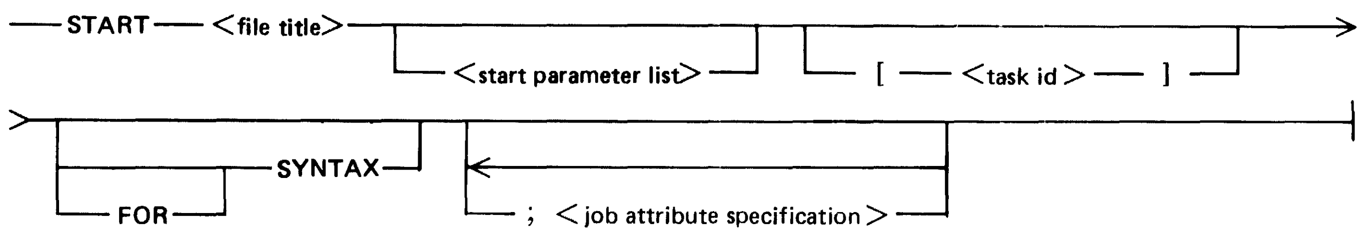
Examples:

```
STRING S1, S2;

SECURITY AB/XY PRIVATE IO;
%    Changes  the SECURITYTYPE of file AB/XY to PRIVATE  and
%    its SECURITYUSE to input and output.

SECURITY Z ON PACK PUBLIC IN;
%    Changes the SECURITYTYPE of file Z on the disk PACK  to
%    PUBLIC and its SECURITYUSE to input only.

S1 := "A/B";
S2 := "C/D ON MYPACK";
SECURITY #S1, #S2, PUBLIC IO;
%    Changes the SECURITYTYPE of the file A/B on the default
%    family  and file C/D on the disk MYPACK to  PUBLIC  and
%    their SECURITYUSE to input and output.
```

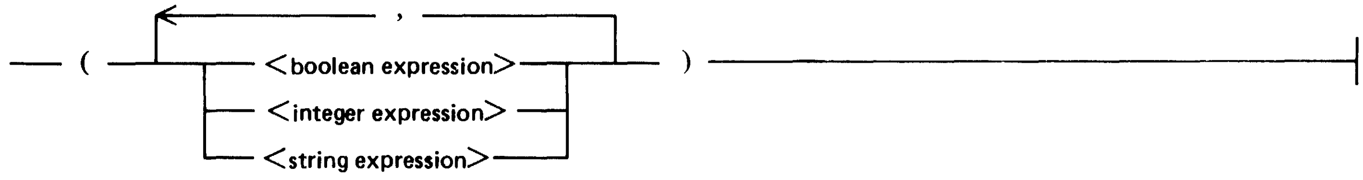
START STATEMENT

The <START statement> is used to initiate a job which is independent of the requesting task.

<START statement> syntax:



<start parameter list> syntax:



The <START statement> initiates a RUN of the SYSTEM/WFL program as a synchronous dependent task. After the SYSTEM/WFL program completes, the resultant WFL program is initiated as a job task. The job waits for SYSTEM/WFL, but not the started job, to complete before continuing to the next statement.

The started job is independent of the job that started it and runs according to its own attributes. If the started job does not have a USERCODE job attribute, the usercode of the job that performed the START operation is used.

The <file title> must refer to a file containing the source of a WFL <job> to be started.

The path through the <start parameter list> diagram that does not cross any token is an empty expression. The value of an empty expression is FALSE when assigned to a boolean variable, zero when assigned to an integer variable, and a zero length string when assigned to a string variable. However, if the job has user defined DEFAULT values, they override these default values.

NOTE

Empty expressions may only be passed to a corresponding job parameter which has been declared as OPTIONAL.

The expressions in the <start parameter list> are substituted for the parameter identifiers in the <job parameter list> of the started job. The number of parameters in the <start parameter list> must be less than, or equal to, the number of parameters in the <job parameter list> of the started job. A correspondence between the parameters in the <start parameter list> and the <job parameter list> is established by the position of the parameters in the lists. Each expression in the <start parameter list> must be the same type as the parameter identifier in the <job parameter list>. If the number of parameters in the <start parameter list> is less than the number of parameter identifiers in the <job parameter list>, the <start parameter list> is treated as though there were sufficient empty expressions before the right parenthesis to match the number of parameters in the <job parameter list>. If the empty expressions in <start parameter list> do not have corresponding OPTIONAL job parameters, or if the number of parameters in the <start parameter list> is greater than the number of job parameters, or if the parameters are not of the same type, the started job gets a syntax error. However, the starting job is not affected and continues executing. All parameters in the <start parameter list> are passed by value. The expressions are evaluated in the job performing the <START statement>. The actual parameters are passed to the SYSTEM/WFL program which passes these parameters to the job being started.

A <string expression> in the <start parameter list> must be less than or equal to 255 characters in length. If this restriction is violated, the starting job is abnormally terminated.

The <task id> is associated with the starting of the job, that is, the SYSTEM/WFL program.

The SYNTAX specification indicates that the started job is to be compiled for syntax only and execution is not to occur.

The <job attribute specification> specifies task attributes, a FETCH specification, a CLASS specification, or a STARTTIME specification for the initiated job. An out-of-range value in a <job attribute specification> causes a syntax error in the started job; the starting job is not affected and continues executing.

Example:

Disk File INVENTORY/RB contains WFL symbolic statements as follows:

```
BEGIN JOB REPORT/BACKUP (STRING REPORTNAME, STRING TAPENAME
 OPTIONAL DEFAULT = "ARCHIVE")
RUN #REPORTNAME/REPORT;
  FILE LINE (PRINTDISPOSITION=CLOSE);
  PRIORITY=4;
COPY #REPORTNAME/= FROM PARTS TO #TAPENAME (KIND=TAPE);
END JOB
```

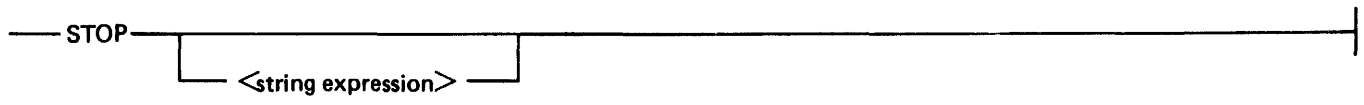
Sample invocation of the above job:

```
START INVENTORY/RB ("INVENTORY"); PRIORITY=7;
% The job REPORT/BACKUP will run at a priority of 7.
% The starting job will continue without waiting
% for the job REPORT/BACKUP.
```

STOP STATEMENT

The <STOP statement> terminates the job task.

<STOP statement> syntax:



The <STOP statement> differs from the <ABORT statement> in that an <ABORT statement> causes an abnormal termination of the job task, while the <STOP statement> causes a normal termination. The <string expression> specified is displayed prior to termination of the job.

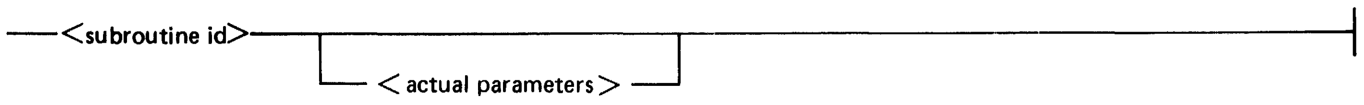
Example:

```
STOP "END OF UPDATE RUN";
```

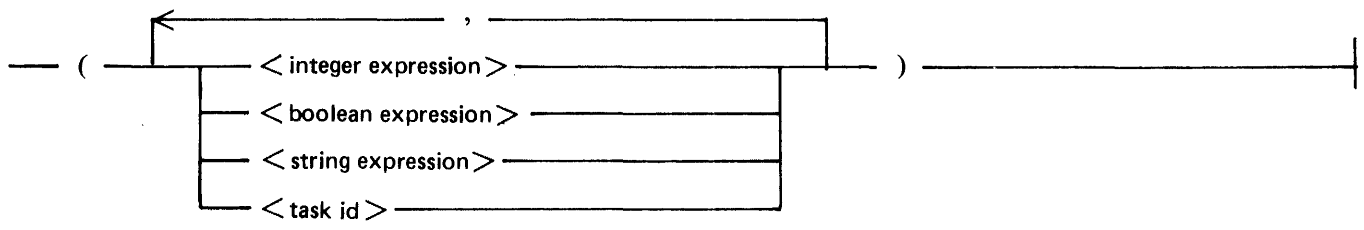
SUBROUTINE INVOCATION STATEMENT

The <subroutine invocation statement> transfers control to a subroutine.

<subroutine invocation statement> syntax:



< actual parameters> syntax:



The <subroutine invocation statement> causes the subroutine denoted by <subroutine id> to be executed. The statement may contain a list of <actual parameters> that are substituted for their corresponding <specified parameters>.

The number of parameters in <actual parameters> must be the same as the number of parameters in <specified parameters> when <subroutine id> was declared. There is a correspondence between <actual parameters> and <specified parameters> that is established by the position of the parameters in the lists. Each expression in <actual parameters> must be assignment compatible with its corrresponding specified parameter, that is, the type of the expression must be such that an <assignment statement> of that expression to the actual parameter would be allowed.

Example 1:

```
BEGIN JOB SUBROUTINE/EXAMPLE;
  SUBROUTINE SUB;
    BEGIN
      RUN PROG;
    END;
  SUB; % runs PROG
END JOB
```

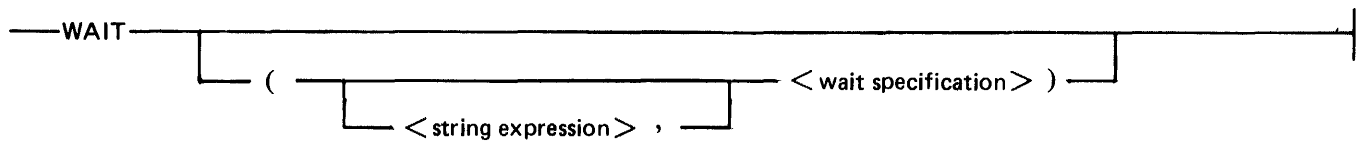
Example 2:

```
BEGIN JOB COMPILE/SUBROUTINE;
 SUBROUTINE COMPL (STRING COMPILERNAME, INTEGER MAX);
   BEGIN
     COMPILE X ITH #COMPILENAME LIBRARY;
        COMPILER ILE CARD (TITLE=X/DATA, DISK);
        COMPILER MAXTIME = MAX;
   END;
 COMPL ("PASCAL", 12*5);
END JOB
```

WAIT STATEMENT

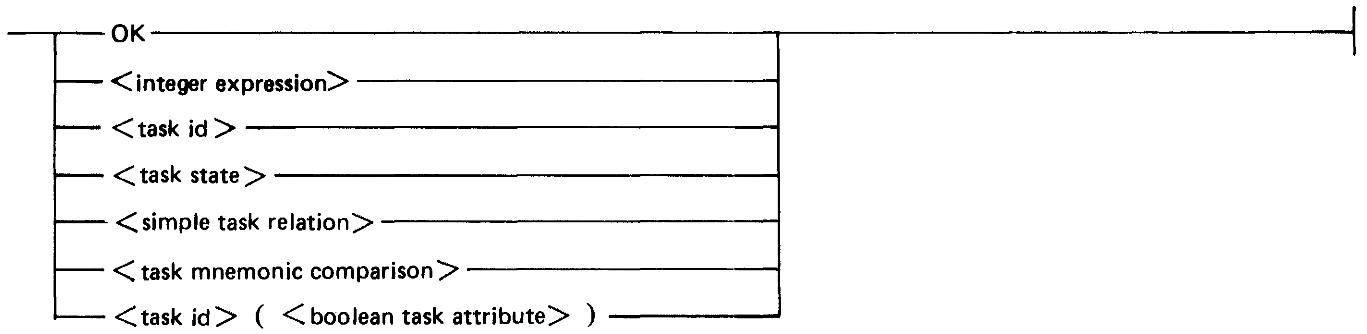
The <WAIT statement> suspends execution of the job task until a specified time period elapses or an event occurs.

<WAIT statement> syntax:

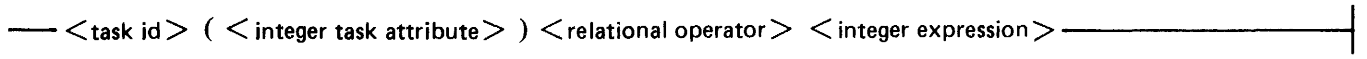


If a <string expression> appears in the <WAIT statement>, it is displayed on the Operator Display Terminal (ODT) prior to performing the wait.

<wait specification> syntax:



<simple task relation> syntax:



Semantics:

OK
: Suspends the job until the operator enters an OK ODT-command.

<integer expression>
: Suspends the job for <integer expression> seconds and then execution resumes.

<task id>
: Suspends the job until the task has completed.

<task state>
: Suspends the job until either the task is completed or the task achieves the given state.

<simple task relation>
: Suspends the job until either the task is completed or the task attribute satisfies the relation. The job waits for its own EXCEPTIONEVENT. The job does not resume execution until this event

<task mnemonic comparision>
: Suspends the job until either the task is complete or the task mnemonic comparison evaluates to TRUE. The job waits for its own EXCEPTIONEVENT to be caused. The job does not resume execution until this event is caused and one of the above conditions is TRUE.

<task id> (<boolean task attribute>)
: Suspends the job until either the task is completed or the <boolean task attribute> is TRUE. The job waits for its own EXCEPTIONEVENT to be caused. The job does not resume execution until this event is caused and one of the above conditions is TRUE.

no <wait specification>
: Suspends the job until its own EXCEPTIONEVENT occurs. An EXCEPTIONEVENT is caused every time the status of a subtask changes. An EXCEPTIONEVENT can also be caused by using the HI system command, or by programmatically setting the task attribute EXCEPTIONEVENT to TRUE. is caused and one of the above conditions is TRUE.

The EXCEPTIONEVENT of the job can be caused by the following:

1. A task that changes task state.
2. Entry of <job number> HI from ODT.
3. Programmatic cause from the task. For example, in a COBOL74B task:

```
CAUSE AND RESET ATTRIBUTE EXCEPTIONEVENT OF MYJOB.
```

Examples:

```
TASK T;

WAIT (5) % suspends the job for 5 seconds.

WAIT ("ENTER OK WHEN READY", OK);

WAIT (T); % suspends the job until task T has completed.
```

WHILE STATEMENT

The <WHILE statement> performs a <statement> while a condition is TRUE.

<WHILE statement> syntax:

```
—— WHILE —— <boolean expression> —— DO —— <statement> ——————————————————————|
```

The <boolean expression> is evaluated and if the result is TRUE, the <statement> following the keyword DO is executed. This sequence of events continues until the value becomes FALSE.

If the <boolean expression> is FALSE the first time it is evaluated, the <statement> is not executed at all.

Example:

```
WHILE I LSS 10 DO
  BEGIN
    RUN X[T] ;
    RUN Y[T] ;
    I := I + 1;
  END
```

SECTION 9
RESERVED, SPECIAL, AND KEY WORDS

Each word in WFL belongs to one of the following categories: RESERVED WORDS, SPECIAL WORDS, or KEY WORDS.

Note that a RESERVED, SPECIAL, or KEY WORD may be used as a <name constant> without any difficulty.

RESERVED WORDS

These words may not be used as an <identifier> in WFL.

BCL	BEGIN	BINARY	BOOLEAN
CONSTANT			
DATA			
EBCDIC	ELSE	END	
FALSE	FILE	FUNCTION	
INTEGER			
JOB			
LABEL			
REAL			
STRING	SUBROUTINE		
TASK	THEN	TRUE	
UNTIL			

SPECIAL WORDS

These words have a special meaning in WFL. They may be declared as an <identifier>; however, they lose their special meaning for the scope of the declaration.

ABORT	ACCEPT	ADD	
CASE	CHANGE	COMPILE	COPY
DECIMAL	DISPLAY	DO	DROP
GO			
HEAD	HEX		
IF	INITIALIZE	INSTRUCTION	
LENGTH			
MODIFY	MYJOB	MYSELF	
NOT			
OCTAL	OK	ON	
PASSWORD			
REMOVE	RETURN	RUN	
SECURITY	START	STOP	SYSTEM
TAIL	TAKE	TIMEDATE	
WAIT	WHILE		

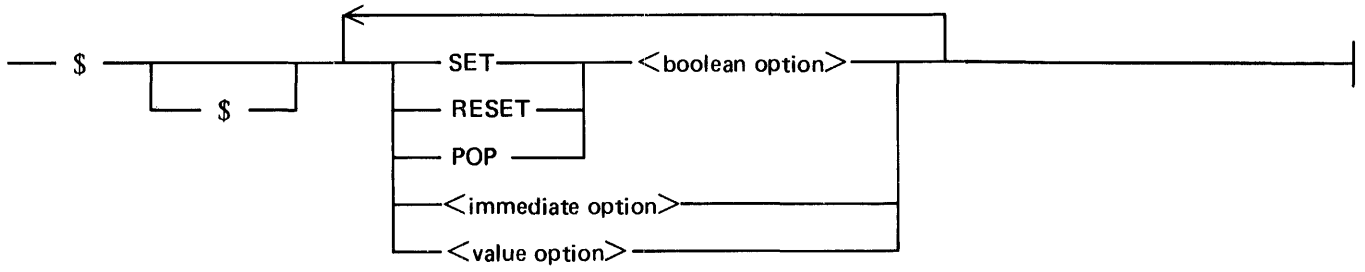
KEYWORDS

All words not listed as reserved or special are key words. Key words are context-sensitive. If they appear in the correct context, their predefined meaning is used; otherwise, they are assumed to be an <identifier>.

SECTION 10
COMPILER CONTROL IMAGES (CCI)

The compiler control image, <CCI>, is used to control certain options which are available during the compilation process. A <CCI> may appear anywhere in the source file and becomes active or inactive at that point.

<CCI> syntax:



The first dollar sign ($) must appear in the first character position of the record. If present, the second dollar sign must appear in the second character position of the record. The second dollar sign is allowed for compatibility with other languages but has no other function.

A <CCI> is terminated by the end of the record image.

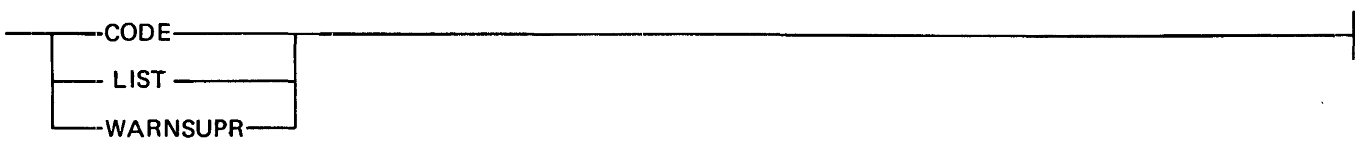
<Boolean option>s must be preceded by SET, RESET, or POP. Each <boolean option> has a stack which can contain up to 48 entries. Overflow and underflow of this stack are undetected.

SET
: SET causes the previous value of the specified <boolean option> to be stacked and the current value to be set to TRUE.

RESET
: RESET causes the previous value of the specified <boolean option> to be stacked and the current value to be set to FALSE.

POP
: POP causes the current value of the specified <boolean option> to be discarded and then set to the immediately previous value from the top of the <boolean option>'s stack.

<Boolean option> syntax:



CODE

When enabled, the CODE option causes the compiler to list the object code produced by the compilation process. The default value is FALSE.

LIST

When enabled, the LIST option causes the compiler to list the source language accepted for compilation. The default value is TRUE if the job is being compiled as the result of a START statement. A <job> entered as an ODT command will not be listed, but will appear in the job summary as operator input. If a job had syntax errors or it was compiled FOR SYNTAX, the listing is removed without being printed (as though the default for LIST was FALSE). Explicitly setting LIST to TRUE causes the listing to be printed even if there were syntax errors or the job was compiled FOR SYNTAX. For more information refer to Source Listing in section 12, Operation.

WARNSUPR

When enabled, the WARNSUPR option causes the compiler to suppress the printing of warning messages. The default value is FALSE. For more information refer to PORTABILITY WARNINGS, section 11.

<Immediate option> syntax:

```
——INCLUDE——<file title>————————————————————————————————————————|
```

INCLUDE

The INCLUDE option causes the compiler to suspend reading input to read input from the file specified by the <file title>. No CCI options may follow the <file title>. The file which is included may itself contain CCIs but must not contain INCLUDE CCIs. The source language images included as a result of enabling INCLUDE are listed if the LIST option is enabled.

<Value option> syntax:

```
——ERRORLIMIT—— = —— <integer constant>——————————————————————————|
```

ERRORLIMIT

The ERRORLIMIT option specifies the maximum number of errors allowed before a compilation is terminated. If the error limit is exceeded, the compiler informs the user that the error limit was exceeded and terminates the compile. The default value is 7 for a START or job entered at a terminal, and 100 otherwise.

SECTION 11
PORTABILITY WARNINGS

B 1000 WFL is a subset of B 5000/B 6000/B 7000/A Series WFL with certain exceptions. A warning message is given for using any construct that is not in the subset. Following are the constructs that receive warning messages.

1. The task attributes B1000MEMORY, B1000VIRTUALDISK, DEBUG, INVISIBLE, MAXTIME, and PROTECTED.
2. The file attributes AREABLOCKS, B1000AUDITED, BACKUPPERMITTED, DUMMYFILE, KIND=TAPECASSETTE, OTHERUSE=IN, and PRINTDISPOSITION.
3. <COMPILE statement> with a <task id> after the object code file title and a disposition of LIBRARY.
4. <Compile task equation list> with an attribute not preceded by the word COMPILER and a disposition of GO.
5. <START statement> with a SYNTAX clause or a <task equation list>.
6. <Copy options list>.
7. <Input volume spec> or < output volume spec> without an explicit KIND attribute.
8. <Creation file attr list>.
9. <Input volume attribute list> with VOLUMEINDEX.
10. <Output volume attribute list> with SAVEFACTOR or DENSITY.
11. <SECURITY statement> with a <directory name>.
12. <FUNCTION declaration>
13. DEFAULT values for OPTIONAL job parameters.

A non-suppressible warning message is printed at the end of the program if any of the above constructs were used in the program. If the WARNSUPR CCI option is set, the warning will be printed in the job summary, but will not be displayed. For more information, refer to Error Messages in section 12, Operation.

SECTION 12 OPERATION

This section describes job initiation and how an initiated job interfaces to the MCP, the SYSTEM/WFL program, and the SYSTEM/BACKUP program.

The following pages describe ODT commands that have WFL syntax. Some of the examples have the word "WFL" preceding them. This prefix may or may not be required. For more information, see the WFL system command and the WFL system option in the *B 1000 Systems System Software Operation Guide, Volume 1.*

JOB INITIATION

A job is initiated by using the START ODT command to refer to a file which contains the text of a WFL program or by entering the <job> as an ODT command. Both the START ODT command and a <job> may also be the object of a ZIP statement (or CALL SYSTEM WFL) in the various programming languages.

Both the START and the <job> ODT commands require that the command end with the end of the text, that is, another ODT command may not follow the START or <job> .

The START ODT command has the same semantics and pragmatics as the <START statement>. The syntax differs only in the fact that a [<task-id>] cannot appear in a START ODT-command. For more information, refer to the START Statement in section 8, Statements.

If a job is started under a non-privileged usercode and it has syntax errors, a security error message is displayed when attempting to remove the JOBPRT file.

RUN AND COMPILE AS ODT COMMANDS

Both the RUN and COMPILE statements can be entered as ODT commands. The syntax is the same as the corresponding WFL statement except for the following:

1. No task variables can be specified.
2. All expressions must be constants of the type of the expression. For example, an integer expression becomes an integer constant.
3. <Data specifications>s are not allowed.

Any tasks involved will be initiated as INDEPENDENT tasks.

MODIFY STATEMENT AS ODT COMMAND

The <MODIFY statement> can be entered directly as an ODT-command.

To distinguish the WFL <MODIFY statement> from a Control Card MODIFY command, the WFL <MODIFY statement> must be preceded by the word WFL.

There is a limit to the length of the <task equation list>. WFL is guaranteed to handle at least 3 lines of input from an ODT (240 characters of text). This is a worst case instance and, in general use, 10 lines to a full screen can be handled. An error message is displayed if the list is too long. In this case, the MODIFY must be split into separate statements.

Examples (including equivalent Control Card (CC) syntax):

```
WFL MODIFY (ZOT)CODEO ON DISK;
  PRIORITY = 5;
  FILE SOURCE (KIND=DISK);

CC MO DISK/(ZOT)/CODEO PR 5 FI SOURCE DISK;

WFL MODIFY *DMPALL;
  MAXLINES = 5000;
  PRIORITY = 1;
  SW1;
  FILE SPEC (TITLE = DISKFILE, KIND = DISK);
  FILE LISTFILE (PRINTDISPOSITION = EOJ);

CC MODIFY *DMPALL;
  % MAXLINES HAS NO EQUIVALENT
  PR 1; SW 1 = 1; FILE SPEC TITLE = DISKFILE DISK;
  % PRINTDISPOSITION = EOJ HAS NO EQUIVALENT
```

PASSWORD STATEMENT AS ODT COMMAND

The <PASSWORD statement> can be entered directly as an ODT-command. This allows users to change their passwords without writing a WFL job.

Example:

```
WFL PASSWORD = THEOLD/THENEW
PASSWORD = THENEW/SECRET
```

FILE MANAGEMENT STATEMENTS AS ODT COMMANDS

The <CHANGE statement>, <REMOVE statement>, and <SECURITY statement> may be entered directly as ODT commands.

In order to distinguish between the REMOVE ODT-command and the <REMOVE statement> as well as between the CHANGE ODT-command and the <CHANGE statement> , the <REMOVE statement> and <CHANGE statement> must be preceded by the word WFL.

Examples:

```
WFL REMOVE A/B ON USERS                        % WFL statement
WFL REMOVE X, Y FROM MYPACK                    % WFL statement
REMOVE USERS/A/B                               % ODT command
RE MYPACK/X/, MYPACK/Y/                        % ODT command
WFL CHANGE A/B ON USERS TO C/D                 % WFL statement
WFL CHANGE X TO X/X, Y TO Y/Y FROM MYPACK      % WFL statement
CHANGE USERS/A/B TO USERS/C/D                  % ODT command
CH MYPACK/X/ MYPACK/X/X,                       % ODT command
  MYPACK/Y/ MYPACK/Y/Y
SECURITY A/B ON USERS PRIVATE IO               % WFL statement
WFL SECURITY X, Y FROM MYPACK PUBLIC IN        % WFL statement
MH USERS/A/B SEC PRIVATE                       % ODT command
MH USERS/A/B SUS I.O                           % ODT command
```

SYSTEM/WFL PROGRAM

The MCP executes a copy of the SYSTEM/WFL program to process START ODT-commands and <job>s entered as ODT commands. The SYSTEM/WFL program continues to process jobs until it is terminated. Jobs are queued and processed sequentially, one at a time. The SYSTEM/WFL program does not wait for the job task to complete before processing the next job. When there are no jobs to be processed, the SYSTEM/WFL program waits for another START ODT-command or <job> to be entered.

While the SYSTEM/WFL program is waiting for another START ODT-command or <job> to be entered, it does not use any processor time and is rolled out of memory if memory space is needed. When rolled out, it uses less than 1000 bits of memory. With the SYSTEM/WFL program in this state, it is using a very small amount of resources and can process the next START ODT-command or < job> quickly (without having to go through beginning of job processing and initialization).

The SYSTEM/WFL program can be terminated by entering WFL EOJ, WFL END, or WFL STOP as as ODT-command. They all cause the program to terminate before processing the next ODT command.

A <START statement> (within a < job>) executes a copy of the SYSTEM/WFL program. This copy terminates when it finishes processing the one job referenced in the <START statement>.

A file titled "*JOB" & STRING (jobnumber, *) & "/CODE ON DISK" is created for each job initiated without syntax errors and without a job disposition of SYNTAX. The file is removed when the job task terminates.

ERROR MESSAGES

If a <job> is entered at the Operator Display Terminal (ODT) or if a START ODT-command or < job> is entered by means of a ZIP with LS set, error messages are displayed. For each error, the message contains the line of source text up to the point where the error occurred followed by a diagnostic message.

If a START ODT-command is entered at the ODT or if a START ODT-command or <job> is entered by means of a ZIP with LS not set, error diagnostic messages appear on the source listing following the source language with which they are associated.

JOB SUMMARY

A job summary consists of a WFL source program listing, a job log showing how the job ran, and any backup print files created by the job with a PRINTDISPOSITION of EOJ. Heading and trailing pages are printed which include the jobnumber, usercode, and title of the job.

If a job is to be initiated (it did not have syntax errors and it was not started FOR SYNTAX), printing of the job summary is based upon the setting of the JOBSUMMARY task attribute. All backup print files created by the job with a PRINTDISPOSITION of EOJ will be printed in the summary regardless of the setting of the JOBSUMMARY task attribute.

If the job will not be initiated and the LIST option is set, the job summary will be printed. If there were syntax errors and the errors were not displayed the job summary will be printed.

For more information refer to the LIST option in section 10, Compiler Control Images, the JOBSUMMARY task attribute in appendix A, and the PRINTDISPOSITION file attribute in appendix B.

Source Listing

The source listing consists of input source language (including that resulting from any INCLUDE options), diagnostic messages (error and warning), and summary information.

The source listing is in a file named " *JOBPRT/" & STRING (jobnumber, *) and will be on the backup designated disk. If the job summary is not printed, the source listing file is removed.

Job Log

The job log contains user input and MCP output messages associated with the job and all tasks initiated by the job.

It is in a file named "*JOBLOG/" & STRING (jobnumber, *) and will be on the backup designated disk. If the job summary is not printed, the log file is removed.

Printing the Job Summary

The job summary is automatically printed by the SYSTEM/BACKUP program only if the number of auto backup servers (AB) is greater than zero.

If auto backup is not used, the job summary may be printed by specifying JOB <jobnumber> in a PB ODT-command.

The job log may be printed by specifying JOBLOG/< jobnumber> in a PB ODT-command.

After the job summary is printed, the source listing and job log files are removed.

APPENDIX A
TASK ATTRIBUTES AND MNEMONICS IN WFL

Table A-1 contains a list of task attribute names, the attribute data type, when the attribute can be queried, when the attribute can be set, whether the attribute is permanently stored with the code file, and whether the attribute is inherited.

If an attribute can be set only when it is not INUSE, it may not be set when the task is scheduled, executing, or stopped.

If an attribute is inherited, a task, when initiated, receives the current value of the parent task's attribute, unless the parent task explicitly set the attribute prior to initiating the task. For more information, refer to Task Attributes and File Equation in section 8, Statements.

Table A-1. Task Attributes

Name	Type	Get	Set	Permanent	Inherited
ACCUMPROCTIME	Real	Anytime			
B1000MEMORY	Integer	Anytime	Not INUSE	Yes	
B1000VIRTUALDISK	Integer	Anytime	Not INUSE	Yes	
CHARGE	String	Anytime	Not INUSE	Yes	Yes
DEBUG	Boolean	Anytime	Anytime	Yes	
ELAPSEDTIME	Real	Anytime			
INVISIBLE	Boolean	Anytime	Anytime	Yes	
JOBNUMBER	Integer	Anytime			
JOBSUMMARY	Mnemonic	Anytime	Anytime	Yes	
MAXCARDS	Integer	Anytime	Not INUSE	Yes	Yes
MAXLINES	Integer	Anytime	Not INUSE	Yes	Yes
MAXPROCTIME	Real	Anytime	Not INUSE	Yes	Yes
MAXTIME	Real	Anytime	Not INUSE	Yes	
MAXWAIT	Real	Anytime	Anytime	Yes	Yes
MIXNUMBER	Integer	INUSE			
PRIORITY	Integer	Anytime	Anytime	Yes	Yes
PROTECTED	Boolean	Anytime	Not INUSE	Yes	Yes
STATUS	Mnemonic	Anytime	Anytime		
SW1-SW8	Boolean	Anytime	Anytime	Yes	
TASKVALUE	Integer	Anytime	Anytime	Yes	
TITLE	String	Anytime			
USERCODE	String	Anytime	Not INUSE		Yes

Following are descriptions of the task attributes.

ACCUMPROCTIME

The ACCUMPROCTIME task attribute is the amount of accumulated processor time for the task in seconds. This attribute is intended to be a measure of the processor resources consumed by the task. The default value is 0.

B1000MEMORY

This task attribute is the amount of dynamic memory in bits allocated for data overlays. The default value is 0.

B1000VIRTUALDISK

The B1000VIRTUALDISK task attribute is the number of disk segments assigned to hold non-memory-resident data overlays during execution. Virtual disk is the secondary store for overlayable data whereas dynamic memory is the primary store. The default value is 0.

CHARGE

The CHARGE task attribute is used in the logging function of the MCP. A task is terminated if it attempts to set CHARGE to an invalid <chargecode> for a particular installation. (See also <chargecode> in section 5, Task Attributes). The default value is the charge number of the job.

DEBUG

This task attribute invokes run-time debugging in COBOL74 programs. The default value is FALSE.

ELAPSEDTIME

The ELAPSEDTIME task attribute is the elapsed clock time in seconds since task initiation. The default value is 0.

INVISIBLE

This task attribute prevents the task from appearing in a display of the mix when it is active. Only the mix commands (MX and WY) containing the ALL qualifier will cause the INVISIBLE tasks to appear in the mix. The default value is FALSE.

JOBNUMBER

The JOBNUMBER task attribute is the MIXNUMBER of the job task under which the task is being run.

JOBSUMMARY

This task attribute provides a means by which the job summary can be suppressed or made available. The use of "job summary" here includes WFL statements and the ODT log of the job.

Mnemonic values are:

UNCONDITIONAL
The full job summary is printed unconditionally.

CONDITIONAL
The job summary is printed only if a task terminates abnormally, a compile detected syntax errors, or a task created printer backup file(s) with PRINTDISPOSITION set to EOJ.

SUPPRESSED
No job summary is printed (even if backup files are produced). All backup files created within the job with PRINTDISPOSITION set to EOJ are printed bound together with the usual heading and trailing pages.

The default value is UNCONDITIONAL, however, a site may change the default to CONDITIONAL by setting SYSTEM/WFL SW1 = TRUE.

The value of this attribute is not used until EOJ. Therefore, JOBSUMMARY can be set a number of times, but only the last setting before EOJ has meaning. Because this attribute has meaning only to a job and not to a task, the task designator MYJOB should be used to set JOBSUMMARY rather than MY-SELF.

MAXCARDS

The MAXCARDS task attribute is the maximum number of cards which may be punched by a task and its dependent tasks. When the maximum is reached, the task is abnormally terminated. A task inherits a value for MAXCARDS which is the MAXCARDS from the parent task less the number of cards punched by the parent task (and its dependent tasks). Explicitly setting MAXCARDS assigns the mini-mum of the amount requested or the amount remaining for the parent task. The default value is 0.

MAXLINES

This task attribute is the maximum number of lines which may be printed by the task and its dependent tasks. When the maximum is reached, the task is abnormally terminated. A task inherits a value for MAXLINES which is the MAXLINES from the parent task less the number of lines printed by the parent task (and its dependent tasks). Explicitly setting MAXLINES assigns the minimum of the amount re-quested or the amount remaining for the parent task. The default value is 0.

MAXPROCTIME

The MAXPROCTIME task attribute is the maximum amount of accumulated processor time in seconds which may be used by the task and its dependent tasks. The task is abnormally terminated when the max-imum is reached. A task inherits a value for MAXPROCTIME which is the MAXPROCTIME from the parent task less the amount of processor time used by the parent task (and its dependent tasks). Explicitly setting MAXPROCTIME assigns the minimum of the amount requested or the amount remaining for the parent task. The default value is 0.

MAXTIME

This task attribute is the maximum amount of elapsed clock time in seconds which the task may run. The task is abnormally terminated when the maximum is reached. The default value is 0.

MAXWAIT

The MAXWAIT task attribute is the maximum amount of time in seconds a task will wait for certain DMS functions to complete. Explicitly setting MAXWAIT assigns the minimum of the amount requested or the MAXWAIT of the parent task. The default value is 0.

MIXNUMBER

This task attribute is the MIX number of the task.

PRIORITY

The PRIORITY task attribute is the processor priority for the task. Priority informs the system of the urgency with which the task should be completed. The integer must be in the range 0 to 15 inclusive. The default value is the priority at which the job is running.

PROTECTED

This task attribute prevents certain keyboard commands from affecting an initiated task. For more information refer to the LP ODT-command in the *B 1000 Systems System Software Operation Guide, Volume 1*. The default value is the same as the PROTECTED attribute of the job.

STATUS

This task attribute starts, stops, or aborts an active task. See Task State in section 7 for a more extensive method to query the status of a task. Assignments to inactive task variables result in run-time errors.

ACTIVE
    The task is activated. The ACTIVE mnemonic has no effect unless the task is STOPPED.

BADINITIATE
    This is a read only mnemonic. The task failed initiation.

NEVERUSED
    This is a read only mnemonic. The task variable has not been used for a task initiation.

SCHEDULED
    This is a read only mnemonic. The task is in the schedule.

SUSPENDED
    The task is suspended.

TERMINATED
    The task is terminated (DSed or DPed).

The STATUS task attribute may not be used in the task equation list of a task initiation.

SW1, SW2, SW3, SW4, SW5, SW6, SW7, SW8

The eight switch attributes allow the user to alter the path of control of a task. They are a set of eight Boolean values that can be interrogated in various language dependent ways. For more information refer to the B 1000 language manuals. The default values are FALSE.

TASKVALUE

This attribute may be set or tested for specific purposes as desired by the programmer. Values are specified by the programmer. This attribute is generally used as a means of communication between processes that share a task variable. The default value is 0.

TITLE

The TITLE task attribute is the title of the task. The default value is "".

USERCODE

The USERCODE task attribute is the usercode under which the task is run. When an attempt is made to set USERCODE to a non-existent <user specification>, a run-time error occurs. The USERCODE returned by this attribute will not contain the <password>. The default value is the USERCODE of the job.

APPENDIX B
FILE ATTRIBUTES AND MNEMONICS IN WFL

Each of the file attributes is described in this appendix.

AREABLOCKS

This attribute is for disk files only, and is type Integer.

The value of the attribute AREABLOCKS is the number of blocks (physical records) in an area of the disk file. When a file is created, this value is associated with the physical file.

The default value is 100.

AREAS

The AREAS attribute is valid for disk files only, and is type Integer.

The value of the attribute AREAS is the maximum number of areas (or rows) a disk file can allocate.

The AREAS attribute may be set only when the file is closed. If AREAS is zero, the default value of 25 is used when creating a new disk file. The maximum value is 105. The setting of the AREAS attribute is ignored when opening a permanent disk file, or reopening a file closed with retention. The AREAS attribute can be overridden when expanding the end of file if the attribute FLEXIBLE has been set to TRUE.

B1000AUDITED

This attribute is type BOOLEAN.

If the attribute B1000AUDITED is TRUE, the user program is held waiting until each I/O operation to this file is complete. The B1000AUDITED attribute may be set only when the file is closed.

The default value is FALSE.

BACKUPKIND

This attribute is valid for printer and punch files, and is type Mnemonic.

The BACKUPKIND attribute indicates the peripheral associated with a logical file as an intermediate peripheral. The I/O operations of the logical file take place on the intermediate file but all the restrictions and capabilities of the peripheral specified by the KIND attribute are applied to the logical file. After the logical file is closed, the intermediate file may be transferred to a peripheral as specified by the KIND attribute. The TITLE attribute specifies the TITLE of the file when it is ultimately transferred to the peripheral specified by the KIND attribute. The title of the intermediate file may be determined with the TITLE and USERBACKUPNAME attributes.

The mnemonic values are:

DISK
: Backup peripheral is DISK.

TAPE
: Backup peripheral is TAPE.

DONTCARE
: Backup peripheral is site dependent. For further information, refer to the PBD and PBT system options in the *B 1000 Systems System Software Operation Guide, Volume 1.*

BACKUPPERMITTED

This attribute is valid for printer and punchfiles, and is type Mnemonic.

The BACKUPPERMITTED attribute indicates whether an intermediate peripheral may be associated with the logical file.

The mnemonic values are:

DONTBACKUP
: No intermediate peripheral usage is allowed.

DONTCARE
: Intermediate peripheral usage is allowed. A task which is part of a WFL job will use an intermediate peripheral for printer files so that the file is included in the job summary.

MUSTBACKUP
: Intermediate peripheral usage is required.

BLOCKSIZE

The BLOCKSIZE attribute is type Integer.

The values of the BLOCKSIZE attribute is the length of a block in FRAMESIZE units. BLOCKSIZE may be set only when the file is closed. If BLOCKSIZE is less than MAXRECSIZE, it is set to MAXRECSIZE when the file is opened. The default value of BLOCKSIZE is dependent upon the physical unit (KIND) assigned to the file and the value of the attribute MAXRECSIZE. Refer to the discussion under the MAXRECSIZE attribute in this appendix.

BLOCKSTRUCTURE

This attribute is type Mnemonic.

The attribute BLOCKSTRUCTURE specifies the format of the records and the structure of the file.

The mnemonics values are:

FIXED
: Blocked or unblocked fixed-length records.

VARIABLE
: Variable-length records. The record length is contained in the first four characters of the record.

The default value is FIXED.

BUFFERS

This attribute is type Integer.

The attribute BUFFERS specifies the number of buffers assigned to the file. The maximum number of buffers which may be assigned to a file is 255. Only in exceptional conditions do more than two buffers add to the efficiency of the I/O operations of a file.

If the number of buffers is not specified, a maximum of two buffers will be assigned.

DEPENDENTSPECS

This attribute is type Boolean.

If the attribute DEPENDENTSPECS is TRUE, the format of the records and the structure of the logical file are to be determined by the structure of the associated labeled permanent file; that is to say, the attributes BLOCKSTRUCTURE, MINRECSIZE, MAXRECSIZE, BLOCKSIZE, and FRAMESIZE are changed to agree with the values used to create the permanent file. If no permanent file is associated with the logical file (that is, a new file is being created), or if the permanent file is unlabeled, the attribute DEPENDENTSPECS is ignored.

DIRECTION

This attribute is valid for tape and paper tape reader files, and is type Mnemonic.

The attribute DIRECTION indicates the direction in which records are accessed from a file.

The mnemonics values are:

FORWARD

REVERSE

The default value is FORWARD.

Only BLOCKSTRUCTURE equal to FIXED files can be read in a reverse direction.

DUMMYFILE

This attribute is type Boolean.

When the attribute DUMMYFILE is TRUE, any READ from the file causes the END-OF-FILE condition, any WRITE to the file causes the END-OF-FILE condition if it is tested, otherwise it is a no-op. OPEN returns no errors and CLOSE returns no errors. All other I/O operations function as if the file was a non-resident optional file to which the operator responded with an OF ODT-command.

The default value for DUMMYFILE is FALSE.

EXTMODE

This attribute is type Mnemonic.

The attribute EXTMODE specifies the external or physical character size (mode) of the records in a file.

The mnemonic values are:

BINARY
: For card files 12 bits per column, 960 bits per record.

EBCDIC
: 8-bit.

ASCII
: 8-bit.

FAMILYNAME

This attribute is valid for disk files only, and is type String.

The attribute FAMILYNAME indicates the family on which the physical file is located. FAMILYNAME must be a <name>. The default FAMILYNAME is "", implying system disk, unless the task is running under a USERCODE with a default PACKID. In that case, FAMILYNAME "" will imply the USERCODE default PACKID.

FILEKIND

This attribute is valid for disk files only, is type Mnemonic, and may be modified at any time.

The FILEKIND attribute describes the internal structure and/or purpose of a record of a disk file. The mnemonic values are:

Data files:
- DATA
- BASICSYMBOL
- COBOLSYMBOL
- DASDALSYMBOL
- FORTRANSYMBOL
- NDLSYMBOL
- SEQDATA
- JOBSYMBOL
- RPGSYMBOL
- FORTRAN77SYMBOL
- PASCALSYMBOL
- SORTSYMBOL
- COBOL74SYMBOL

Code files:
- CODEFILE
- BASICCODE
- COBOLCODE
- FORTRANCODE
- NDLCODE
- JOBCODE
- INTRINSICSFILE
- MCPCODEFILE
- RPGCODE
- FORTRAN77CODE
- PASCALCODE
- SORTCODE
- COBOL74CODE
- COBOLINTERPRETER
- FORTRANINTERPRETER
- RPGINTERPRETER
- INTERPRETERFILE
- SDLINTERPRETER
- SDL2INTERPRETER
- COBOL74INTERPRETER
- FORTRAN77INTERPRETER
- BASICINTERPRETER
- IBASICINTERPRETER
- PASCALINTERPRETER
- B500INTERPRETER
- IBM1400INTERPRETER

System files:
- DIRECTORY
- BACKUPDISK
- BACKUPPUNCH
- CONTROLDECK

The default value is DATA.

NOTES

1. Only data files are expected to be portable between systems.
2. The type of a disk file, as defined for CANDE, is similar to the FILEKIND attribute. The CANDE type omits the "SYMBOL" or "CODE" suffix and only allows a subset of the FILEKINDs.

FILENAME

This attribute is type String. It must be a valid < filename>.

The attribute FILENAME is an external filename and is used to associate a logical file with a physical or permanent file. The default value is the value of the INTNAME attribute.

FLEXIBLE

This attribute is valid for disk files only, and is type Boolean.

The attribute FLEXIBLE indicates whether or not a disk file can be allocated more areas, if needed, than the number originally specified by the AREAS attribute. The setting of FLEXIBLE is ignored if the file has been crunched or ALLOCATE.ALL.AT.OPEN is true.

Setting FLEXIBLE is equivalent to requesting a maximum of 105 AREAS.

FRAMESIZE

The FRAMESIZE attribute is type Integer.

The attribute FRAMESIZE indicates the number of bits to be transferred as a unit of data. The values of the attributes MINRECSIZE, MAXRECSIZE, BLOCKSIZE, and AREALENGTH are expressed in FRAMESIZE units.

The default value is 8.

HOSTNAME

This attribute is type String.

The HOSTNAME attribute specifies the logical system host name in a BNA network at which the physical file is expected to reside.

KIND

The KIND attribute is type Mnemonic.

The KIND attribute indicates the peripheral associated with the logical file. Each logical file has exactly one value for KIND. Peripherals may be specified in two ways: a "specific" KIND which indicates only that peripheral and a "general" KIND which indicates a set of specified peripherals. The default value for KIND is language dependent.

The mnemonic values are:

DISK
Any mass storage peripheral.

PAPERREADER
Paper tape reader.

PAPERPUNCH
Paper tape punch.

PORT

READER
Any card reader.

PUNCH
Any card punch.

PRINTER
Any line printer.

READERSORTER
Any MICR or OCR reader and document sorter.

REMOTE
Data communications station.

TAPE
Any magnetic tape.

TAPE7
7-track magnetic tape.

TAPE9
9-track NRZ magnetic tape (DENSITY = BPI800).

TAPEPE
9-track Phase Encoded magnetic tape (DENSITY = BPI1600).

TAPECASSETTE
Magnetic tape cassette.

LABEL

This attribute is type Mnemonic.

The attribute LABEL specifies whether or not the file has label records.

The mnemonic values are:

STANDARD
ANSI labels are written as the beginning and ending records of the file.

OMITTED
Label records are not included in the file.

The value of LABEL is ignored for Data Comm files and the value OMITTED is used.

The default value is STANDARD.

MAXRECSIZE

This attribute is type Integer.

The attribute MAXRECSIZE specifies the maximum size of records in the logical file in FRAMESIZE units.

The attributes MAXRECSIZE, MINRECSIZE, BLOCKSTRUCTURE, and KIND are closely related. MAXRECSIZE must be less than or equal to BLOCKSIZE. If MAXRECSIZE is larger than BLOCKSIZE, BLOCKSIZE will be changed to MAXRECSIZE. If BLOCKSTRUCTURE is equal to FIXED, BLOCKSIZE must be a multiple of MAXRECSIZE. If MINRECSIZE is set greater than MAXRECSIZE, it will be set to MAXRECSIZE. If MAXRECSIZE is equal to zero and BLOCKSIZE is greater than zero, the value of MAXRECSIZE is set to the value of BLOCKSIZE.

MAXSUBFILES

The MAXSUBFILES attribute is valid for Ports only, and is type Integer.

The MAXSUBFILES attribute defines the maximum number of subfiles that the PORT may have open at any given time.

The largest value for MAXSUBFILES is 255.

MINRECSIZE

This attribute is type Integer.

The attribute MINRECSIZE specifies the minimum size of records in the logical file in FRAMESIZE units. If MINRECSIZE is left unset or is set to zero, a default value will be assigned when the file is opened, which depends upon the value of the attribute MAXRECSIZE. If MINRECSIZE was set greater than the value of MAXRECSIZE, it will be set equal to MAXRECSIZE.

The minimum record size, used by the logical I/O subsystem for deblocking the file, is determined by taking the largest of: 1) logical minimum record size (MINRECSIZE), 2) the minimum record size used when creating the physical file, or 3) the minimum allowable record size (which is dependent upon the value of the BLOCKSTRUCTURE attribute).

Files with BLOCKSTRUCTURE other than FIXED require the minimum record size to be large enough to contain the record length information of 4 characters. If MINRECSIZE is set smaller than 4 characters, its value is changed to 4. This attribute has meaning for variable length records only.

MYNAME

This attribute is valid for Ports only, and is type String.

The MYNAME attribute is the name by which a process wishes to be known when opening a port.

MYUSE

The MYUSE attribute is type Mnemonic.

The attribute MYUSE specifies how the file will be used.

The mnemonic values are:

IN
    Input only.

OUT
    Output only.

IO
    Both Input and Output.

NEWFILE

This attribute is type Boolean.

When a file is opened and can potentially be assigned to a device that is capable of both input and output, such as disk or tape, the NEWFILE attribute determines if an existing permanent file is desired (NEWFILE=FALSE) or a file is being newly created (NEWFILE=TRUE).

OPTIONAL

The OPTIONAL attribute is type Boolean.

The attribute OPTIONAL indicates whether or not (TRUE or FALSE) the assignment of a permanent file is optional. If the permanent file described by the attributes TITLE, KIND, and so forth is not present when the file is opened, a "NO FILE" message is sent to the operator display terminal and the program is suspended. If OPTIONAL is TRUE, the operator may respond with an OF ODT-command and the program proceeds without a physical file assigned to the logical file.

The default value for OPTIONAL is FALSE.

OTHERUSE

This attribute is type Mnemonic.

The attribute OTHERUSE specifies how the file may be used by other programs during the time that this program has the file open.

The mnemonic values are:

IO	Both Input and Output.
IN	Input only.
SECURED	Neither input nor output.

The default value is IO.

PAGESIZE

The PAGESIZE attribute is valid for printer files only, and is type Integer.

The attribute PAGESIZE indicates the number of lines on a logical page. PAGESIZE can have a value between zero (0) and 255 inclusive.

PAGESIZE cannot change from zero to non-zero, or vice versa, when the file is open.

PARITY

This attribute is valid for tape and paper tape punch files only, and is type Mnemonic.

The attribute PARITY specifies the parity used on the file.

The mnemonic values are:

ODD	Binary or standard parity.
EVEN	Alpha or non-standard parity.

PARITY may be set to EVEN for only 7-track magnetic tape and paper tape files.

The default value is ODD.

PRINTCOPIES

This attribute is valid for printer and punch files, and is type Integer.

The attribute PRINTCOPIES specifies the number of copies of the file to be printed. The maximum number of PRINTCOPIES is 63.

The default value is one copy.

PRINTDISPOSITION

This attribute is valid for printer files only, and is type Mnemonic.

The attribute PRINTDISPOSITION specifies whether and when a backup print file is to be automatically printed.

The mnemonic values are:

DONTPRINT
    The file is not automatically printed.

CLOSE
    The file is automatically printed as soon as the file is closed, provided the maximum number of auto backup servers (AB) is greater than zero.

EOJ
    The file is included in the job summary if it belongs to a task within a WFL job. Otherwise; EOJ is the same as CLOSE.

The default value is EOJ.

PROTECTION

The PROTECTION attribute is valid for disk files only, and is type Mnemonic.

The attribute PROTECTION indicates the amount of extra effort desired to preserve a file in case of a system failure. The mnemonic values are:

TEMPORARY
    A new disk file is not retained when the program is discontinued, unless the file is explicitly closed with an overriding close statement. If the disk file is locked or crunched, an entry is then made in the directory and the file is saved.

SAVE
    An entry in the directory is made immediately when the disk file is opened. The file becomes a permanent file and remains on disk unless the file is explicitly purged.

PROTECTED
    An entry in the directory is made immediately when the disk file is opened. The file becomes a permanent file and remains on disk unless the file is explicitly purged. All PROTECTED files are also SAVE. In addition, conceptually, a known pattern is written to disk in order to be able to recover the end of file pointer. The actual process is transparent to the user. Please refer to appendix C of the *B 1000 Systems System Software Operation Guide Volume 1,* for more information.

The default value is TEMPORARY. Files with an ORGANIZATION of RELATIVE or INDEXED will always have a PROTECTION of SAVE.

SAVEFACTOR

This attribute is valid for disk and tape files, and is type Integer.

The attribute SAVEFACTOR indicates the expiration date of a file in terms of the number of days past the creation date.

The system does not purge a tape whose SAVEFACTOR has expired unless explicitly told to do so.

The SAVEFACTOR has no system-defined meaning for a disk file.

SECURITYTYPE

This attribute is valid for disk or port files only, and is type Mnemonic.

For disk files, the attribute SECURITYTYPE specifies what users, apart from the owner (creator) of a permanent disk file (as identified by the USERCODE), may access the file.

For port files, the attribute SECURITYTYPE specifies the class of users who may access the port. The values for SECURITYTYPE are:

PRIVATE
: The file may be accessed by only the owner or a privileged user.

PUBLIC
: The file may be accessed by any user.

The default value for disk files is PRIVATE if the file is created under a usercode, and PUBLIC otherwise. The default value for port files is PRIVATE.

SECURITYUSE

This attribute is valid for disk files only, and is type Mnemonic.

The attribute SECURITYUSE specifies the manner in which a disk file may be accessed.

The values for SECURITYUSE are:

IO
: Read and write access is allowed.

IN
: Only read access is allowed.

OUT
: Only write access is allowed.

The default value is IO.

TITLE

The TITLE attribute is type String. It must be a valid <title>.

The attribute TITLE is an external filename and family name. It is used to associate a logical file with a physical or permanent file. The default TITLE for a file is the value of the INTNAME attribute.

Setting the TITLE attribute changes the values of the FILENAME and FAMILYNAME attributes.

USERBACKUPNAME

This attribute is valid for printer files only, and is type Boolean.

Setting the USERBACKUPNAME attribute specifies that the TITLE attribute be used rather than the default BACKUP-FILE-ID (BACKUP/PRT< integer> or BACKUP/PCH<integer>) when entering the filename in the directory.

APPENDIX C CONVERTING CONTROL CARD ATTRIBUTES TO WFL ATTRIBUTES

TASK ATTRIBUTES

CC Abrev	Control Card Program Attribute	WFL Task Attribute
AT	ATTRIBUTES	none
CG	CHARGE	CHARGE
DS	DYNAMIC.SPACES	none
FI	FILE	FILE
FR	FREEZE	none
ID	INTRINSIC.DIRECTORY	none
IN	INTERPRETER	B1000INTERPRETER
IT	INTRINSIC.NAME	none
IV	INVISIBLE	INVISIBLE
LE	LEVEL	none
ME	MEMORY	B1000MEMORY
MP	MEMORY.PRIORITY	none
MS	MEMORY.STATIC	none
	MAXTIME	MAXTIME
MW	MAXWAIT	MAXWAIT
NODIF	NO.DEATH.IN.FAMILY	none
OV	OVERRIDE	none
PP	PROCESSOR.PRIORITY	PRIORITY
PR	PRIORITY	PRIORITY
PT	PROTECTED	PROTECTED
SB	SECONDS.BEFORE.DECAY	none
SC	SCHEDULE.PRIORITY	none
SQN	SYMBOLIC QUEUE NAME	none
SW	SWITCH	SW1 - SW8, DEBUG
TC	TRACE	none
TI	TIME	MAXPROCTIME
UF	UNFREEZE	none
UV	UNOVERRIDE	none
VI	VIRTUAL.DISK	B1000VIRTUALDISK
AF	AFTER	Use synchronous tasks
AN	AFTER.NUMBER	Use synchronous tasks
CA	CONDITIONAL	IF task-id IS COMPLETEDOK
HO	HOLD	Use FETCH for job task
OBJ		Do not specify COMPILER
TH	THEN	Use synchronous tasks
UC	UNCONDITIONAL	WFL default

FILE ATTRIBUTES

CC Abrev	Control Card File Attribute	WFL File Attribute
ALL	ALLOCATE.AT.OPEN	none
ARE	AREAS	AREAS
ASC	ASCII	EXTMODE=ASCII
ATP	AUTOPRINT	PRINTDISPOSITION
BAC	BACKUP	BACKUPPERMITTED
BDK	BACKUP.DISK	BACKUPKIND
BTP	BACKUP.TAPE	BACKUPKIND
BCL	BCL	none
BIN	BINARY	EXTMODE=BINARY
B.A	BLOCKS.AREA	AREABLOCKS
BUF	BUFFERS	BUFFERS
CAS	CASSETTE	KIND=TAPECASSETTE
CPC	CARD.PUNCH	KIND=PUNCH
CPY	COPY	none
CRD	CARD.READER	KIND=READER
DCG	DISK.CARTRIDGE	none
DFL	DISK.FILE	none
DEF	DEFAULT	DEPENDENTSPECS
DPC	DISK.PACK	none
D.R	DELAYED.RANDOM	none
DRC	DATA.RECORDER.80	none
D.F	DUMMY.FILE	DUMMYFILE
DRI	DRIVE	none
DSK	DISK	KIND=DISK
EBC	EBCDIC	EXTMODE=EBCDIC
EMT	EMULATOR.TAPE	none
EOP	EOP	none
EU	EU	none
EVN	EVEN	PARITY=EVEN
EXT	EXTEND	none
FLE	FLEXIBLE	FLEXIBLE
FTP	FILE.TYPE	FILEKIND
FMS	FORMS	none
FOOT	FOOTING	none
HAR	HARDWARE	BACKUPPERMITTED
HDR	HEADER	none
HNM	HOSTNAME	HOSTNAME
IMP	IMPLIED.OPEN	none
INP	INPUT	MYUSE
INV	INVALID.CHARACTERS	none
LAB	LABEL.TYPE	LABEL
LIN	LINEFORMAT	none
LOC	LOCK	Suggest PROTECTION=SAVE
L.M	LOWER.MARGIN	none
MAX	MAXIMUM.BLOCK.SIZE	none
	MAXRECSIZE	MAXRECSIZE
	MINRECSIZE	MINRECSIZE

CC Abrev	Control Card File Attribute	WFL File Attribute
MSF	MAXSUBFILES	MAXSUBFILES
MUL	MULTI.PACK	none
MYN	MYNAME	MYNAME
NAM	NAME	TITLE
NEW	NEW.FILE	NEWFILE
NST	NUMBER.STATIONS	none
ODD	ODD	PARITY=ODD
OLK	OPEN.LOCK	OTHERUSE=IN
OLO	OPEN.LOCKOUT	OTHERUSE=SECURED
OPT	OPTIONAL	OPTIONAL
OUT	OUTPUT	MYUSE
P.S	PAGE.SIZE	PAGESIZE
PID	PACK.ID	FAMILYNAME
PKY	PORT.KEY	none
PORT	PORT.FILE	KIND=PORT
PTL	PROTOCOL	none
PTP	PAPER.TAPE.PUNCH	KIND=PAPERPUNCH
PTR	PAPER.TAPE.READER	KIND=PAPERREADER
	PROTECTION	PROTECTION
PRT	PRINTER	KIND=PRINTER
PSE	PSEUDO	none
QFO	QUEUE.OLD	none
QFS	Q.FAMILY.SIZE	MAXSUBFILES
QMX	Q.MAX.MESSAGES	none
QUE	QUEUE	none
R96	READER.96	none
RAN	RANDOM	none
R.B	RECORDS.BLOCK	BLOCKSIZE
REE	REEL	none
REM	REMOTE	KIND=REMOTE
REP	REPETITIONS	PRINTCOPIES
REV	REVERSE	DIRECTION
REW	REWIND	none
RPP	READER.PUNCH.PRINTER	none
RS2	READER.SORTER.2	none
RSR	READER.SORTER	KIND=READERSORTER
RST	READER.SORTER.STATIONS	none
RSZ	RECORD.SIZE	MAXRECSIZE
S.H	SIMPLE.HEADERS	none
SAA	SENDALL	none
SAV	SAVE	SAVEFACTOR
SEC	SECURITYTYPE	SECURITYTYPE
SEQ	SEQUENTIAL	none
SER	SERIAL	none
SNO	SERIAL.NUMBER	none
STA	STATIONS	none
SUS	SECURITYUSE	SECURITYUSE
TAP	TAPE	KIND=TAPE
TPN	TAPE.NRZ (9 track)	KIND=TAPE9

CC Abrev	Control Card File Attribute	WFL File Attribute
TP7	TAPE.7	KIND=TAPE7
TP9	TAPE.9 (NRZ or PE)	none
TPE	TAPE.PE	KIND=TAPEPE
	TITLE	TITLE
TRN	TRANSLATE	none
TNM	TRANSLATE.NAME	none
UNI	UNIT.NAME	none
UNL	UNLABELED	LABEL
U.M	UPPERMARGIN	none
U.N	USER.BACKUP.NAME	USERBACKUPNAME
VAR	VARIABLE	BLOCKSTRUCTURE
WIN	WITH.INTERPRET	none
WPR	WITH.PRINT	none
WPC	WITH.PUNCH	none
WST	WITH.STACKERS	none
WFL	WORK.FILE	none

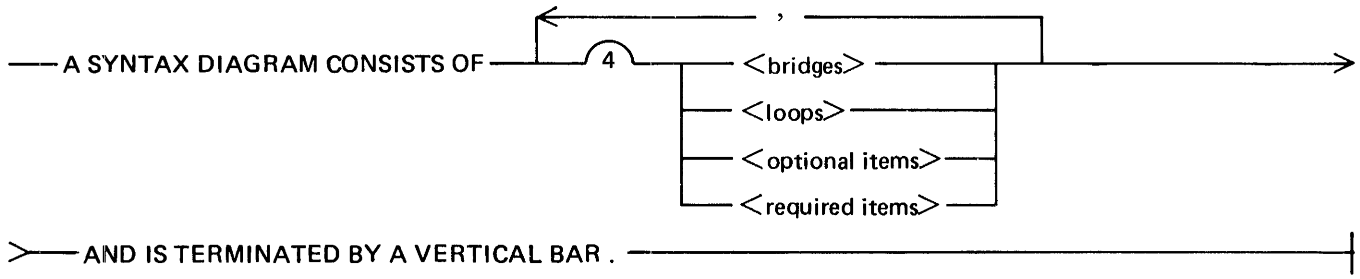
APPENDIX D
SYNTAX AND NOTATION CONVENTIONS

GENERAL

Railroad syntax diagrams show how syntactically valid statements can be constructed. Traversing a railroad syntax diagram from left to right, down on the left vertical line, up on the right vertical line, or in the direction of the arrow heads, and adhering to the limits illustrated by bridges produces a syntactically valid statement. Continuation from one line of the diagram to another is represented by a right arrow appearing at the end of the current line and beginning of the next line. The complete syntax diagram is terminated by a vertical bar (|).

Items contained in broken brackets (< >) are syntactic variables which must be further defined, or for which the user is required to supply the information requested.

Upper-case items must appear literally.



A SYNTAX DIAGRAM CONSISTS OF
,
4
<bridges>
<loops>
<optional items>
<required items>
AND IS TERMINATED BY A VERTICAL BAR .

The following syntactically valid statements can be constructed from the above diagram:

A SYNTAX DIAGRAM CONSISTS OF <bridges> AND IS TERMINATED BY A VERTICAL BAR.

A SYNTAX DIAGRAM CONSISTS OF <optional items> AND IS TERMINATED BY A VERTICAL BAR.

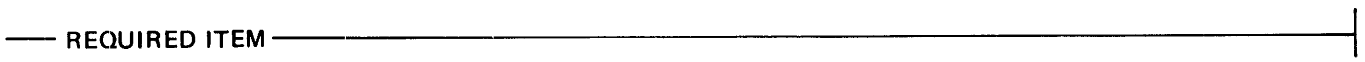
A SYNTAX DIAGRAM CONSISTS OF <bridges>, <loops> AND IS TERMINATED BY A VERTICAL BAR.

A SYNTAX DIAGRAM CONSISTS OF <optional items>, <required items>, <bridges>, <loops> AND IS TERMINATED BY A VERTICAL BAR.

REQUIRED ITEMS

No alternate path through the syntax diagram exists for required items or required punctuation.

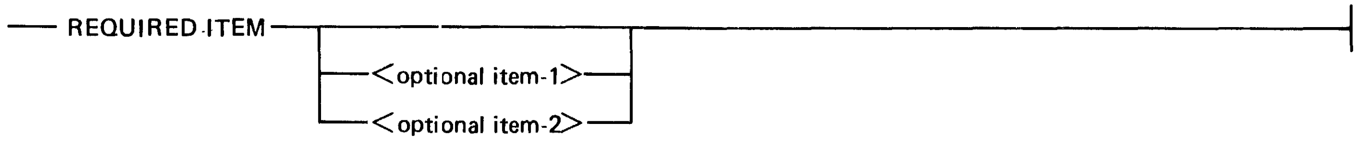
Example:



OPTIONAL ITEMS

A vertical list of items indicates that the user must make a choice of the items specified. An empty path through the list allows the optional item to be absent.

Example:



The following valid statements can be constructed from the above diagram:

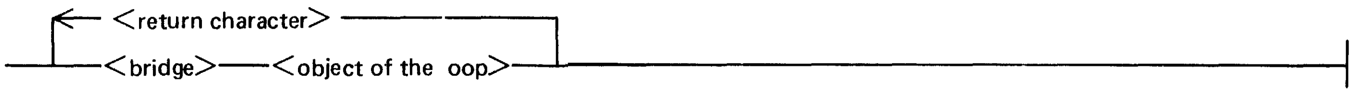
REQUIRED ITEM

REQUIRED ITEM <optional item-1>

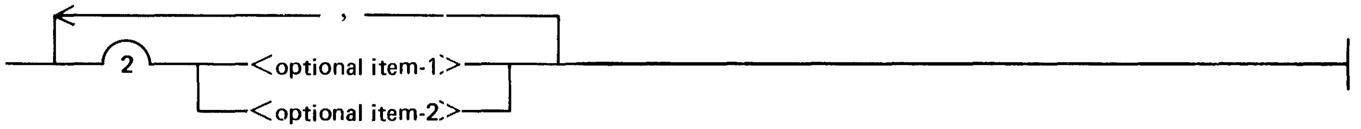
REQUIRED ITEM <optional item-2>

LOOPS

A loop is a recurrent path throɩ.gh a syntax diagram and has the following general format.



Example:



The following statements can be constructed from the syntax diagram in the example.

<optional item-1>

<optional item-1>,<optional item-1>

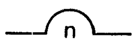
<optional item-2>,<optional item-1>

A loop must be traversed in the direction of the arrow heads, and the limits specified by bridges cannot be exceeded.

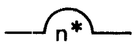
BRIDGES

A bridge illustrates the minimum or maximum number of times a path can be traversed in a syntax diagram.

There are two forms of bridges.

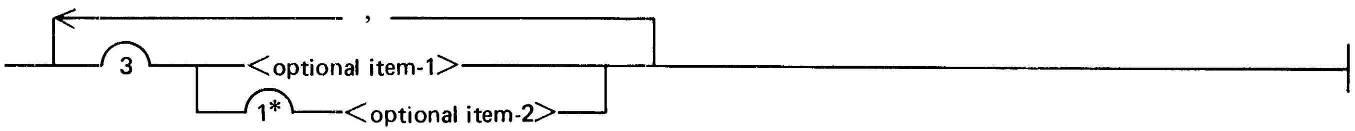


n is an integer which specifies the maximum number of times the path can be traversed.



When n is followed by an asterisk (*), n is an integer which specifies the minimum number of times the path must be traversed.

Example:



The loop can be traversed a maximum of two times; however, the path for <optional item-2> must be traversed at least one time.

The following statements can be constructed from the syntax diagram in the example.

<optional item-1>,<optional item-2>

<optional item-2>,<optional item-2>,<optional item-1>

<optional item-2>

INDEX

Documentation Evaluation Form

Title: B 1000 Systems Work Flow Language (WFL) Language Manual

Form No: 5025265

Date: September 1986

Burroughs Corporation is interested in receiving your comments and suggestions, regarding this manual. Comments will be utilized in ensuing revisions to improve this manual.

Please check type of Suggestion:

☐ Addition ☐ Deletion ☐ Revision ☐ Error

Comments: ______________________________

From:

Name ______________________________

Title ______________________________

Company ______________________________

Address ______________________________

____

Phone Number ______________ Date ______________

Remove form and mail to:

Burroughs Corporation
Documentation Dept., TIO - West
1300 John Reed Court
City of Industry, CA 91745
U.S.A.