

User-Defined Variables

User-defined variables can be used to store intermediate results in a program or routine.

- Naming Conventions
 - Definition of Variables
 - Statement Reference Notation - r
 - Definition of Format and Length
 - Special Formats
 - Index Notation
 - Referencing a Database Array
 - Referencing the Internal Count for a Database Array
 - Qualifying Data Structures
-

Naming Conventions

When working with user-defined variables, the following naming conventions must be met.

Length of Variable Names

The name of a user-defined variable may be 1 to 32 characters long.

You can use variable names of over 32 characters (for example, in complex applications where longer meaningful variable names enhance the readability of programs); however, only the first 32 characters are significant and must therefore be unique, the remaining characters will be ignored by Natural.

Limitations of Variable Names

The name of a user-defined variable must not be a Natural reserved word.

Within one Natural program, you must not use the same name for a user-defined variable and a database field, because this might lead to referencing errors (see Qualifying Data Structures).

Characters Allowed in Variable Names

The name of a user-defined variable can consist of the following characters:

Character	Explanation
A - Z	alphabetical characters (upper and lower case)
0 - 9	numeric characters
-	hyphen
@	at sign
_	underline
/	slash
\$	dollar sign
§	paragraph sign
&	ampersand
#	hash/number sign
+	plus sign (only allowed as first character)

First Character of Variable Names

The first character of the name must be one of the following:

- an upper-case alphabetical character
- #
- +
- &

If the first character is a "#", "+" or "&", the name must consist of at least one additional character.

Variables in a GDA with a "+" as first character must be defined on Level 01. Other levels are only used in a redefinition.

"+" as the first character of a name is only allowed for application-independent variables (AIVs) and variables in a global data area. Names of AIVs must begin with a "+".

"&" as the first character of a name is used in conjunction with dynamic source program modification (see the RUN statement in the Natural Statements documentation), and as a dynamically replaceable character when defining processing rules (see the map editor description in your Natural Editors documentation).

Special Considerations Regarding the Case of Characters in Variable Names

On Windows and UNIX, lower-case characters entered as part of a variable name are internally converted to upper case. The same happens on mainframe computers if the LOWSRCE option of the COMPOPT system command is set to ON.

Lower-case characters can only be entered as the second and subsequent characters of a variable name.

On mainframe computers, lower-case characters are not translated to upper case and are therefore interpreted as being different from the respective upper-case characters, if

- the LOWSRCE option of the COMPOPT system command is set to OFF (the default value) and
- input in the editor is not translated to upper case (translation to upper case in the editor is controlled by editor profile options and by options depending on the operating system).

For example, this will cause the names #FIELD and #field to be interpreted as two different field names.

Note:

For compatibility reasons, you should not use this feature if you plan to port applications developed on mainframe computers to Windows or UNIX.

If you use lower-case characters as part of the variable name, it is highly recommended that variable names are unique regardless of their case.

Definition of Variables

You define the characteristics of a variable with the following notation:

(r,format-length/index)

This notation follows the variable name, optionally separated by one or more blanks. No blanks are allowed between the individual elements of the notation. The individual elements may be specified selectively as required, but when used together, they must be separated by the characters as indicated above.

Attention:

If operating in structured mode or if a program contains a DEFINE DATA LOCAL clause, variables cannot be defined dynamically in a statement. This does not apply to application-independent variables (AIVs).

Statement Reference Notation - r

A statement label or the source-code line number can be used to refer to a previous Natural statement. This can be used to override Natural's default referencing (as described for each statement, where applicable), or for documentation purposes.

Default Referencing of Database Fields

Generally, the following applies if you specify no statement reference notation: By default, the innermost active database loop (FIND, READ or HISTOGRAM) in which the database field in question has been read is referenced. If the field is not read in any active database loop, the last previous GET statement (in reporting mode also FIND FIRST or FIND UNIQUE statement) which has read the field is referenced.

Referencing with Statement Labels

Any Natural statement which causes a processing loop to be initiated and/or causes data elements to be accessed in the database may be marked with a symbolic label for subsequent referencing.

A label may be specified either in the form label. before the referencing object or in parentheses (label.) after the referencing object (but not both simultaneously).

The naming conventions for labels are identical to those for variables. The period after the label name serves to identify the entry as a label.

Example:

```

...
RD. READ PERSON-VIEW BY NAME STARTING FROM 'JONES'
  FD. FIND AUTO-VIEW WITH PERSONNEL-ID = PERSONNEL-ID (FD.)
      DISPLAY NAME (RD.) FIRST-NAME (RD.) MAKE (FD.)
      END-FIND
  END-READ
...

```

Referencing with Source-Code Line Numbers

A statement may also be referenced by using the number of the source-code line in which the statement is located.

All four digits of the line number must be specified (leading zeros must not be omitted).

Example:

```

...
0110 FIND EMPLOYEES-VIEW WITH NAME = 'SMITH'
0120  FIND VEHICLES-VIEW WITH MODEL = 'FORD'
0130      DISPLAY NAME (0110) MODEL (0120) 0140  END-FIND
0150 END-FIND
...

```

For further information on the referencing of statements, see the Natural Programming Guide.

Definition of Format and Length

Format and length of a user-defined variable are specified in parentheses after the variable name.

Fixed-length variables can be defined with the following formats and corresponding lengths:

Note:

For the definition of Format and Length in dynamic variables, see Definition of Dynamic Variables.

Format	Definable Length	Internal Length (in Bytes)
A Alphanumeric	1 - 1073741824 (1GB)	1 - 1073741824
B Binary	1 - 1073741824 (1GB)	1 - 1073741824
C Attribute Control	-	2
D Date	-	4
F Floating Point	4 or 8	4 or 8
I Integer	1, 2 or 4	1, 2 or 4
L Logical	-	1
N Numeric (unpacked)	1 - 29	1 - 29
P Packed numeric	1 - 29	1 - 15
T Time	-	7

Length can only be specified if format is specified. With some formats, the length need not be explicitly specified (as shown in the table above).

For fields defined with format N or P, you can use decimal position notation in the form "nn.m". "nn" represents the number of positions before the decimal point, and "m" represents the number of positions after the decimal point. The sum of the values of "nn" and "m" must not exceed 29 and the value of "m" must not exceed 7.

Note:

In reporting mode, if format and length are not specified for a user-defined variable, the default format/length N7 will be used, unless this default assignment has been disabled by the session parameter FS.

For a database field, the format/length as defined for the field in the DDM apply. (In reporting mode, it is also possible to define in a program a different format/length for a database field.)

In structured mode, format and length may only be specified in a data area definition or with a DEFINE DATA statement.

Example of Format/Length Definition - Structured Mode:

```

DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 NAME
  2 FIRST-NAME
1 #NEW-SALARY (N6.2) END-DEFINE
...
FIND EMPLOY-VIEW ...
...
COMPUTE #NEW-SALARY = ...
...

```

In reporting mode, format/length may be defined within the body of the program, if no DEFINE DATA statement is used.

Example of Format/Length Definition - Reporting Mode:

```

...
  FIND EMPLOYEES
... .. COMPUTE #NEW-SALARY(N6.2) = ...
...

```

Special Formats

In addition to the standard alphanumeric (A) and numeric (B, F, I, N, P) formats, Natural supports the special formats C, D, T and L, which are described below.

Format C - Attribute Control

A variable defined with format C may be used to assign attributes dynamically to a field used in a DISPLAY, INPUT or WRITE statement.

For a variable of format C, no length can be specified. The variable is always assigned a length of 2 bytes by Natural.

Example:

```

DEFINE DATA LOCAL
1 #ATTR(C)
1 #A(N5)
END-DEFINE
...
MOVE (AD=I CD=RE) TO #ATTR
INPUT #A (CV=#ATTR)
...

```

For further information, see the session parameter CV.

Formats D - Date, and T - Time

Variables defined with formats D and T can be used for date and time arithmetic and display. Format D can contain date information only. Format T can contain date and time information; in other words, date information is a subset of time information. Time is counted in tenths of seconds.

For variables of formats D and T, no length can be specified. A variable with format D is always assigned a length of 4 bytes (P6) and a variable with format T is always assigned a length of 7 bytes (P12) by Natural.

Example:

```

DEFINE DATA LOCAL
1 #DAT1 (D)
  END-DEFINE
*
MOVE *DATX TO #DAT1
ADD 7 TO #DAT1
WRITE '=' #DAT1
END

```

For further information, see the session parameter EM and the system variables *DATX and *TIMX.

The value in a date field must be in the range from 1st January 1582 to 31st December 2699.

Format L - Logical

A variable defined with format L may be used as a logical condition criterion. It can take the value "TRUE" or "FALSE".

For a variable of format L, no length can be specified. A variable of format L is always assigned a length of 1 byte by Natural.

Example:

```

DEFINE DATA LOCAL
1 #SWITCH(L)
  END-DEFINE
MOVE TRUE TO #SWITCH
...
IF #SWITCH
  ...
  MOVE FALSE TO #SWITCH
ELSE
  ...
  MOVE TRUE TO #SWITCH
END-IF

```

For further information on logical value presentation, see the session parameter EM.

Format "Handle"

A variable defined as "HANDLE OF dialog-element-type" can be used as a GUI handle.

A variable defined as "HANDLE OF OBJECT" can be used as an object handle.

For further information on GUI handles, see the Natural Programming Guide. For further information on object handles, see the NaturalX documentation.

Index Notation

An index notation is used for fields that represent an array.

An integer numeric constant or user-defined variable may be used in index notations. A system variable, system function or qualified variable cannot be used in index notations.

Array Definition - Examples:

1. **#ARRAY (3)**
Defines a one-dimensional array with three occurrences.
2. **FIELD (label.,A20/5) or label.FIELD(A20/5)**
Defines an array from a database field referencing the statement marked by "label." with format alphanumeric, length 20 and 5 occurrences.
3. **#ARRAY (N7.2/1:5,10:12,1:4)**
Defines an array with format/length N7.2 and three array dimensions with 5 occurrences in the first, 3 occurrences in the second and 4 occurrences in the third dimension.
4. **FIELD (label./i:i + 5) or label.FIELD(i:i + 5)**
Defines an array from a database field referencing the statement marked by "label.". FIELD represents a multiple-value field or a field from a periodic group where "i" specifies the offset index within the database occurrence. The size of the array within the program is defined as 6 occurrences (i:i + 5). The database offset index is specified as a variable to allow for the positioning of the program array within the occurrences of the multiple-value field or periodic group. For any repositioning of "i" a new access must be made to the database via a GET or GET SAME statement.

Natural allows for the definition of arrays where the index does not have to begin with "1". At runtime, Natural checks that index values specified in the reference do not exceed the maximum size of dimensions as specified in the definition.

Note:

For compatibility with Natural Version 1, an array range may be specified using a hyphen (-) instead of a colon (:). A mix of both notations, however, is not permitted. The hyphen notation is only allowed in reporting mode (but not in a DEFINE DATA statement).

The maximum index value is 1,073,741,824. The maximum size of a data area per programming object is 1,073,741,824 bytes (1 GB). Use the DSLM profile parameter (available on UNIX and Windows only) to reduce these limits for compatibility reasons to the limits applicable for Natural Version 3.1 on mainframe computers.

Simple arithmetic expressions using the "+" and "-" operators may be used in index references. When arithmetic expressions are used as indices, the operators "+" or "-" must be preceded and followed by a blank.

Arrays in group structures are resolved by Natural field by field, not group occurrence by group occurrence.

Example of Group Array Resolution:

```
DEFINE DATA LOCAL
1 #GROUP (1:2)
  2 #FIELD A (A5/1:2)
  2 #FIELD B (A5)
END-DEFINE
...
```

If the group defined above were output in a WRITE statement:

```
WRITE #GROUP (*)
```

the occurrences would be output in the following order:

```
#FIELD A(1,1) #FIELD A(1,2) #FIELD A(2,1) #FIELD A(2,2) #FIELD B(1) #FIELD B(2)
```

and not:

```
#FIELD A(1,1) #FIELD A(1,2) #FIELD B(1) #FIELD A(2,1) #FIELD A(2,2) #FIELD B(2)
```

Array Referencing - Examples:

1. **#ARRAY (1)**
References the first occurrence of a one-dimensional array.
2. **#ARRAY (7:12)**
References the seventh to twelfth occurrence of a one-dimensional array.
3. **#ARRAY (i + 5)**
References the i+fifth occurrence of a one-dimensional array.
4. **#ARRAY (5,3,7,1:4)**
Reference is made within a three dimensional array to occurrence 5 in the first dimension, occurrences 3 to 7 (5 occurrences) in the second dimension and 1 to 4 (4 occurrences) in the third dimension.
5. An asterisk may be used to reference all occurrences within a dimension:

```

DEFINE DATA LOCAL
1 #ARRAY1 (N5/1:4,1:4)
1 #ARRAY2 (N5/1:4,1:4)
END-DEFINE
...
ADD #ARRAY1 (2,*) TO #ARRAY2 (4,*)
...

```

Using a Slash before an Array Occurrence

If a variable name is followed by a 4-digit number enclosed in parentheses, Natural interprets this number as a line-number reference to a statement. Therefore a 4-digit array occurrence must be preceded by a slash "/" to indicate that it is an array occurrence; for example:

```

#ARRAY(/1000)
not: #ARRAY(1000)

```

because the latter would be interpreted as a reference to source code line 1000.

If an index variable name could be misinterpreted as a format/length specification, a slash "/" must be used to indicate that an index is being specified. If, for example, the occurrence of an array is defined by the value of the variable "N7", the occurrence must be specified as:

```

#ARRAY (/N7)
not: #ARRAY (N7)

```

because the latter would be misinterpreted as the definition of a 7-byte numeric field.

Referencing a Database Array**Referencing Multiple-Value Fields and Periodic-Group Fields**

A multiple-value field or periodic-group field within a view/DDM may be defined and referenced using various index notations.

For example, the first to tenth values and the Ith to Ith+10 values of the same multiple-value field/periodic-group field of a database record:

```

DEFINE DATA LOCAL
1 I(I2)
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 LANG (1:10)
  2 LANG (I:I + 10)
END-DEFINE

```

or:

```

RESET I(I2)
...
READ EMPLOYEES
OBTAIN LANG(1:10) LANG(I:I + 10)

```

Note:

The same lower bound index may only be used once per array, (this applies to constant indexes as well as variable indexes). For an array definition using a variable index, the lower bound must be specified using the variable by itself, and the upper bound must be specified using the same variable plus a constant.

Referencing Arrays defined with Constants

An array defined with constants may be referenced using either constants or variables. The upper bound of the array cannot be exceeded. The upper bound will be checked by Natural at compilation time if a constant is used.

```

RESET I(I2)
I = 1
READ EMPLOYEES
OBTAIN LANG(1:10)
WRITE LANG(1) / LANG(5:9) / LANG(1:10)

```

```

DEFINE DATA LOCAL
1 I(I2)
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 LANG (1:10)
END-DEFINE
*
READ EMPLOY-VIEW
FOR I 1 TO 5
  WRITE LANG(1.I)
END-FOR
END-READ
END

```

If a multiple-value field or periodic-group field is defined several times using constants and is to be referenced using variables, the following syntax is used:

```

DEFINE DATA LOCAL
1 I(I2)
  1 J(I2)
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 LANG (1:5)
  2 LANG (11:20)
END-DEFINE
*
READ EMPLOY-VIEW
FOR I 1 TO 2
  FOR J I TO 4
    DISPLAY 'LANGUAGE' LANG(1.I:J)
  
```

```

        END-FOR
    END-FOR
END-READ
END

```

Referencing Arrays defined with Variables

Multiple-value fields or periodic-group fields in arrays defined with variables must be referenced using the same variable.

```

RESET I(I2)
I = 1
READ EMPLOYEES
OBTAIN LANG(I:I+10)
WRITE LANG(I) / LANG (I+5:I+6)
END

```

If a different index is to be used, an unambiguous reference to the first encountered definition of the array with variable index must be made. This is done by qualifying the index expression as shown below:

```

RESET I(I2) J(I2)
I = 1
J = 1
READ EMPLOYEES
OBTAIN LANG(I:I+10)
WRITE LANG(I.J) / LANG(I.1:5)
END

```

The expression "I." is used to create an unambiguous reference to the array definition and "positions" to the first value within the read array range (LANG(I: I + 10)).

The current content of "I" at the time of the database access determines the starting occurrence of the database array.

Referencing Multiple-Defined Arrays

For multiple-defined arrays, a reference with qualification of the index expression is usually necessary to ensure an unambiguous reference to the desired array range.

Example:

```

DEFINE DATA LOCAL
    1 I(I2) INIT <1>
    1 J(I2) INIT <2>
    1 EMPLOY-VIEW VIEW OF EMPLOYEES
        2 LANG (1:10)
        2 LANG (5:20)
END-DEFINE
READ (2) EMPLOY-VIEW
WRITE LANG(1.1:10) / LANG(5.5:15)
DISPLAY LANG(1.I:I+2) / LANG(5.J)
END-READ
END

```

A similar syntax is also used if multiple-value fields or periodic-group fields are defined using index variables.

Example:

```

DEFINE DATA LOCAL
  1 I(I2) INIT <1>
  1 J(I2) INIT <1>
  1 N(I2) INIT <1>
  1 M(I2) INIT <1>
  1 EMPLOY-VIEW VIEW OF EMPLOYEES
    2 LANG (I:I+10)
    2 LANG (J:J+5)
    2 LANG (1:2)
END-DEFINE
READ (2) EMPLOY-VIEW
  WRITE LANG(I.I) / LANG(I.I:I+10)
  DISPLAY LANG(J.N) / LANG(J.N:M)
  DISPLAY LANG(1.N) / LANG(1.N:M)
END-READ
END

```

Referencing the Internal Count for a Database Array

It is sometimes necessary to reference a multiple-value field and/or a periodic group without knowing how many values/occurrences exist in a given record. Adabas maintains an internal count of the number of values of each multiple-value field and the number of occurrences of each periodic group. This count may be referenced by specifying "C*" immediately before the field name. See also the data-area-editor line command ".*" (as described in your Natural Editor documentation).

The count is returned in format N3.

Examples:

C*LANG	Returns the count of the number of values for the multiple-value field LANG.
C*INCOME	Returns the count of the number of occurrences for the periodic group INCOME.
C*BONUS(1)	Returns the count of the number of values for the multiple-value field BONUS in periodic group occurrence 1 (assuming that BONUS is a multiple-value field within a periodic group.)

Note for SQL databases:

The C* notation cannot be used for SQL databases.

Note for VSAM databases:

The C* notation does not return the number of values/occurrences but the maximum occurrence/value as defined in the DDM (MAXOCC).

Example of C* Variable:

```

/* EXAMPLE 'ICOUNT':
/* USING C*NOTATION TO OBTAIN INTERNAL COUNT FOR DATABASE ARRAY
/*****
LIMIT 2
READ EMPLOYEES BY CITY
OBTAIN SALARY(1:5)
WRITE NOTITLE 'NAME:' NAME / 'NUMBER OF LANGUAGES SPOKEN:' C*LANG 5X 'LANGUAGE 1:' LANG (1)
          5X 'LANGUAGE 2:' LANG (2)
/*****
WRITE 'SALARY DATA:'
FOR #A (N1) FROM 1 TO C*INCOME
  WRITE 'SALARY' #A SALARY (1.#A)
LOOP
/*****
WRITE 'THIS YEAR BONUS:' C*BONUS(1) BONUS (1,1) BONUS (1,2)
  / 'LAST YEAR BONUS:' C*BONUS(2) BONUS (2,1) BONUS (2,2)
SKIP 1
END

```

```

NAME: SENKO
NUMBER OF LANGUAGES SPOKEN:   1      LANGUAGE 1: ENG      LANGUAGE 2:
SALARY DATA:
SALARY  1      31500
SALARY  2      29900
SALARY  3      28100
SALARY  4      26600
SALARY  5      25200
THIS YEAR BONUS:   0          0          0
LAST YEAR BONUS:  0          0          0

NAME: GODEFROY
NUMBER OF LANGUAGES SPOKEN:   1      LANGUAGE 1: FRE      LANGUAGE 2:
SALARY DATA:
SALARY  1      170300
THIS YEAR BONUS:   1      50000      0
LAST YEAR BONUS:  0          0          0

```

C* for Multiple-Value Fields Within Periodic Groups

For a multiple-value field within a periodic group, you can also define a C* variable with an index range specification.

The following examples use the multiple-value field BONUS, which is part of the periodic group INCOME. All three examples yield the same result.

Example 1 - Reporting Mode:

```

READ EMPLOYEES BY PERSONNEL-ID FROM 11100117
  OBTAIN C*BONUS (1:3)
          BONUS (1:3,1:3)
*
  DISPLAY C*BONUS (1:3)
          BONUS (1:3,1:3)
END

```

Example 2 - Structured Mode:

```

DEFINE DATA LOCAL
1 EMP VIEW OF EMPLOYEES
  2 PERSONNEL-ID
  2 INCOME (1:3)
  3 C*BONUS
  3 BONUS (1:3)
END-DEFINE
READ EMP BY PERSONNEL-ID FROM 11100117
  DISPLAY C*BONUS (1:3)
  BONUS (1:3,1:3)
END-READ
END

```

Example 3 - Structured Mode:

```

DEFINE DATA LOCAL
1 EMP VIEW OF EMPLOYEES
  2 PERSONNEL-ID
  2 C*BONUS (1:3)
  2 INCOME (1:3)
  3 BONUS (1:3)
END-DEFINE
READ EMP BY PERSONNEL-ID FROM 11100117
  DISPLAY PERSONNEL-ID C*BONUS (*) BONUS (*,*)
END-READ
END

```

Note:

As the Adabas format buffer does not permit ranges for count fields, they are generated as individual fields; therefore a C* index range for a large array may cause an Adabas format buffer overflow.

Qualifying Data Structures

To identify a field when referencing it, you may qualify the field; that is, before the field name, you specify the name of the level-1 data element in which the field is located and a period.

If a field cannot be identified uniquely by its name (for example, if the same field name is used in multiple groups/views), you must qualify the field when you reference it.

The combination of level-1 data element and field name must be unique.

Example:

```
DEFINE DATA LOCAL
1 FULL-NAME
  2 LAST-NAME (A20)
  2 FIRST-NAME (A15)
1 OUTPUT-NAME
  2 LAST-NAME (A20)
  2 FIRST-NAME (A15)
END-DEFINE
...
MOVE FULL-NAME.LAST-NAME TO OUTPUT-NAME.LAST-NAME
...
```

The qualifier must be a level-1 data element.

Example:

```
DEFINE DATA LOCAL
1 GROUP1
  2 SUB-GROUP
    3 FIELD1 (A15)
    3 FIELD2 (A15)
END-DEFINE
...
MOVE 'ABC' TO GROUP1.FIELD1
...
```

Note:

If you use the same name for a user-defined variable and a database field (which you should not do anyway), you must qualify the database field when you want to reference it; because if you do not, the user-defined variable will be referenced instead.