Logical Condition Criteria

This chapter describes purpose and use of logical condition criteria that can be used in the statements `FIND`, `READ`, `HISTOGRAM`, `ACCEPT`/`REJECT`, `IF`, `DECIDE FOR`, `REPEAT`.

The following topics are covered:

- Introduction
- Relational Expression
- Extended Relational Expression
- Evaluation of a Logical Variable
- Fields Used within Logical Condition Criteria
- Logical Operators in Complex Logical Expressions
- BREAK Option - Compare Current Value with Value of Previous Loop Pass
- IS Option - Check whether Content of Alphanumeric or Unicode Field can be Converted
- MASK Option - Check Selected Positions of a Field for Specific Content
- MASK Option Compared with IS Option
- MODIFIED Option - Check whether Field Content has been Modified
- SCAN Option - Scan for a Value within a Field
- SPECIFIED Option - Check whether a Value is Passed for an Optional Parameter

---

Introduction

The basic criterion is a relational expression. Multiple relational expressions may be combined with logical operators (`AND`, `OR`) to form complex criteria.

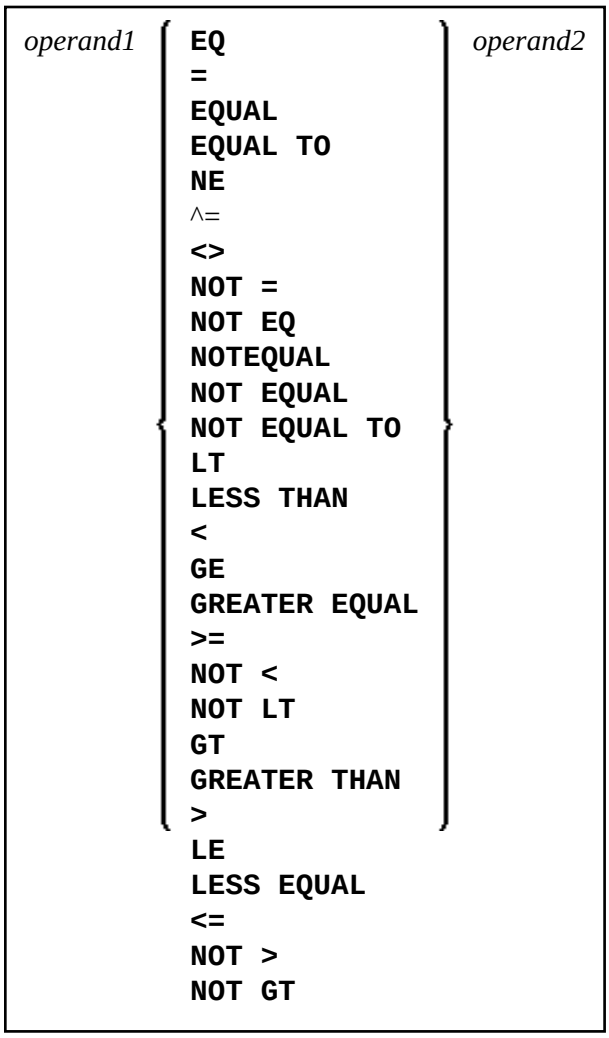
Arithmetic expressions may also be used to form a relational expression.

Logical condition criteria can be used in the following statements:

Statement	Usage
FIND	A WHERE clause containing logical condition criteria may be used to indicate criteria in addition to the basic selection criteria as specified in the WITH clause. The logical condition criteria specified with the WHERE clause are evaluated after the record has been selected and read. In a WITH clause, "basic search criteria" (as described with the FIND statement) are used, but not logical condition criteria.
READ	A WHERE clause containing logical condition criteria may be used to specify whether a record that has just been read is to be processed. The logical condition criteria are evaluated after the record has been read.
HISTOGRAM	A WHERE clause containing logical condition criteria may be used to specify whether the value that has just been read is to be processed. The logical condition criteria are evaluated after the value has been read.
ACCEPT/REJECT	An IF clause may be used with an ACCEPT or REJECT statement to specify logical condition criteria in addition to that specified when the record was selected/read with a FIND, READ, or HISTOGRAM statement. The logical condition criteria are evaluated after the record has been read and after record processing has started.
IF	Logical condition criteria are used to control statement execution.
DECIDE FOR	Logical condition criteria are used to control statement execution.
REPEAT	The UNTIL or WHILE clause of a REPEAT statement contain logical condition criteria which determine when a processing loop is to be terminated.

Relational Expression

Syntax:



Operand Definition Table:

Operand	Possible Structure						Possible Formats														Referencing Permitted	Dynamic Definition
operand1	C	S	A		N	E	A	U	N	P	I	F	B	D	T	L		G	O		yes	yes
operand2	C	S	A		N	E	A	U	N	P	I	F	B	D	T	L		G	O		yes	no

For an explanation of the Operand Definition Table shown above, see *Syntax Symbols and Operand Definition Tables* in the *Statements* documentation.

In the "Possible Structure" column of the table above, "E" stands for arithmetic expressions; that is, any arithmetic expression may be specified as an operand within the relational expression. For further information on arithmetic expressions, see *arithmetic-expression* in the COMPUTE statement description.

Explanation of the comparison operators:

Comparison Operator	Explanation
EQ = EQUAL EQUAL TO	equal to
NE ^= <> NOT = NOT EQ NOTEQUAL NOT EQUAL NOT EQUAL TO	not equal to
LT LESS THAN <	less than
GE GREATER EQUAL >=	greater than or equal to
NOT < NOT LT	not less than
GT GREATER THAN >	greater than
LE LESS EQUAL <=	less than or equal to
NOT > NOT GT	not greater than

Examples of Relational Expressions:

```
IF NAME = 'SMITH'
IF LEAVE-DUE GT 40
IF NAME = #NAME
```

For information on comparing arrays in a relational expression, see *Processing of Arrays*.

Note:
If a floating-point operand is used, comparison is performed in floating point. Floating-point numbers as such have only a limited precision; therefore, rounding/truncation errors cannot be precluded when numbers are converted to/from floating-point representation.

Arithmetic Expressions in Logical Conditions

The following example shows how arithmetic expressions can be used in logical conditions:

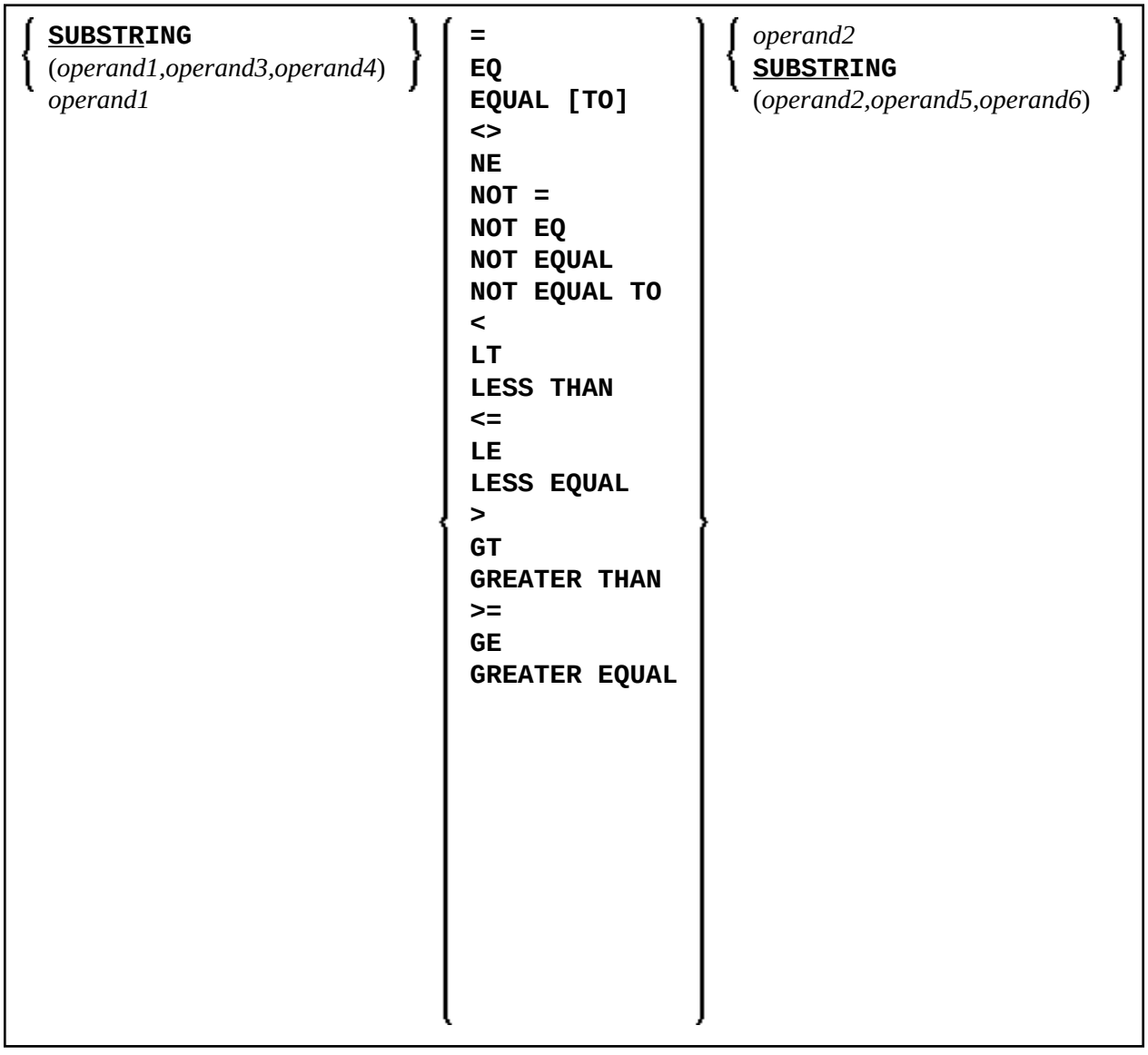
```
IF #A + 3 GT #B - 5 AND #C * 3 LE #A + #B
```

Handles in Logical Conditions

If the operands in a relation expression are handles, only `EQUAL` and `NOT EQUAL` operators may be used.

SUBSTRING Option in Relational Expression

Syntax:



Operand Definition Table:

Operand	Possible Structure						Possible Formats												Referencing Permitted	Dynamic Definition
operand1	C	S	A		N		A	U				B							yes	yes
operand2	C	S	A		N		A	U				B							yes	no
operand3	C	S							N	P	I	B							yes	no
operand4	C	S							N	P	I								yes	no
operand5	C	S							N	P	I								yes	no
operand6	C	S							N	P	I								yes	no

With the SUBSTRING option, you can compare a *part* of an alphanumeric, a binary or a Unicode field. After the field name (*operand1*) you specify first the starting position (*operand3*) and then the length (*operand4*) of the field portion to be compared.

Also, you can compare a field value with part of another field value. After the field name (*operand2*) you specify first the starting position (*operand5*) and then the length (*operand6*) of the field portion *operand1* is to be compared with.

You can also combine both forms, that is, you can specify a SUBSTRING for both *operand1* and *operand2*.

Examples:

The following expression compares the 5th to 12th position inclusive of the value in field #A with the value of field #B:

```
SUBSTRING(#A,5,8) = #B
```

- where 5 is the starting position and 8 is the length.

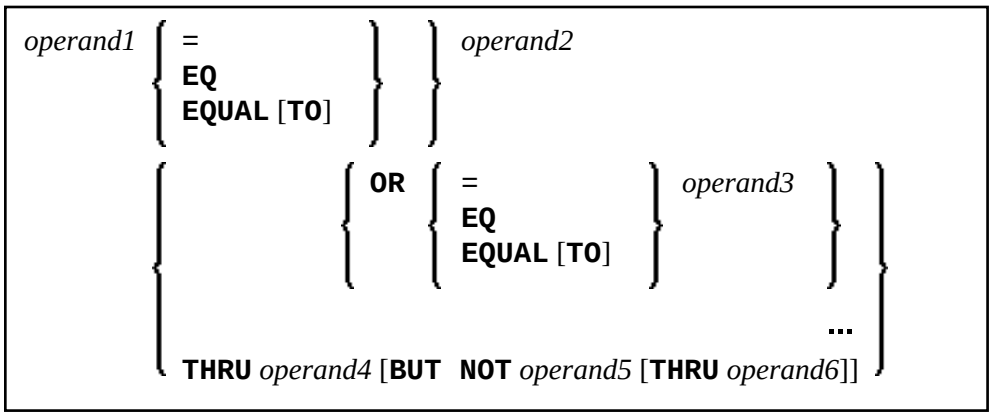
The following expression compares the value of field #A with the 3rd to 6th position inclusive of the value in field #B:

```
#A = SUBSTRING(#B,3,4)
```

Note:
If you omit *operand3/operand5*, the starting position is assumed to be 1. If you omit *operand4/operand6*, the length is assumed to be from the starting position to the end of the field.

Extended Relational Expression

Syntax:



Operand Definition Table:

Operand	Possible Structure						Possible Formats												Referencing Permitted	Dynamic Definition
operand1	C	S	A		N*	E	A	U	N	P	I	F	B	D	T			G	O	yes	no
operand2	C	S	A		N*	E	A	U	N	P	I	F	B	D	T			G	O	yes	no
operand3	C	S	A		N*	E	A	U	N	P	I	F	B	D	T			G	O	yes	no
operand4	C	S	A		N*	E	A	U	N	P	I	F	B	D	T			G	O	yes	no
operand5	C	S	A		N*	E	A	U	N	P	I	F	B	D	T			G	O	yes	no
operand6	C	S	A		N*	E	A	U	N	P	I	F	B	D	T			G	O	yes	no

* Mathematical functions and system variables are permitted. Break functions are not permitted.

operand3 can also be specified using a MASK or SCAN option; that is, it can be specified as:

MASK (*mask-definition*) [*operand*]
MASK *operand*
SCAN *operand*

For details on these options, see the sections *MASK Option* and *SCAN Option*.

Examples:

```
IF #A = 2 OR = 4 OR = 7
IF #A = 5 THRU 11 BUT NOT 7 THRU 8
```

Evaluation of a Logical Variable

Syntax:

operand1

This option is used in conjunction with a logical variable (format L). A logical variable may take the value TRUE or FALSE. As *operand1* you specify the name of the logical variable to be used.

Operand Definition Table:

Operand	Possible Structure			Possible Formats													Referencing Permitted	Dynamic Definition
operand1	C	S	A												L				no	no

Example of Logical Variable:

```
** Example 'LOGICX05': Logical variable in logical condition
************************************************************************
DEFINE DATA LOCAL
1 #SWITCH (L)  INIT <true>
1 #INDEX  (I1)
END-DEFINE
*
FOR #INDEX 1 5
  WRITE NOTITLE #SWITCH (EM=FALSE/TRUE) 5X 'INDEX =' #INDEX
  WRITE NOTITLE #SWITCH (EM=OFF/ON) 7X 'INDEX =' #INDEX
  IF #SWITCH
    MOVE FALSE TO #SWITCH
  ELSE
    MOVE TRUE TO #SWITCH
  END-IF
  /*
  SKIP 1
END-FOR
END
```

Output of Program LOGICX05:

```
TRUE     INDEX =    1
ON       INDEX =    1

FALSE    INDEX =    2
OFF      INDEX =    2

TRUE     INDEX =    3
ON       INDEX =    3

FALSE    INDEX =    4
OFF      INDEX =    4

TRUE     INDEX =    5
ON       INDEX =    5
```

Fields Used within Logical Condition Criteria

Database fields and user-defined variables may be used to construct logical condition criteria. A database field which is a multiple-value field or is contained in a periodic group can also be used. If a range of values for a multiple-value field or a range of occurrences for a periodic group is specified, the condition is true if the search value is found in any value/occurrence within the specified range.

Each value used must be compatible with the field used on the opposite side of the expression. Decimal notation may be specified only for values used with numeric fields, and the number of decimal positions of the value must agree with the number of decimal positions defined for the field.

If the operands are not of the same format, the second operand is converted to the format of the first operand.

Note:
A numeric constant without decimal point notation is stored with format I for the values `-2147483648` to `+2147483647`, see *Numeric Constants*. Consequently the comparison with such an integer constant as *`operand1`* is performed by converting *`operand2`* to a integer value. This means that the digits after the decimal point of *`operand2`* are not considered due to truncation.

Example:

```
IF 0 = 0.5      /* is true because 0.5 (operand2) is converted to 0 (format I of operand1)
IF 0.0 = 0.5  /* is false
IF 0.5 = 0      /* is false
IF 0.5 = 0.0  /* is false
```

The following table shows which operand formats can be used together in a logical condition:

operand1	***operand2***												
	A	**U**	**B*n* (*n*=<4)**	**B*n* (*n*>=5)**	**D**	**T**	**I**	**F**	**L**	**N**	**P**	**GH**	**OH**
A	Y	Y	Y	Y									
U	Y	Y	[2]	[2]									
B*n* (*n*=<4)	Y	Y	Y	Y	Y	Y	Y	Y		Y	Y		
B*n* (*n*>=5)	Y	Y	Y	Y									
D			Y		Y	Y	Y	Y		Y	Y		
T			Y		Y	Y	Y	Y		Y	Y		
I			Y		Y	Y	Y	Y		Y	Y		
F			Y		Y	Y	Y	Y		Y	Y		
L													
N			Y		Y	Y	Y	Y		Y	Y		
P			Y		Y	Y	Y	Y		Y	Y		
GH [1]												Y	
OH [1]													Y

Notes:

1. [1] where GH = GUI handle, OH = object handle.
2. [2] The binary value will be assumed to contain Unicode code points, and the comparison is performed as for a comparison of two Unicode values. The length of the binary field must be even.

If two values are compared as alphanumeric values, the shorter value is assumed to be extended with trailing blanks in order to get the same length as the longer value.

If two values are compared as binary values, the shorter value is assumed to be extended with leading binary zeroes in order to get the same length as the longer value.

If two values are compared as Unicode values, trailing blanks are removed from both values before the ICU collation algorithm is used to compare the two resulting values. See also *Logical Condition Criteria* in the *Unicode and Code Page Support* documentation.

Comparison Examples:

```
A1(A1) := 'A'
A5(A5) := 'A    '
B1(B1) := H'FF'
B5(B5) := H'00000000FF'
U1(U1) := UH'00E4'
U2(U2) := UH'00610308'
IF A1 = A5 THEN ...                /* TRUE
IF B1 = B5 THEN ...                /* TRUE
IF U1 = U2 THEN ...                /* TRUE
```

If an array is compared with a scalar value, each element of the array will be compared with the scalar value. The condition will be true if at least one of the array elements meets the condition (OR operation).

If an array is compared with an array, each element in the array is compared with the corresponding element of the other array. The result is true only if all element comparisons meet the condition (AND operation).

See also *Processing of Arrays*.

Note:
An Adabas phonetic descriptor cannot be used within a logical condition.

Examples of Logical Condition Criteria:

```
FIND EMPLOYEES-VIEW WITH CITY = 'BOSTON' WHERE SEX = 'M'
READ EMPLOYEES-VIEW BY NAME WHERE SEX = 'M'
ACCEPT IF LEAVE-DUE GT 45
IF #A GT #B THEN COMPUTE #C = #A + #B
REPEAT UNTIL #X = 500
```

Logical Operators in Complex Logical Expressions

Logical condition criteria may be combined using the Boolean operators AND, OR, and NOT. Parentheses may also be used to indicate logical grouping.

The operators are evaluated in the following order:

Priority	Operator	Meaning
1	( )	Parentheses
2	NOT	Negation
3	AND	AND operation
4	OR	OR operation

The following *logical-condition-criteria* may be combined by logical operators to form a complex *logical-expression*:

- Relational expressions
- Extended relational expressions
- MASK option
- SCAN option
- BREAK option

The syntax for a *logical-expression* is as follows:

```
[NOT] { logical-condition-criterion } [ { OR  } logical-expression ] ...
      { (logical-expression)        }   { AND }
```

Examples of Logical Expressions:

```
FIND STAFF-VIEW WITH CITY = 'TOKYO'
     WHERE BIRTH GT 19610101 AND SEX = 'F'
IF NOT (#CITY = 'A' THRU 'E')
```

For information on comparing arrays in a logical expression, see *Processing of Arrays*.

Note:
If multiple logical-condition-criteria are connected with AND, the evaluation terminates as soon as the first of these criteria is not true.

BREAK Option - Compare Current Value with Value of Previous Loop Pass

The BREAK option allows the current value or a portion of a value of a field to be compared with the value contained in the same field in the previous pass through the processing loop.

Syntax:

```
BREAK [OF] operand1 [/n/]
```

Operand Definition Table:

Operand	Possible Structure						Possible Formats													Referencing Permitted	Dynamic Definition
operand1		S					A	U	N	P	I	F	B	D	T	L				yes	no

Syntax Element Description:

operand1	Specifies the control field which is to be checked. A specific occurrence of an array can also be used as a control field.
/n/	The notation */n/* may be used to indicate that only the first *n* positions (counting from left to right) of the control field are to be checked for a change in value. This notation can only be used with operands of format A, B, N, or P. The result of the BREAK operation is true when a change in the specified positions of the field occurs. The result of the BREAK operation is not true if an AT END OF DATA condition occurs. Example: In this example, a check is made for a different value in the first position of the field FIRST-NAME. `BREAK FIRST-NAME /1/` Natural system functions (which are available with the AT BREAK statement) are not available with this option.

Example of BREAK Option:

```
** Example 'LOGICX03': BREAK option in logical condition
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 NAME
  2 FIRST-NAME
  2 BIRTH
*
1 #BIRTH (A8)
END-DEFINE
*
LIMIT 10
READ EMPLOY-VIEW BY BIRTH
  MOVE EDITED BIRTH (EM=YYYYMMDD) TO #BIRTH
  /*
  IF BREAK OF #BIRTH /6/
    NEWPAGE IF LESS THAN 5 LINES LEFT
    WRITE / '-' (50) /
  END-IF
  /*
  DISPLAY NOTITLE BIRTH (EM=YYYY-MM-DD) NAME FIRST-NAME
END-READ
END
```

Output of Program LOGICX03:

```
  DATE            NAME              FIRST-NAME
   OF
 BIRTH
---------- -------------------- --------------------

1940-01-01 GARRET               WILLIAM
1940-01-09 TAILOR               ROBERT
1940-01-09 PIETSCH              VENUS
1940-01-31 LYTTLETON            BETTY

------------------------------------------------

1940-02-02 WINTRICH             MARIA
1940-02-13 KUNEY                MARY
1940-02-14 KOLENCE              MARSHA
1940-02-24 DILWORTH             TOM

------------------------------------------------

1940-03-03 DEKKER               SYLVIA
1940-03-06 STEFFERUD            BILL
```

IS Option - Check whether Content of Alphanumeric or Unicode Field can be Converted

Syntax:

operand1 IS (*format*)

This option is used to check whether the content of an alphanumeric or Unicode field (*operand1*) can be converted to a specific other format.

Operand Definition Table:

Operand	Possible Structure						Possible Formats												Referencing Permitted	Dynamic Definition
operand1	C	S	A		N		A	U											yes	no

The *format* for which the check is performed can be:

N*ll.ll*	Numeric with length *ll.ll*.
F*ll*	Floating point with length *ll*.
D	Date. The following date formats are possible: *dd-mm-yy*, *dd-mm-yyyy*, *ddmmyyyy* (*dd* = day, *mm* = month, *yy* or *yyyy* = year). The sequence of the day, month and year components as well as the characters between the components are determined by the profile parameter DTFORM (which is described in the *Parameter Reference*).
T	Time (according to the default time display format).
P*ll.ll*	Packed numeric with length *ll.ll*.
I*ll*	Integer with length *ll*.

When the check is performed, leading and trailing blanks in *operand1* will be ignored.

The IS option may, for example, be used to check the content of a field before the mathematical function VAL (extract numeric value from an alphanumeric field) is used to ensure that it will not result in a runtime error.

Note:
The IS option cannot be used to check if the value of an alphanumeric field is in the specified "format", but if it can be *converted* to that "format". To check if a value is in a specific format, you can use the MASK option. For further information, see *MASK Option Compared with IS Option* and *Checking Packed or Unpacked Numeric Data*.

Example of IS Option:

```
** Example 'LOGICX04': IS option as format/length check
************************************************************************
DEFINE DATA LOCAL
1 #FIELDA (A10)          /* INPUT FIELD TO BE CHECKED
1 #FIELDB (N5)           /* RECEIVING FIELD OF VAL FUNCTION
1 #DATE   (A10)          /* INPUT FIELD FOR DATE
END-DEFINE
*
INPUT #DATE #FIELDA
IF #DATE IS(D)
  IF #FIELDA IS (N5)
    COMPUTE #FIELDB = VAL(#FIELDA)
    WRITE NOTITLE 'VAL FUNCTION OK' //  '=' #FIELDA '=' #FIELDB
  ELSE
    REINPUT 'FIELD DOES NOT FIT INTO N5 FORMAT'
            MARK *#FIELDA
  END-IF
ELSE
  REINPUT 'INPUT IS NOT IN DATE FORMAT (YY-MM-DD) '
          MARK *#DATE
END-IF
*
END
```

Output of Program LOGICX04:

```
#DATE 150487    #FIELDA


INPUT IS NOT IN DATE FORMAT (YY-MM-DD)
```

MASK Option - Check Selected Positions of a Field for Specific Content

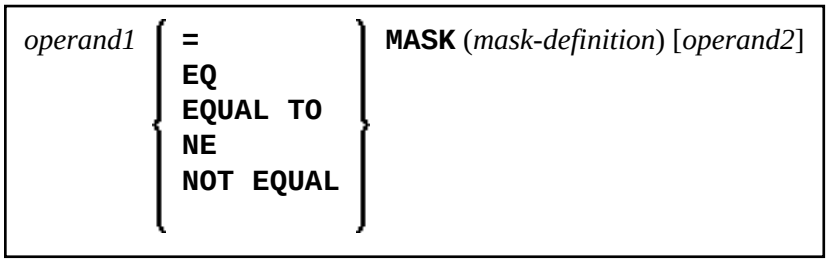
With the MASK option, you can check selected positions of a field for specific content.

The following topics are covered below:

- Constant Mask
- Variable Mask
- Characters in a Mask
- Mask Length
- Checking Dates
- Checking Against the Content of Constants or Variables
- Range Checks
- Checking Packed or Unpacked Numeric Data

Constant Mask

Syntax:



Operand Definition Table:

Operand	Possible Structure						Possible Formats												Referencing Permitted	Dynamic Definition
operand1	C	S	A		N		A	U	N	P									yes	no
operand2	C	S					A	U	N	P			B						yes	no

operand2 can only be used if the *mask-definition* contains at least one X. *operand1* and *operand2* must be format-compatible:

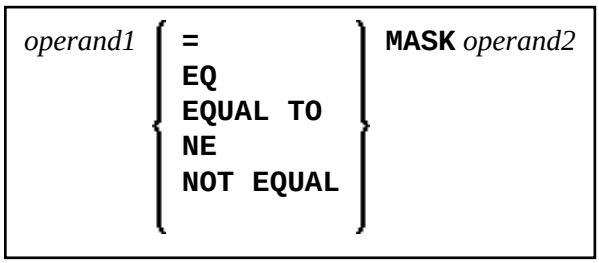
- If *operand1* is of format A, *operand2* must be of format A, B, N or U.
- If *operand1* is of format U, *operand2* must be of format A, B, N or U.
- If *operand1* is of format N or P, *operand2* must be of format N or P.

An X in the *mask-definition* selects the corresponding positions of the content of *operand1* and *operand2* for comparison.

Variable Mask

Apart from a constant *mask-definition* (see above), you may also specify a variable mask definition.

Syntax:



Operand Definition Table:

Operand	Possible Structure						Possible Formats												Referencing Permitted	Dynamic Definition
operand1	C	S	A		N		A	U	N	P									yes	no
operand2		S					A	U											yes	no

The content of *operand2* will be taken as the mask definition. Trailing blanks in *operand2* will be ignored.

- If *operand1* is of format A, N or P, *operand2* must be of format A.
- If *operand1* is of format U, *operand2* must be of format U.

Characters in a Mask

The following characters may be used within a mask definition (the mask definition is contained in *mask-definition* for a constant mask and *operand2* for a variable mask):

Character	Meaning
. or ? or _	A period, question mark or underscore indicates a single position that is not to be checked.
* or %	An asterisk or percent mark is used to indicate any number of positions not to be checked.
/	A slash (/) is used to check if a value ends with a specific character (or string of characters). For example, the following condition will be true if there is either an E in the last position of the field, or the last E in the field is followed by nothing but blanks: `IF #FIELD = MASK (*'E'/)`
A	The position is to be checked for an alphabetical character (upper or lower case).
'c'	One or more positions are to be checked for the characters bounded by apostrophes (a double apostrophe indicates that a single apostrophe is the character to be checked for). If *operand1* is in Unicode format, *'c'* must contain Unicode characters.
C	The position is to be checked for an alphabetical character (upper or lower case), a numeric character, or a blank.
DD	The two positions are to be checked for a valid day notation (01 - 31; dependent on the values of MM and YY/YYYY, if specified; see also *Checking Dates*).
H	The position is to be checked for hexadecimal content (A - F, 0 - 9).
JJJ	The positions are to be checked for a valid Julian Day; that is, the day number in the year (001-366, dependent on the value of YY/YYYY, if specified. See also *Checking Dates*.)
L	The position is to be checked for a lower-case alphabetical character (a - z).
MM	The positions are to be checked for a valid month (01 - 12); see also *Checking Dates*.
N	The position is to be checked for a numeric digit.
n...	One (or more) positions are to be checked for a numeric value in the range 0 - *n*.
*n*1-*n*2 or *n*1:*n*2	The positions are checked for a numeric value in the range *n*1-*n*2. *n*1 and *n*2 must be of the same length.
P	The position is to be checked for a displayable character (U, L, N or S).
S	The position is to be checked for special characters. See also *Support of Different Character Sets with NATCONV.INI* in the *Operations* documentation.
U	The position is to be checked for an upper-case alphabetical character (A - Z).

Character	Meaning
X	The position is to be checked against the equivalent position in the value (*operand2*) following the mask-definition. X is not allowed in a variable mask definition, as it makes no sense.
YY	The two positions are to be checked for a valid year (00 - 99). See also *Checking Dates*.
YYYY	The four positions are checked for a valid year (0000 - 2699).
Z	The position is to be checked for a character whose left half-byte is hexadecimally 3 or 7, and whose right half-byte is hexadecimally 0 - 9. This may be used to correctly check for numeric digits in negative numbers. With N (which indicates a position to be checked for a numeric digit), a check for numeric digits in negative numbers leads to incorrect results, because the sign of the number is stored in the last digit of the number, causing that digit to be hexadecimally represented as non-numeric. Within a mask, use only one Z for each sequence of numeric digits that is checked.

Mask Length

The length of the mask determines how many positions are to be checked.

Example:

```
DEFINE DATA LOCAL
1 #CODE (A15)
END-DEFINE
...
IF #CODE = MASK (NN'ABC'....NN)
...
```

In the above example, the first two positions of #CODE are to be checked for numeric content. The three following positions are checked for the contents ABC. The next four positions are not to be checked. Positions ten and eleven are to be checked for numeric content. Positions twelve to fifteen are not to be checked.

Checking Dates

Only one date may be checked within a given mask. When the same date component (JJJ, DD, MM, YY or YYYY) is specified more than once in the mask, only the value of the last occurrence is checked for consistency with other date components.

When dates are checked for a day (DD) and no month (MM) is specified in the mask, the current month will be assumed.

When dates are checked for a day (DD) or a Julian day (JJJ) and no year (YY or YYYY) is specified in the mask, the current year will be assumed.

When dates are checked for a 2-digit year (YY), the current century will be assumed if no Sliding or Fixed Window is set. For more details about Sliding or Fixed Windows, refer to profile parameter YSLW in the *Parameter Reference*.

Example 1:

```
MOVE 1131 TO #DATE (N4)
IF #DATE = MASK (MMDD)
```

In this example, month and day are checked for validity. The value for month (11) will be considered valid, whereas the value for day (31) will be invalid since the 11th month has only 30 days.

Example 2:

```
IF #DATE(A8) = MASK (MM'/'DD'/'YY)
```

In this example, the content of the field #DATE is be checked for a valid date with the format MM/DD/YY (month/day/year).

Example 3:

```
IF #DATE (A8) = MASK (1950-2020MMDD)
```

In this example, the content of field #DATE is checked for a four-digit number in the range 1950 to 2020 followed by a valid month and day in the current year.

Note:
Although apparent, the above mask does not allow to check for a valid date in the years 1950 through 2020, because the numeric value range 1950-2020 is checked independent of the validation of month and day. The check will deliver the intended results except for February, 29, where the result depends on whether the current year is a leap year or not. To check for a specific year range in addition to the date validation, code one check for the date validation and another for the range validation:

```
IF #DATE (A8) = MASK (YYYYMMDD) AND #DATE = MASK (1950-2020)
```

Example 4:

```
IF #DATE (A4) = MASK (19-20YY)
```

In this example, the content of field #DATE is checked for a two-digit number in the range 19 to 20 followed by a valid two-digit year (00 through 99). The century is supplied by Natural as described above.

Note:
Although apparent, the above mask does not allow to check for a valid year in the range 1900 through 2099, because the numeric value range 19-20 is checked independent of the year validation. To check for year ranges, code one check for the date validation and another for the range validation:

```
IF #DATE (A10) = MASK (YYYY'-'MM'-'DD) AND #DATE = MASK (19-20)
```

Checking Against the Content of Constants or Variables

If the value for the mask check is to be taken from either a constant or a variable, this value (*operand2*) must be specified immediately following the *mask-definition*.

operand2 must be at least as long as the mask.

In the mask, you indicate each position to be checked with an X, and each position not to be checked with a period (.) or a question mark (?) or an underscore (_).

Example:

```
DEFINE DATA LOCAL
1 #NAME (A15)
END-DEFINE
...
IF #NAME = MASK (..XX) 'ABCD'
...
```

In the above example, it is checked whether the field #NAME contains CD in the third and fourth positions. Positions one and two are not checked.

The length of the mask determines how many positions are to be checked. The mask is left-justified against any field or constant used in the mask operation. The format of the field (or constant) on the right side of the expression must be the same as the format of the field on the left side of the expression.

If the field to be checked (*operand1*) is of format A, any constant used (*operand2*) must be enclosed in apostrophes. If the field is numeric, the value used must be a numeric constant or the content of a numeric database field or user-defined variable.

In either case, any characters/digits within the value specified whose positions do not match the X indicator within the mask are ignored.

The result of the MASK operation is true when the indicated positions in both values are identical.

Example:

```
** Example 'LOGICX01': MASK option in logical condition
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 CITY
END-DEFINE
*
HISTOGRAM EMPLOY-VIEW CITY
  IF CITY =
MASK (....XX) '....NN'


    DISPLAY NOTITLE CITY *NUMBER
  END-IF
END-HISTOGRAM
*
END
```

In the above example, the record will be accepted if the fifth and sixth positions of the field CITY each contain the character N.

Range Checks

When performing range checks, the number of positions verified in the supplied variable is defined by the precision of the value supplied in the mask specification. For example, a mask of `(...193...)` will verify positions 4 to 6 for a three-digit number in the range 000 to 193.

Additional Examples of Mask Definitions:

- In this example, each character of #NAME is checked for an alphabetical character:

```
IF #NAME (A10) = MASK (AAAAAAAAAA)
```

- In this example, positions 4 to 6 of #NUMBER are checked for a numeric value:

```
IF #NUMBER (A6) = MASK (...NNN)
```

- In this example, positions 4 to 6 of #VALUE are to be checked for the value 123:

```
IF #VALUE(A10) = MASK (...'123')
```

- This example will check if #LICENSE contains a license number which begins with NY- and whose last five characters are identical to the last five positions of #VALUE:

```
DEFINE DATA LOCAL
1 #VALUE(A8)
1 #LICENSE(A8)
END-DEFINE
INPUT 'ENTER KNOWN POSITIONS OF LICENSE PLATE:' #VALUE
IF #LICENSE = MASK ('NY-'XXXXX) #VALUE
```

- The following condition would be met by any value which contains NAT and AL no matter which and how many other characters are between NAT and AL (this would include the values NATURAL and NATIONALITY as well as NATAL):

```
MASK('NAT'*'AL')
```

Checking Packed or Unpacked Numeric Data

Legacy applications often have packed or unpacked numeric variables redefined with alphanumeric or binary fields. Such redefinitions are not recommended, because using the packed or unpacked variable in an assignment or computation may lead to errors or unpredictable results. To validate the contents of such a redefined variable before the variable is used, use the N option (see *Characters in a Mask*) as many as number of digits - 1 times followed by a single Z option.

Examples :

```
IF #P1 (P1) = MASK (Z)
IF #N4 (N4) = MASK (NNNZ)
IF #P5 (P5) = MASK (NNNNZ)
```

For further information about checking field contents, see *MASK Option Compared with IS Option*.

MASK Option Compared with IS Option

This section points out the difference between MASK option and IS option and contains a sample program to illustrate the difference.

The IS option can be used to check whether the content of an alphanumeric or Unicode field can be converted to a specific other format, but it cannot be used to check if the value of an alphanumeric field is in the specified format.

The MASK option can be used to validate the contents of a redefined packed or unpacked numeric variable.

Example Illustrating the Difference:

```
** Example 'LOGICX09': MASK versus IS option in logical condition
************************************************************************
DEFINE DATA LOCAL
1 #A2   (A2)
1 REDEFINE #A2
  2 #N2 (N2)
1 REDEFINE #A2
  2 #P3 (P3)
1 #CONV-N2 (N2)
1 #CONV-P3 (P3)
END-DEFINE
*
#A2 := '12'
WRITE NOTITLE   'Assignment #A2 := "12" results in:'
PERFORM SUBTEST
#A2 := '-1'
WRITE NOTITLE / 'Assignment #A2 := "-1" results in:'
PERFORM SUBTEST
#N2 :=  12
WRITE NOTITLE / 'Assignment #N2 :=  12  results in:'
PERFORM SUBTEST
#N2 :=  -1
WRITE NOTITLE / 'Assignment #N2 :=  -1  results in:'
PERFORM SUBTEST
#P3 :=  12
WRITE NOTITLE / 'Assignment #P3 :=  12  results in:'
PERFORM SUBTEST
#P3 :=  -1
WRITE NOTITLE / 'Assignment #P3 :=  -1  results in:'
PERFORM SUBTEST
*
DEFINE SUBROUTINE SUBTEST
IF #A2 IS (N2) THEN
  #CONV-N2 := VAL(#A2)
  WRITE NOTITLE 12T '#A2 can be converted to' #CONV-N2 '(N2)'
END-IF
IF #A2 IS (P3) THEN
  #CONV-P3 := VAL(#A2)
  WRITE NOTITLE 12T '#A2 can be converted to' #CONV-P3 '(P3)'
END-IF
IF #N2 = MASK(NZ) THEN
  WRITE NOTITLE 12T '#N2 contains the valid unpacked number' #N2
END-IF
IF #P3 = MASK(NNZ) THEN
  WRITE NOTITLE 12T '#P3 contains the valid packed number' #P3
```

```
END-IF
END-SUBROUTINE
*
END
```

Output of Program LOGICX09:

```
Assignment #A2 := ’12’ results in:
           #A2 can be converted to  12 (N2)
           #A2 can be converted to   12 (P3)
           #N2 contains the valid unpacked number  12

Assignment #A2 := ’-1’ results in:
           #A2 can be converted to  -1 (N2)
           #A2 can be converted to   -1 (P3)

Assignment #N2 :=  12  results in:
           #A2 can be converted to  12 (N2)
           #A2 can be converted to   12 (P3)
           #N2 contains the valid unpacked number  12

Assignment #N2 :=  -1  results in:
           #N2 contains the valid unpacked number  -1

Assignment #P3 :=  12  results in:
           #P3 contains the valid packed number   12

Assignment #P3 :=  -1  results in:
           #P3 contains the valid packed number   -1
```

MODIFIED Option - Check whether Field Content has been Modified

Syntax:

operand1 [**NOT**] **MODIFIED**

This option is used to determine whether the content of a field has been modified during the execution of an INPUT or PROCESS PAGE statement. As a precondition, a control variable must have been assigned using the parameter CV.

Operand Definition Table:

Operand	Possible Structure						Possible Formats												Referencing Permitted	Dynamic Definition
operand1		S	A													C			no	no

Attribute control variables referenced in an INPUT or PROCESS PAGE statement are always assigned the status "not modified" when the map is transmitted to the terminal.

Whenever the content of a field referencing an attribute control variable is modified, the attribute control variable has been assigned the status "modified". When multiple fields reference the same attribute control variable, the variable is marked "modified" if any of these fields is modified.

If *operand1* is an array, the result will be true if at least one of the array elements has been assigned the status "modified" (OR operation).

Example of MODIFIED Option:

```
** Example 'LOGICX06': MODIFIED option in logical condition
************************************************************************
DEFINE DATA LOCAL
1 #ATTR (C)
1 #A    (A1)
1 #B    (A1)
END-DEFINE
*
MOVE (AD=I) TO #ATTR
*
INPUT (CV=#ATTR) #A #B
IF #ATTR NOT MODIFIED
  WRITE NOTITLE 'FIELD #A OR #B HAS NOT BEEN MODIFIED'
END-IF
*
IF #ATTR MODIFIED
  WRITE NOTITLE 'FIELD #A OR #B HAS BEEN MODIFIED'
END-IF
*
END
```

Output of Program LOGICX06:

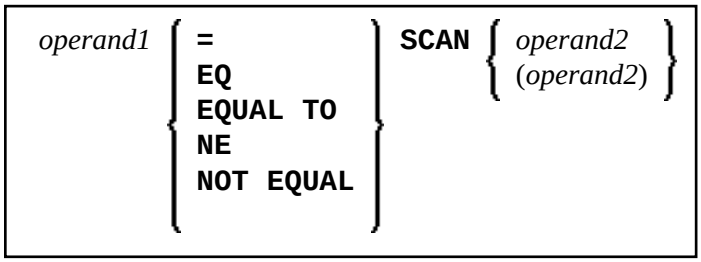
```
#A   #B
```

After entering any value and pressing ENTER, the following output is displayed:

```
FIELD #A OR #B HAS BEEN MODIFIED
```

SCAN Option - Scan for a Value within a Field

Syntax:



Operand Definition Table:

Operand	Possible Structure						Possible Formats												Referencing Permitted	Dynamic Definition
operand1	C	S	A		N		A	U	N	P									yes	no
operand2	C	S					A	U					B*						yes	no

* *operand2* may only be binary if *operand1* is of format A or U. If *operand1* is of format U and *operand2* is of format B, then the length of *operand2* must be even.

The SCAN option is used to scan for a specific value within a field.

The characters used in the SCAN option (*operand2*) may be specified as an alphanumeric or Unicode constant (a character string bounded by apostrophes) or the contents of an alphanumeric or Unicode database field or user-defined variable.

Caution:
Trailing blanks are automatically eliminated from *operand1* and *operand2*. Therefore, the SCAN option cannot be used to scan for values containing trailing blanks. *operand1* and *operand2* may contain leading or embedded blanks. If *operand2* consists of blanks only, scanning will be assumed to be successful, regardless of the value of *operand1*; confer EXAMINE FULL statement if trailing blanks are not to be ignored in the scan operation.

The field to be scanned (*operand1*) may be of format A, N, P or U. The SCAN operation may be specified with the equal (EQ) or not equal (NE) operators.

The length of the character string for the SCAN operation should be less than the length of the field to be scanned. If the length of the character string specified is identical to the length of the field to be scanned, then an EQUAL operator should be used instead of SCAN.

Example of SCAN Option:

```
** Example 'LOGICX02': SCAN option in logical condition
************************************************************************
DEFINE DATA
LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 NAME
*
1 #VALUE   (A4)
1 #COMMENT (A10)
END-DEFINE
*
INPUT 'ENTER SCAN VALUE:' #VALUE
LIMIT 15
*
HISTOGRAM EMPLOY-VIEW FOR NAME
  RESET #COMMENT
  IF NAME = SCAN #VALUE
    MOVE 'MATCH' TO #COMMENT
  END-IF
  DISPLAY NOTITLE NAME *NUMBER #COMMENT
END-HISTOGRAM
*
END
```

Output of Program LOGICX02:

```
ENTER SCAN VALUE:
```

A scan for example for LL delivers three matches in 15 names:

```
        NAME           NMBR      #COMMENT
-------------------- --------- ----------

ABELLAN                      1 MATCH
ACHIESON                     1
ADAM                         1
ADKINSON                     8
AECKERLE                     1
AFANASSIEV                   2
AHL                          1
AKROYD                       1
ALEMAN                       1
ALESTIA                      1
ALEXANDER                    5
ALLEGRE                      1 MATCH
ALLSOP                       1 MATCH
ALTINOK                      1
ALVAREZ                      1
```

SPECIFIED Option - Check whether a Value is Passed for an Optional Parameter

Syntax:

parameter-name [**NOT**] **SPECIFIED**

This option is used to check whether an optional parameter in an invoked object (subprogram, external subroutine or ActiveX control) has received a value from the invoking object or not.

An optional parameter is a field defined with the keyword `OPTIONAL` in the `DEFINE DATA PARAMETER` statement of the invoked object. If a field is defined as `OPTIONAL`, a value can - but need not - be passed from an invoking object to this field.

In the invoking statement, the notation *n*`X` is used to indicate parameters for which no values are passed.

If you process an optional parameter which has not received a value, this will cause a runtime error. To avoid such an error, you use the `SPECIFIED` option in the invoked object to check whether an optional parameter has received a value or not, and then only process it if it has.

`parameter-name` is the name of the parameter as specified in the `DEFINE DATA PARAMETER` statement of the invoked object.

For a field not defined as `OPTIONAL`, the `SPECIFIED` condition is always `TRUE`.

Example of SPECIFIED Option:

Calling Programming:

```
** Example 'LOGICX07': SPECIFIED option in logical condition
************************************************************************
DEFINE DATA LOCAL
1 #PARM1 (A3)
1 #PARM3 (N2)
END-DEFINE
*
```

```
#PARM1 := 'ABC'
#PARM3 := 20
*
CALLNAT 'LOGICX08' #PARM1 1X #PARM3
*
END
```

Subprogram Called:

```
** Example 'LOGICX08': SPECIFIED option in logical condition
************************************************************************
DEFINE DATA PARAMETER
1 #PARM1 (A3)
1 #PARM2 (N2) OPTIONAL
1 #PARM3 (N2) OPTIONAL
END-DEFINE
*
WRITE '=' #PARM1
*
IF #PARM2 SPECIFIED
  WRITE '#PARM2 is specified'
  WRITE '=' #PARM2
ELSE
  WRITE '#PARM2 is not specified'
* WRITE '=' #PARM2               /* would cause runtime error NAT1322
END-IF
*
IF #PARM3 NOT SPECIFIED
  WRITE '#PARM3 is not specified'
ELSE
  WRITE '#PARM3 is specified'
  WRITE '=' #PARM3
END-IF
END
```

Output of Program LOGICX07:

```
Page      1                                                 04-12-15  11:25:41

#PARM1: ABC
#PARM2 is not specified
#PARM3 is specified
#PARM3:  20
```