

# Programming Language Enhancements

The following programming language enhancements are provided with Natural Version 4.1:

- New Statements
  - Enhanced Statements
  - Size of Alphanumeric and Binary Variables
  - Size of Data Elements
  - Dynamic Variables
  - Object Handles
  - Optional Parameters
  - SPECIFIED Option in Logical Condition
  - MASK Option in Logical Condition
  - System Variables Now Available under Natural for Mainframes
  - New System Variables
  - Changed System Variable
  - More Precise Results for Floating Point Conversion
  - Precision of Floating Point Format Results of an Arithmetic Operation Improved
  - Arithmetic Operations with Date and Time Enhanced
  - Evaluation of SQRT for (Un)Packed Number Enhanced
  - Substring Evaluation in MASK Corrected
  - New Compiler Options
-

## New Statements

The following new Natural statements are available with Natural Version 4.1:

- EXPAND
- REDUCE
- RESIZE

### EXPAND Statement

The new statement EXPAND is used to increase the size of the currently allocated storage of a dynamic variable.

In the statement, you specify the name of the variable and its desired size. If that size is smaller than the size of the storage currently allocated to that dynamic variable, the EXPAND statement has no effect.

For further information, see Dynamic Variables.

### REDUCE Statement

The new statement REDUCE is used to reduce the size of the used storage (available to the programmer).

In the statement, you specify the name of the variable and its desired size.

The storage allocated to the dynamic variable beyond the specified size may be released at any time, when the statement is executed or at a later time.

If the currently used size (as contained in the new system variable \*LENGTH) of the dynamic variable is greater than the given size, \*LENGTH is set to the specified size and the content of the variable is truncated (but not modified). If the specified size is larger than the size of the storage currently allocated to the dynamic variable, the REDUCE statement will be ignored.

For further information, see Dynamic Variables.

### RESIZE Statement

The new statement RESIZE is used to adjust the length of a dynamic variable to exactly the size specified.

- If the specified size is smaller than the used size (as indicated by \*LENGTH) of the dynamic variable, the used size is reduced accordingly.
- If the specified size is larger than the size of the storage currently allocated to the dynamic variable, the size of the storage allocated to the dynamic variable is increased. The currently used size (as indicated by \*LENGTH) of the dynamic variable is not affected and remains unchanged.
- If the specified size is the same as the size of the storage currently allocated to the dynamic variable, the execution of the RESIZE statement has no effect.

For further information, see Dynamic Variables.

## Enhanced Statements

The following Natural statements have been enhanced:

- CALL
- CALLNAT
- CLOSE PRINTER
- DEFINE DATA
- DEFINE PRINTER
- DEFINE WORK FILE
- ESCAPE
- FIND
- HISTOGRAM
- INPUT
- PERFORM
- READ
- SEND METHOD

### CALL Statement

The CALL statement provides the following enhancements:

- The limit of 32 KB for the maximum length per parameter has been removed.
- A new option, INTERFACE4, provides for enhanced parameter descriptions. Also, with this option, the number of parameters to be passed to the invoked non-Natural program (currently 40) has been raised to 32767.

### CALLNAT Statement

The CALLNAT statement provides the following enhancements:

- **Notation "nX"** - see Optional Parameters.
- **Parameter Transfer with Dynamic Variables** - see Dynamic Variables.

### CLOSE PRINTER Statement

The CLOSE PRINTER statement provides a new option that enables the hardcopy printer to be specified as Printer 0 to close the printer.

### DEFINE DATA Statement

The DEFINE DATA statement provides two new options to be specified in the *parameter-data-definition* of a DEFINE DATA PARAMETER statement:

<b>DYNAMIC</b>	If you define a parameter as DYNAMIC, its length will be determined at runtime. For further information, see Dynamic Variables.
<b>OPTIONAL</b>	By default, a parameter is defined without OPTIONAL, which means that a value <b>must</b> be passed from the invoking object to the parameter. If you define a parameter as OPTIONAL, a value can - but need not - be passed from the invoking object to this parameter. For further information, see Optional Parameters.

## DEFINE PRINTER Statement

The DEFINE PRINTER statement provides a new option that enables the hardcopy printer to be specified as Printer 0 to define a printer.

## DEFINE WORK FILE Statement

The DEFINE WORK FILE statement provides a new option TYPE UNFORMATTED allows you to specify that a work file is to be used in stream mode instead of the other new option FORMATTED for record-oriented files.

## ESCAPE Statement

The ESCAPE statement provides the following enhancements:

- ESCAPE TOP REPOSITION
- ESCAPE MODULE

### ESCAPE TOP REPOSITION

The new option ESCAPE TOP REPOSITION allows you to dynamically reposition within a READ statement loop that is being executed, and restart the READ loop with another start value.

When an ESCAPE TOP REPOSITION statement is executed, Natural immediately continues processing at the top of the active READ loop, using the current value of the search variable as a new start value.

At the same time, ESCAPE TOP REPOSITION resets the system variable \*COUNTER to "0".

ESCAPE TOP REPOSITION can be specified within a READ statement loop accessing an Adabas, DL/I or VSAM database. The READ statement concerned must contain the option WITH REPOSITION.

### ESCAPE MODULE

This new option ESCAPE MODULE allows you to stop an inline subroutine and continue processing with the programming object which has invoked the object containing the inline subroutine.

When used within a subroutine, the existing option ESCAPE ROUTINE causes processing to continue with the statement following the PERFORM statement that has invoked the subroutine. In the case of an **inline** subroutine, this would be within the same programming object. If nested subroutines are used, that is, if the PERFORM statement is itself contained within another inline subroutine, it would take a lot of coding to leave the active programming object entirely.

The new option ESCAPE MODULE, however, will not only stop the processing of the inline subroutine, but also of the programming object containing the inline subroutine; processing will then continue with the object invoking that programming object. This will be particularly useful when multiple nested inline subroutines are used, as a single ESCAPE MODULE statement will suffice to leave the programming object altogether.

ESCAPE MODULE is only relevant in **inline** subroutines. In external subroutines, subprograms and invoked programs, it has the same effect as ESCAPE ROUTINE.

As with ESCAPE ROUTINE, loop-end processing will be performed. If you specify the keyword IMMEDIATE, no loop-end processing is performed.

## FIND Statement

The FIND statement provides the following enhancement:

### Multi-Fetch

Traditionally, Natural retrieves database records one by one. However, Adabas's Multi-Fetch functionality makes it possible to retrieve more than one database record per database access.

To make use of this functionality, the FIND statement provides a new MULTI-FETCH option. With this option, you are able to specify the number of records to be retrieved per database access when the statement is executed.

The MULTI-FETCH option is available for accesses to Adabas databases only. For database updates, the MULTI-FETCH option cannot be used.

MULTI-FETCH only affects the way in which the records are retrieved from the database. The program's record-processing logic is not affected; that is, the number of FIND processing loops executed is the same as without MULTI-FETCH, and the records are still processed one by one.

## HISTOGRAM Statement

The HISTOGRAM statement provides the following enhancements:

- Dynamic Change of Reading Direction
- New Comparators
- Multi-Fetch
- End of Range Condition (ENDING AT) Controlled by Database

### Dynamic Change of Reading Direction

With Natural Version 3.1, the database field values to be retrieved by a HISTOGRAM statement could be read in ascending or descending sequence. This is determined by the keywords ASCENDING and DESCENDING in the SEQUENCE clause. Also, the VARIABLE option allows you to determine the reading direction at runtime. However, once the HISTOGRAM statement is executed, you cannot change the reading direction.

With Natural Version 4.1, the new keyword DYNAMIC is provided for the SEQUENCE clause: It allows you to change the reading direction from ascending to descending (or vice versa) within an active HISTOGRAM processing loop that is being executed, without having to restart the loop. After the keyword DYNAMIC, you specify a variable to which the value "A" (for "ascending") or "D" (for "descending") can be assigned. The DYNAMIC option is available for accesses to Adabas and DB2 databases.

### New Comparators

In addition to the comparators EQUAL TO, STARTING FROM and ENDING AT, Natural Version 4.1 provides the possibility to specify start/end values with the following options:

- LESS THAN
- GREATER THAN
- LESS EQUAL
- GREATER EQUAL

These new comparators are available for accesses to Adabas, DB2, DL/I and VSAM databases.

## Multi-Fetch

Traditionally, Natural retrieves database records one by one. However, Adabas's Multi-Fetch functionality makes it possible to retrieve more than one database record per database access.

To make use of this functionality, the HISTOGRAM statement provides a new MULTI-FETCH option. With this option, you are able to specify the number of records to be retrieved per database access when the statement is executed.

The MULTI-FETCH option is available for accesses to Adabas databases only.

MULTI-FETCH only affects the way in which the records are retrieved from the database. The program's record-processing logic is not affected; that is, the number of HISTOGRAM processing loops executed is the same as without MULTI-FETCH, and the records are still processed one by one.

## End of Range Condition (ENDING AT) Controlled by Database

With Natural Version 3.1, if the ENDING AT clause was used to limit the range of values to be read, Natural internally read one value beyond the specified ENDING AT value in order to determine the end of the range to be read. This was necessary due to restrictions inherent in the underlying databases.

With Natural Version 4.1, these restrictions no longer apply, and the ENDING AT value can now be determined by the accessed databases themselves. This means that Natural is able to read the values only until including the specified ENDING AT value, but not beyond.

As this may lead to different results and so as not confuse the "old" end-value mechanism with the "new" one, a new keyword, TO, is provided for the specification of the database-controlled end value. The existing ENDING AT clause is not affected and continues to yield the same results as before.

The new keyword TO is available for Adabas, DB2, DL/I and VSAM databases.

## INPUT Statement

The INPUT statement provides the following enhancements:

- Selection Boxes
- Edit Mask Free Mode

### Selection Boxes

Natural Version 4.1 provides the possibility to attach selection boxes to input fields. These selection boxes are similar to those used in graphical user interfaces and are a comfortable alternative to help routines attached to fields.

To assign a selection box to a field, the INPUT statement provides the new field attribute SB. With SB, you specify the contents of the selection box, that is, the values, or the name of an array field that provides the values, to be displayed within the selection box. The size and position of the selection box are determined automatically (using the same algorithm as for help windows).

For a field for which the field attribute SB is specified, a selection-box indicator "V" is displayed next to the field. To invoke the selection box, the user positions the cursor on the "V" and presses the help key. The selection box is then displayed as a window on the screen. If the list of values within the selection box is longer than the selection box itself, the user can scroll by placing the cursor on the "More/Top/Bottom" lines of the selection box and pressing ENTER. To select a value from the selection box, the user positions the cursor on the desired value and presses ENTER. The selected value is then copied into the input field.

The field attribute SB is only available for alphanumeric fields.

## Edit Mask Free Mode

The edit mask free mode is an alternative INPUT mode for entering numeric fields with an edit mask use. When activated (either at session startup with the new profile parameter EMFM or in a running Natural session via the terminal command %FM+), all or some of the edit mask insert characters may be left out from input.

## PERFORM Statement

The PERFORM statement provides the following enhancements:

- **Notation "nX"** - see Optional Parameters.
- **Parameter Transfer with Dynamic Variables** - see Dynamic Variables.

## READ Statement

The READ statement provides the following enhancements:

- Dynamic Change of Reading Direction
- New Comparators
- Multi-Fetch
- End of Range Condition (ENDING AT) Controlled by Database
- WITH REPOSITION for Non-VSAM Databases

### Dynamic Change of Reading Direction

With Natural Version 3.1, the records to be retrieved by a READ statement could be read in ascending or descending sequence. This is determined by the keywords ASCENDING and DESCENDING in the **sequence/range-specification**. Also, the VARIABLE option allows you to determine the reading direction at runtime. However, once the READ statement is executed, you cannot change the reading direction.

With Natural Version 4.1, the new keyword DYNAMIC is provided for the **sequence/range-specification**: It enables you to change the reading direction from ascending to descending (or vice versa) within an active READ processing loop that is being executed, without having to restart the loop. After the keyword DYNAMIC, you specify a variable to which the value "A" (for "ascending") or "D" (for "descending") can be assigned. The DYNAMIC option is available for accesses to Adabas and DB2 databases.

### New Comparators

In addition to the field/value comparators EQUAL TO, STARTING FROM and ENDING AT, Natural Version 4.1 provides the possibility to specify start/end values with the following options:

- LESS THAN
- GREATER THAN
- LESS EQUAL
- GREATER EQUAL

These new comparators are available for accesses to Adabas, DB2, DL/I and VSAM databases.

### Multi-Fetch

Traditionally, Natural retrieves database records one by one. However, Adabas's Multi-Fetch functionality makes it possible to retrieve more than one database record per database access. To make use of this functionality, the READ statement provides a new MULTI-FETCH option. With this option, you are able to specify the number of records to be retrieved per database access when the statement is executed. The MULTI-FETCH option is available for

accesses to Adabas databases only. For database updates, the MULTI-FETCH option cannot be used.

MULTI-FETCH only affects the way in which the records are retrieved from the database. The program's record-processing logic is not affected; that is, the number of READ processing loops executed is the same as without MULTI-FETCH, and the records are still processed one by one.

### **End of Range Condition (ENDING AT) Controlled by Database**

With Natural Version 3.1, if the ENDING AT clause was used to limit the range of values to be read, Natural internally read one value beyond the specified ENDING AT value in order to determine the end of the range to be read. This was necessary due to restrictions inherent in the underlying databases.

With Natural Version 4.1, these restrictions no longer apply, and the ENDING AT value can now be determined by the accessed databases themselves. This means that Natural is able to read the values only until including the specified ENDING AT value, but not beyond.

As this may lead to different results and so as not confuse the "old" end-value mechanism with the "new" one, a new keyword, TO, is provided for the specification of the database-controlled end value. The existing ENDING AT clause is not affected and continues to yield the same results as before.

The new keyword TO is available for Adabas, DB2, DL/I and VSAM databases.

### **WITH REPOSITION for Non-VSAM Databases**

Due to the introduction of the new ESCAPE statement option TOP REPOSITION, the WITH REPOSITION option of the READ statement is no longer restricted to VSAM databases, but is also available for Adabas and DL/I databases.

### **SEND METHOD Statement**

The SEND METHOD statement provides the following enhancement:

- **Notation "nX"** - see Optional Parameters.

## **Size of Alphanumeric and Binary Variables**

With Natural Version 4.1, the maximum possible size of an alphanumeric variable (Format A) has been increased from 253 bytes to 1 GB. The maximum possible size of a binary variable (Format B) has been increased from 126 bytes to 1 GB.

## **Size of Data Elements**

With Natural Version 4.1, the maximum possible size of a single data element (array or indexed group) has been increased from 32 KB to 1 GB.

## Dynamic Variables

In addition to removing the size limitations for alphanumeric and binary variables (see [Size of Alphanumeric and Binary Variables](#)), Natural Version 4.1 makes it possible to allocate the length of such variables dynamically at runtime.

As the maximum size of large data structures (for example, pictures, sounds, videos) may not be known exactly at the time an application is developed, Natural provides for the definition of alphanumeric and binary variables with the attribute `DYNAMIC`. The value space of variables which are defined with this attribute is extended dynamically at runtime when it becomes necessary (for example, during an assignment operation: `#picture1 := #picture2`). This means that large binary and alphanumeric data structures may be processed in Natural without the programmer having to define a length at development time.

The new Natural system variable `*LENGTH` is provided to obtain the value space (number of bytes) currently used and available to the programmer for a given dynamic variable at runtime.

For performance optimization and also to avoid problems with too much or too little allocated memory space, the new statements `EXPAND`, `REDUCE` and `RESIZE` have been introduced. If the space allocated for a dynamic variable is no longer needed, the `REDUCE` or the `RESIZE` statement can be used to reduce or to resize the allocated space (to zero or any other desired size). If the upper limit of memory usage is known for a specific dynamic variable, the `EXPAND` statement can be used to set the space used for the dynamic variable to this specific size.

Dynamic variables can be used, for example, in `CALLNAT` or `PERFORM` statements. It is also possible to define and use arrays of dynamic variables.

See also [Large and Dynamic Variables and Fields in the Natural Statements documentation](#).

## Object Handles

With Natural Version 4.1, it is possible to define object handles within a global data area or as application-independent variables (AIVs).

## Optional Parameters

Natural Version 4.1 supports the use of optional parameters in subprograms, external subroutines and dialogs. Optional parameters may be used to expand an existing subprogram (for example, to provide additional parameters) without having to change all objects that use this subprogram.

An optional parameter is a field defined with the keyword `OPTIONAL` in the `DEFINE DATA PARAMETER` statement of an invoked object (subprogram, external subroutine or dialog). To such a field, a value can - but need not - be passed from an invoking object.

In the invoking statement (`CALLNAT`, `PERFORM` or `SEND METHOD`), the notation `nX` is used to indicate optional parameters for which no values are passed. With `nX` you specify that the next `n` parameters are to be skipped; that is, for the next `n` parameters no values are passed to the invoked object.

### Example:

Subprogram:	Invoking Object:
<code>DEFINE DATA PARAMETER</code>	<code>CALLNAT 'MY-SUB' #A #B #C #D #E</code>
<code>1 #P1 (A10)</code>	
<code>1 #P2 (A10) OPTIONAL</code>	or
<code>1 #P3 (A10)</code>	
<code>1 #P4 (A10) OPTIONAL</code>	<code>CALLNAT 'MY-SUB' #A 1X #C 2X</code>
<code>1 #P5 (A10) OPTIONAL</code>	or
<code>END-DEFINE</code>	
<code>...</code>	<code>CALLNAT 'MY-SUB' #A #B #C 1X #E</code>

To check in the invoked object whether or not an optional parameter has received a value from the invoking object, the new `SPECIFIED` option to be used in a logical condition is available.

## SPECIFIED Option in Logical Condition

With the new SPECIFIED option to be specified in a logical condition, you are able to check whether or not an optional parameter in an invoked object (subprogram, external subroutine or dialog) has received a value from the invoking object.

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.

### Example:

```
IF #OPTFIELD1 SPECIFIED THEN ... ELSE ...  
IF #OPTFIELD2 NOT SPECIFIED THEN ... ELSE ...
```

For a field not defined as OPTIONAL, the SPECIFIED condition will always be "TRUE".

## MASK Option in Logical Condition

With Natural Version 4.1, it is possible to check positions of a field for a date in Julian format. This will be particularly useful when a MASK option is used in conjunction with a MOVE EDITED statement that uses a Julian date in its edit mask.

See also the COMPOPT system command for enhancements related to the MASK option.

## System Variables Now Available under Natural for Mainframes

The following Natural system variables that have been available under Natural for Windows and UNIX are now also provided under Natural Version 4.1:

System Variable	Format	Default DISPLAY Header	Description of Contents
*CPU-TIME	I4	CPU-TIME	CPU time currently used by the Natural process in units of 10 ms. In environments where this variable is not supported, it contains the value 0.  For details, refer to the Natural Variables documentation.
*DATV	A11	DATV	Current date in the format <i>dd-mon-yyyy</i> (where <i>mon</i> is the leading three bytes of the month's name as in *DATG).
*DATVS	A9	DATVS	Current date in the format <i>ddmonyyyy</i> (similar to *DATV).
*HOSTNAME	A64	HOSTNAME	Name of the machine on which Natural is running.
*LENGTH( <i>field</i> )	I4	LENGTH	Currently used length (in bytes) of a <b>field</b> defined as a dynamic variable. See also Dynamic Variables.
*NATVERS	A8	NATVERS	Natural version excluding patch level information in the format <i>rr.vv.ss</i> , where <i>r</i> =release, <i>v</i> =version, <i>s</i> =system maintenance level (example: 04.01.01)
*PARAM-USER	A253	PARAM-USER	Name of the parameter module currently in use (contains blanks if PARAM= <i>name</i> has not been specified as a dynamic parameter).
*PATCH-LEVEL	A8	PATCH-LEVEL	Current Natural patch-level number as a string.
*PID	A32	PID	Unique identifier for the Natural session.

## New System Variables

The following Natural system variables are new as of Natural Version 4.1:

System Variable	Format	Default DISPLAY Header	Description
*LINE	I4	LINE	Contains the number of the line currently executed in a Natural object.
*TP	A8	TP	Contains the name of the TP subsystem under which Natural is running. This value is supplied by the operating system and may be subject to change.
*TPVERS	A8	TPVERS	Contains the version of the TP subsystem under which Natural is running. This value is supplied by the operating system and may be subject to change.

## Changed System Variable

### System Variable \*LANGUAGE

The value "0" of the system variable \*LANGUAGE which was tolerated in earlier Natural for Mainframes versions has been dropped to ensure compatibility with Natural for UNIX and Windows.

## More Precise Results for Floating Point Conversion

The format conversion for the transfer of data from literals (as in assignments, INIT or CONST clauses), alphanumeric fields (as in the VAL system function) or packed numeric fields to floating-point fields has been improved. This may in some cases lead to different results. However, these results will be of a greater precision than with Natural Version 3.1.

### Example:

F(F8) = 5.4E-79

Result with Version 3.1: +5.399999999999999E-79

Result with Version 4.1: +5.400000000000000E-79

## Precision of Floating Point Format Results of an Arithmetic Operation Improved

For arithmetic operations with a result of format/length F4 in previous Natural versions, format/length F8 is now being used to improve the precision of the results.

## Arithmetic Operations with Date and Time Enhanced

Multiplication and division are now allowed on intermediate results of additions and subtractions of formats D (date) and T (time).

## Evaluation of SQRT for (Un)Packed Number Enhanced

The evaluation of the mathematical system function SQRT for packed and unpacked numbers has been enhanced to improve performance and precision of the results.

## Substring Evaluation in MASK Corrected

The evaluation of a SUBSTRING clause in the MASK option has been corrected. This may lead to different results if the evaluated substring is shorter than the specified mask.

### Example:

```
IF SUBSTR(A10, 2, 3) = MASK(....) THEN
```

The length of the evaluated substring is 3, but the mask has length 4.

Result with Version 3.1: TRUE

Result with Version 4.1: FALSE

## New Compiler Options

The following new compiler options have been introduced:

MASKCME	The check for the mask character YYYY can be made compatible with MOVE EDITED.
NMOVE22	Assignment of numeric variables of same length and precision may be performed as with Natural Version 2.2.
TQMARK	The check for the translation of quotation mark.
V31COMP	Disable new syntax of Natural Version 4.1.

For further information, refer to system command COMPOPT.