Using Dynamic and Large Variables

This chapter covers the following topics:

- General Remarks
- Assignments with Dynamic Variables
- Initialization of Dynamic Variables
- String Manipulation with Dynamic Alphanumeric Variables
- Logical Condition Criterion (LCC) with Dynamic Variables
- AT/IF-BREAK of Dynamic Control Fields
- Parameter Transfer with Dynamic Variables
- Work File Access with Large and Dynamic Variables
- Performance Aspects with Dynamic Variables
- Outputting Dynamic Variables
- Dynamic X-Arrays

General Remarks

Generally, the following rules apply:

- A dynamic alphanumeric field may be used wherever an alphanumeric field is allowed.
- A dynamic binary field may be used wherever a binary field is allowed.
- A dynamic Unicode field may be used wherever a Unicode field is allowed.

Exception:

Dynamic variables are not allowed within the `SORT` statement. To use dynamic variables in a `DISPLAY`, `WRITE`, `PRINT`, `REINPUT` or `INPUT` statement, you must use either the session parameter `AL` or `EM` to define the length of the variable.

The used length (as indicated by the Natural system variable `*LENGTH`, see *Value Space Currently Used for a Dynamic Variable*) and the size of the allocated storage of dynamic variables are equal to zero until the variable is accessed as a target operand for the first time. Due to assignments or other manipulation operations, dynamic variables may be firstly allocated or extended (reallocated) to the exact size of the source operand.

The size of a dynamic variable may be extended if it is used as a modifiable operand (target operand) in the following statements:

ASSIGN	*operand1* (destination operand in an assignment).
CALLNAT	See *Parameter Transfer with Dynamic Variables* (except if AD=O, or if BY VALUE exists in the corresponding parameter data area).
COMPRESS	*operand2*, see *Processing*.
EXAMINE	*operand1* in the DELETE REPLACE clause.
MOVE	*operand2* (destination operand), see *Function*.
PERFORM	(except if AD=O, or if BY VALUE exists in the corresponding parameter data area).
READ WORK FILE	*operand1* and *operand2*, see *Handling of Large and Dynamic Variables*.
SEPARATE	*operand4*.
SELECT (SQL)	Parameter in the INTO clause.
SEND METHOD	*operand3* (except if AD=O).

Currently, there is the following limit concerning the usage of large variables:

CALL	Parameter size less than 64 KB per parameter (no limit for the CALL with INTERFACE4 option).

In the following sections, the use of dynamic variables is discussed in more detail with examples.

Assignments with Dynamic Variables

Generally, an assignment is done in the current used length (as indicated by the Natural system variable *LENGTH) of the source operand. If the destination operand is a dynamic variable, its current allocated size is possibly extended in order to move the source operand without truncation.

Example:

```
#MYDYNTEXT1 := OPERAND
MOVE OPERAND TO #MYDYNTEXT1
/* #MYDYNTEXT1 IS AUTOMATICALLY EXTENDED UNTIL THE SOURCE OPERAND CAN BE COPIED
```

MOVE ALL, MOVE ALL UNTIL with dynamic target operands are defined as follows:

- MOVE ALL moves the source operand repeatedly to the target operand until the used length (*LENGTH) of the target operand is reached. *LENGTH is not modified. If *LENGTH is zero, the statement will be ignored.
- MOVE ALL *operand1* TO *operand2* UNTIL operand3 moves *operand1* repeatedly to *operand2* until the length specified in *operand3* is reached. If *operand3* is greater than *LENGTH(*operand2*), *operand2* is extended and *LENGTH(*operand2*) is set to *operand3*. If *operand3* is less than *LENGTH(*operand2*), the used length is reduced to *operand3*. If *operand3* equals *LENGTH(*operand2*), the behavior is equivalent to MOVE ALL.

Example:

```
#MYDYNTEXT1 := 'ABCDEFGHIJKLMNO'          /* *LENGTH(#MYDYNTEXT1) = 15
MOVE ALL 'AB' TO #MYDYNTEXT1              /* CONTENT OF #MYDYNTEXT1 = 'ABABABABABABABA';
                                          /* *LENGTH IS STILL 15
MOVE ALL 'CD' TO #MYDYNTEXT1 UNTIL 6      /* CONTENT OF #MYDYNTEXT1 = 'CDCDCD';
                                          /* *LENGTH = 6
MOVE ALL 'EF' TO #MYDYNTEXT1 UNTIL 10     /* CONTENT OF #MYDYNTEXT1 = 'EFEFEFEFEF';
                                          /* *LENGTH = 10
```

`MOVE JUSTIFIED` is rejected at compile time if the target operand is a dynamic variable.

`MOVE SUBSTR` and `MOVE TO SUBSTR` are allowed. `MOVE SUBSTR` will lead to a runtime error if a sub-string behind the used length of a dynamic variable (`*LENGTH`) is referenced. `MOVE TO SUBSTR` will lead to a runtime error if a sub-string position behind `*LENGTH + 1` is referenced, because this would lead to an undefined gap in the content of the dynamic variable. If the target operand should be extended by `MOVE TO SUBSTR` (for example if the second operand is set to `*LENGTH+1`), the third operand is mandatory.

Valid syntax:

```
#OP2 := *LENGTH(#MYDYNTEXT1)
MOVE SUBSTR (#MYDYNTEXT1, #OP2) TO OPERAND                /* MOVE LAST CHARACTER TO OPERAND
#OP2 := *LENGTH(#MYDYNTEXT1) + 1
MOVE OPERAND TO SUBSTR(#MYDYNTEXT1, #OP2, #lEN_OPERAND)   /* CONCATENATE OPERAND TO #MYDYNTEXT1
```

Invalid syntax:

```
#OP2 := *LENGTH(#MYDYNTEXT1) + 1
MOVE SUBSTR (#MYDYNTEXT1, #OP2, 10) TO OPERAND           /* LEADS TO RUNTIME ERROR; UNDEFINED SUB-STRING
#OP2 := *LENGTH(#MYDYNTEXT1 + 10)
MOVE OPERAND TO SUBSTR(#MYDYNTEXT1, #OP2, #EN_OPERAND)   /* LEADS TO RUNTIME ERROR; UNDEFINED GAP
#OP2 := *LENGTH(#MYDYNTEXT1) + 1
MOVE OPERAND TO SUBSTR(#MYDYNTEXT1, #OP2)                /* LEADS TO RUNTIME ERROR; UNDEFINED LENGTH
```

Assignment Compatibility

Example:

```
#MYDYNTEXT1   := #MYSTATICVAR1
#MYSTATICVAR1 := #MYDYNTEXT2
```

If the source operand is a static variable, the used length of the dynamic destination operand (`*LENGTH(#MYDYNTEXT1)`) is set to the format length of the static variable and the source value is copied in this length including trailing blanks (alphanumeric and Unicode fields) or binary zeros (for binary fields).

If the destination operand is static and the source operand is dynamic, the dynamic variable is copied in its currently used length. If this length is less than the format length of the static variable, the remainder is filled with blanks (for alphanumeric and Unicode fields) or binary zeros (for binary fields). Otherwise, the value will be truncated. If the currently used length of the dynamic variable is 0, the static target operand is filled with blanks (for alphanumeric and Unicode fields) or binary zeros (for binary fields).

Initialization of Dynamic Variables

Dynamic variables can be initialized with blanks (alphanumeric and Unicode fields) or zeros (binary fields) up to the currently used length (= *LENGTH) using the RESET statement. *LENGTH is not modified.

Example:

```
DEFINE DATA LOCAL
1 #MYDYNTEXT1   (A)  DYNAMIC
END-DEFINE
#MYDYNTEXT1 := 'SHORT TEXT'
WRITE *LENGTH(#MYDYNTEXT1)          /* USED LENGTH = 10
RESET #MYDYNTEXT1                   /* USED LENGTH = 10, VALUE = 10 BLANKS
```

To initialize a dynamic variable with a specified value in a specified size, the MOVE ALL UNTIL statement may be used.

Example:

```
MOVE ALL 'Y' TO #MYDYNTEXT1 UNTIL 15          /* #MYDYNTEXT1 CONTAINS 15 'Y'S, USED LENGTH = 15
```

String Manipulation with Dynamic Alphanumeric Variables

If a modifiable operand is a dynamic variable, its current allocated size is possibly extended in order to perform the operation without truncation or an error message. This is valid for the concatenation (COMPRESS) and separation of dynamic alphanumeric variables (SEPARATE).

Example:

```
** Example 'DYNAMX01': Dynamic variables (with COMPRESS and SEPARATE)
************************************************************************
DEFINE DATA LOCAL
1 #MYDYNTEXT1 (A)   DYNAMIC
1 #TEXT       (A20)
1 #DYN1       (A)   DYNAMIC
1 #DYN2       (A)   DYNAMIC
1 #DYN3       (A)   DYNAMIC
END-DEFINE
*
MOVE ' HELLO WORLD ' TO #MYDYNTEXT1
WRITE #MYDYNTEXT1 (AL=25) 'with length' *LENGTH (#MYDYNTEXT1)
/*  dynamic variable with leading and trailing blanks
*
MOVE ' HELLO WORLD ' TO #TEXT
*
MOVE #TEXT TO #MYDYNTEXT1
WRITE #MYDYNTEXT1 (AL=25) 'with length' *LENGTH (#MYDYNTEXT1)
/*  dynamic variable with whole variable length of #TEXT
*
COMPRESS #TEXT INTO #MYDYNTEXT1
WRITE #MYDYNTEXT1 (AL=25) 'with length' *LENGTH (#MYDYNTEXT1)
/*  dynamic variable with leading blanks of #TEXT
*
*
#MYDYNTEXT1 := 'HERE COMES THE SUN'
SEPARATE #MYDYNTEXT1 INTO #DYN1 #DYN2 #DYN3 IGNORE
*
```

```
WRITE / #MYDYNTEXT1 (AL=25) 'with length' *LENGTH (#MYDYNTEXT1)
WRITE #DYN1 (AL=25) 'with length' *LENGTH (#DYN1)
WRITE #DYN2 (AL=25) 'with length' *LENGTH (#DYN2)
WRITE #DYN3 (AL=25) 'with length' *LENGTH (#DYN3)
/*  #DYN1, #DYN2, #DYN3 are automatically extended or reduced
*
EXAMINE #MYDYNTEXT1 FOR 'SUN' REPLACE 'MOON'
WRITE / #MYDYNTEXT1 (AL=25) 'with length' *LENGTH (#MYDYNTEXT1)
/*  #MYDYNTEXT1 is automatically extended or reduced
*
END
```

Note:
In case of non-dynamic variables, an error message may be returned.

Logical Condition Criterion (LCC) with Dynamic Variables

Generally, a read-only operation (such as a comparison) with a dynamic variable is done with its currently used length. Dynamic variables are processed like static variables if they are used in a read-only (non-modifiable) context.

Example:

```
IF  #MYDYNTEXT1  =  #MYDYNTEXT2  OR  #MYDYNTEXT1 =  "**"  THEN ...
IF  #MYDYNTEXT1  <  #MYDYNTEXT2  OR  #MYDYNTEXT1 <  "**"  THEN ...
IF  #MYDYNTEXT1  >  #MYDYNTEXT2  OR  #MYDYNTEXT1 >  "**"  THEN ...
```

Trailing blanks for alphanumeric and Unicode variables or leading binary zeros for binary variables are processed in the same way for static and dynamic variables. For example, alphanumeric variables containing the values AA and AA followed by a blank will be considered being equal, and binary variables containing the values `H'0000031'` and `H'3031'` will be considered being equal. If a comparison result should only be TRUE in case of an exact copy, the used lengths of the dynamic variables have to be compared in addition. If one variable is an exact copy of the other, their used lengths are also equal.

Example:

```
#MYDYNTEXT1  :=  'HELLO'                                    /* USED LENGTH IS 5
#MYDYNTEXT2  :=  'HELLO     '                               /* USED LENGTH IS 10
IF   #MYDYNTEXT1  =  #MYDYNTEXT2   THEN  ...                /* TRUE
IF   #MYDYNTEXT1  =  #MYDYNTEXT2 AND
     *LENGTH(#MYDYNTEXT1) = *LENGTH(#MYDYNTEXT2)  THEN  ...  /* FALSE
```

Two dynamic variables are compared position by position (from left to right for alphanumeric variables, and right to left for binary variables) up to the minimum of their used lengths. The first position where the variables are not equal determines if the first or the second variable is greater than, less than or equal to the other. The variables are equal if they are equal up to the minimum of their used lengths and the remainder of the longer variable contains only blanks for alphanumeric dynamic variables or binary zeros for binary dynamic variables. To compare two Unicode dynamic variables, 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.

Example:

```
#MYDYNTEXT1  :=  'HELLO1'                    /* USED LENGTH IS 6
#MYDYNTEXT2  :=  'HELLO2'                    /* USED LENGTH IS 10
IF #MYDYNTEXT1  <  #MYDYNTEXT2 THEN  ...     /* TRUE
#MYDYNTEXT2  :=  'HALLO'
IF #MYDYNTEXT1  >  #MYDYNTEXT2 THEN  ...     /* TRUE
```

Comparison Compatibility

Comparisons between dynamic and static variables are equivalent to comparisons between dynamic variables. The format length of the static variable is interpreted as its used length.

Example:

```
#MYSTATTEXT1 :=  'HELLO'                      /* FORMAT LENGTH OF MYSTATTEXT1 IS A20
#MYDYNTEXT1  :=  'HELLO'                      /* USED LENGTH IS 5
IF  #MYSTATTEXT1  =  #MYDYNTEXT1  THEN  ...   /* TRUE
IF  #MYSTATTEXT1  >  #MYDYNTEXT1  THEN  ...   /* FALSE
```

AT/IF-BREAK of Dynamic Control Fields

The comparison of the break control field with its old value is performed position by position from left to right. If the old and the new value of the dynamic variable are of different length, then for comparison, the value with shorter length is padded to the right (with blanks for alphanumeric and Unicode dynamic values or binary zeros for binary values).

In case of an alphanumeric or Unicode break control field, trailing blanks are not significant for the comparison, i.e. trailing blanks do not mean a change of the value and no break occurs.

In case of a binary break control field, trailing binary zeros are not significant for the comparison, i.e. trailing binary zeros do not mean a change of the value and no break occurs.

Parameter Transfer with Dynamic Variables

Dynamic variables may be passed as parameters to a called program object (`CALLNAT`, `PERFORM`). Call-by-reference is possible because the value space of a dynamic variable is contiguous. Call-by-value causes an assignment with the variable definition of the caller as the source operand and the parameter definition as the destination operand. Call-by-value result causes in addition the movement in the opposite direction.

For call-by-reference, both definitions must be `DYNAMIC`. If only one of them is `DYNAMIC`, a runtime error is raised. In the case of call-by-value (result), all combinations are possible. The following table illustrates the valid combinations:

Call By Reference

Caller	Parameter	
	Static	**Dynamic**
Static	Yes	No
Dynamic	No	Yes

The formats of dynamic variables A or B must match.

Call by Value (Result)

Caller	Parameter	
	Static	**Dynamic**
Static	Yes	Yes
Dynamic	Yes	Yes

Note:
In the case of static/dynamic or dynamic/static definitions, a value truncation may occur according to the data transfer rules of the appropriate assignments.

Example 1:

```
** Example 'DYNAMX02': Dynamic variables (as parameters)
************************************************************************
DEFINE DATA LOCAL
1 #MYTEXT (A) DYNAMIC
END-DEFINE
*
#MYTEXT := '123456'      /* extended to 6 bytes, *LENGTH(#MYTEXT) = 6
*
CALLNAT 'DYNAMX03' USING #MYTEXT
*
WRITE *LENGTH(#MYTEXT)          /* *LENGTH(#MYTEXT) = 8
*
END
```

Subprogram DYNAMX03:

```
** Example 'DYNAMX03': Dynamic variables (as parameters)
************************************************************************
DEFINE DATA PARAMETER
1 #MYPARM (A) DYNAMIC  BY VALUE RESULT
END-DEFINE
*
WRITE *LENGTH(#MYPARM)                  /* *LENGTH(#MYPARM) = 6
#MYPARM := '1234567'                    /* *LENGTH(#MYPARM) = 7
#MYPARM := '12345678'                   /* *LENGTH(#MYPARM) = 8
EXPAND DYNAMIC VARIABLE #MYPARM TO 10   /* 10 bytes are allocated
*
WRITE *LENGTH(#MYPARM)                  /* *LENGTH(#MYPARM) = 8
*
/* content of #MYPARM is moved back to #MYTEXT
/* used length of #MYTEXT = 8
*
END
```

Example 2:

```
** Example 'DYNAMX04': Dynamic variables (as parameters)
************************************************************************
DEFINE DATA LOCAL
1 #MYTEXT (A) DYNAMIC
END-DEFINE
*
#MYTEXT := '123456'      /* extended to 6 bytes, *LENGTH(#MYTEXT) = 6
*
CALLNAT 'DYNAMX05' USING #MYTEXT
*
WRITE *LENGTH(#MYTEXT)          /* *LENGTH(#MYTEXT) = 8
                                /* at least 10 bytes are
                                /* allocated (extended in DYNAMX05)
*
END
```

Subprogram DYNAMX05:

```
** Example 'DYNAMX05': Dynamic variables (as parameters)
************************************************************************
DEFINE DATA PARAMETER
1 #MYPARM (A) DYNAMIC
END-DEFINE
*
WRITE *LENGTH(#MYPARM)                  /* *LENGTH(#MYPARM) = 6
#MYPARM := '1234567'                    /* *LENGTH(#MYPARM) = 7
#MYPARM := '12345678'                   /* *LENGTH(#MYPARM) = 8
EXPAND DYNAMIC VARIABLE #MYPARM TO 10   /* 10 bytes are allocated
*
WRITE *LENGTH(#MYPARM)                  /* *LENGTH(#MYPARM) = 8
*
END
```

CALL 3GL Program

Dynamic and large variables can sensibly be used with the CALL statement when the option INTERFACE4 is used. Using this option leads to an interface to the 3GL program with a different parameter structure.

Before calling a 3GL program with dynamic parameters, it is important to ensure that the necessary buffer size is allocated. This can be done explicitly with the EXPAND statement.

If an initialized buffer is required, the dynamic variable can be set to the initial value and to the necessary size by using the MOVE ALL UNTIL statement. Natural provides a set of functions that allow the 3GL program to obtain information about the dynamic parameter and to modify the length when parameter data is passed back.

Example:

```
MOVE ALL ' ' TO  #MYDYNTEXT1 UNTIL 10000
  /* a buffer of length 10000 is allocated
  /* #MYDYNTEXT1 is initialized with blanks
  /* and *LENGTH(#MYDYNTEXT1) = 10000
CALL INTERFACE4  'MYPROG'  USING  #MYDYNTEXT1
WRITE *LENGTH(#MYDYNTEXT1)
  /* *LENGTH(#MYDYNTEXT1) may have changed in the 3GL program
```

For a more detailed description, refer to the CALL statement in the *Statements* documentation.

Work File Access with Large and Dynamic Variables

There is no difference in the treatment of fixed length variables with a length of less than or equal to 253 and large variables with a length of greater than 253.

Dynamic variables are written in the length that is in effect (i.e. the value of system variable *LENGTH for this variable) when the WRITE WORK FILE statement is executed. Since the length can be different for each execution of the same WRITE WORK FILE statement, the keyword VARIABLE must be specified.

When reading work files of type FORMATTED, a dynamic variable is filled in the length that is in effect (i.e. the value of system variable *LENGTH for this variable) when the READ WORK FILE statement is executed. If the dynamic variable is longer than the remaining data in the current record, it is padded with blanks for alphanumeric and Unicode fields and binary zeros for binary fields.

When reading a work file of type UNFORMATTED, a dynamic variable is filled with the remainder of the work file. Its size is adjusted accordingly, and is reflected in the value of system variable *LENGTH for this variable.

Performance Aspects with Dynamic Variables

If a dynamic variable is to be expanded in small quantities multiple times (for example, byte-wise), use EXPAND before the iterations if the upper limit of required storage is (approximately) known. This avoids additional overhead to adjust the storage needed.

Use REDUCE or RESIZE if the dynamic variable will no longer be needed, especially for variables with a high value of *LENGTH. This enables Natural to release or reuse the storage. Thus, the overall performance may be improved.

The amount of the allocated memory of a dynamic variable may be reduced using the REDUCE DYNAMIC VARIABLE statement. In order to (re)allocate a variable to a specified length, the EXPAND statement can be used. (If the variable should be initialized, use the MOVE ALL UNTIL statement.)

Example:

```
** Example 'DYNAMX06': Dynamic variables (allocated memory)
************************************************************************
DEFINE DATA LOCAL
1 #MYDYNTEXT1  (A)  DYNAMIC
1 #LEN         (I4)
END-DEFINE
*
#MYDYNTEXT1 := 'a'     /* used length is 1, value is 'a'
                       /* allocated size is still 1
WRITE *LENGTH(#MYDYNTEXT1)
*
EXPAND DYNAMIC VARIABLE  #MYDYNTEXT1 TO 100
                       /* used length is still 1, value is 'a'
                       /* allocated size is 100
*
CALLNAT 'DYNAMX05' USING #MYDYNTEXT1
WRITE *LENGTH(#MYDYNTEXT1)
                       /* used length and allocated size
                       /* may have changed in the subprogram
```

```
*
#LEN := *LENGTH(#MYDYNTEXT1)
REDUCE DYNAMIC VARIABLE  #MYDYNTEXT1  TO  #LEN
                       /* if allocated size is greater than used length,
                       /* the unused memory is released
*
REDUCE DYNAMIC VARIABLE  #MYDYNTEXT1  TO  0
WRITE *LENGTH(#MYDYNTEXT1)
                       /* free allocated memory for dynamic variable
END
```

Rules:

- Use dynamic operands where it makes sense.
- Use EXPAND if upper limit of memory usage is known.
- Use REDUCE if the dynamic operand will no longer be needed.

Outputting Dynamic Variables

Dynamic variables may be used inside output statements like the following:

Statement	Notes
DISPLAY	With these statements, you must set the format of the output or input of dynamic variables using the AL (Alphanumeric Length for Output) or EM (Edit Mask) session parameters.
WRITE	
INPUT	
REINPUT	--
PRINT	Because the output of the PRINT statement is unformatted, the output of dynamic variables in the PRINT statement need not be set using AL and EM parameters. In other words, these parameters may be omitted.

Dynamic X-Arrays

A dynamic X-array may be allocated by first specifying the number of occurrences and then expanding the length of the previously allocated array occurrences.

Example:

```
DEFINE DATA LOCAL
1 #X-ARRAY(A/1:*)  DYNAMIC
END-DEFINE
*
EXPAND  ARRAY   #X-ARRAY TO (1:10)  /* Current boundaries (1:10)
#X-ARRAY(*) := 'ABC'
EXPAND  ARRAY   #X-ARRAY TO (1:20)  /* Current boundaries (1:20)
#X-ARRAY(11:20) := 'DEF'
```