

Programming Language

The following programming enhancements are provided with Natural 4.1:

- New Statements
 - Enhanced Statements
 - Dynamic Variables
 - Optional Parameters
 - SPECIFIED Option Logical Condition
 - MASK Option in Logical Condition
 - New System Variables
-

New Statements

The following new Natural statements will be available with Natural Version 4.1:

- EXPAND
- REDUCE

EXPAND Statement

The new statement EXPAND will be used to increase the allocated memory size 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 currently allocated for that dynamic variable, the EXPAND statement will have no effect.

For further information, see Dynamic Variables.

REDUCE Statement

The new statement REDUCE will be used to reduce the allocated memory size of a dynamic variable.

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

The allocated memory of the dynamic variable which is beyond the given size is released immediately when the statement is executed.

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 given size and the content of the variable is truncated (but not modified). If the specified size is larger than the currently allocated size of the dynamic variable, the REDUCE statement will be ignored.

For further information, see Dynamic Variables.

Enhanced Statements

The following Natural statements will be enhanced with Natural Version 4.1:

- CALL
- CALLNAT
- DEFINE DATA
- DEFINE WORK FILE
- ESCAPE

- FIND
- HISTOGRAM
- INPUT
- INTERFACE
- METHOD
- PERFORM
- PROPERTY
- READ
- SEND METHOD

CALL Statement

The CALL statement will provide the following enhancements:

- The limit of 32KB for the maximum length per parameter will be removed.
- A new option, INTERFACE4, will provide for enhanced parameter descriptions. Also, with this option the number of parameters to be passed to the invoked non-Natural program (currently 40) will no longer be restricted.

CALLNAT Statement

The CALLNAT statement will provide the following enhancements:

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

DEFINE DATA Statement

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

DYNAMIC	If you define a parameter as DYNAMIC, its length will be determined at runtime. For further information, see Dynamic Variables.
OPTIONAL	By default, parameter is defined without OPTIONAL, which means that a value <i>must</i> 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 WORK FILE Statement

The new option TYPE STREAM will allow you to specify that a work file is to be used in stream mode (instead of record-oriented mode).

ESCAPE Statement

The ESCAPE statement will provide the following enhancements:

- ESCAPE TOP REPOSITION
- ESCAPE OBJECT

ESCAPE TOP REPOSITION

This new option will allow 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 will immediately continue processing at the top of the active READ loop, using the current value of the search variable as new start value.

At the same time, ESCAPE TOP REPOSITION will reset 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 OBJECT

This new option will allow 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 OBJECT, 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 OBJECT statement will suffice to leave the programming object altogether.

ESCAPE OBJECT will only be relevant in *inline* subroutines. In external subroutines, subprograms and invoked programs, its would have the same effect as ESCAPE ROUTINE.

As with ESCAPE ROUTINE, the IMMEDIATE option to suppress loop-end processing will also be available with ESCAPE OBJECT.

FIND Statement

The FIND statement will provide 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 will provide a new MULTI-FETCH option. With this option, you will be able to specify the number of records to be retrieved per database access when the statement is executed. The MULTI-FETCH option will be 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 will not be affected; that is, the number of FIND processing loops executed will be the same as without MULTI-FETCH, and the records will still be processed one by one.

HISTOGRAM Statement

The HISTOGRAM statement will provide the following enhancements:

- Dynamic Change of Reading Direction
- New Comparators
- Multi-Fetch
- ENDING AT Controlled by Database

Dynamic Change of Reading Direction

With Natural 3.1, the database field values to be retrieved by a HISTOGRAM statement can 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 4.1, the new keyword DYNAMIC will be provided for the SEQUENCE clause: It will allow 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 will specify a variable to which the values "A" (for "ascending") or "D" (for "descending") can be assigned. The DYNAMIC option will be available for accesses to Adabas and DB2 databases.

New Comparators

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

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

These new comparators will be 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 will provide a new MULTI-FETCH option. With this option, you will be able to specify the number of records to be retrieved per database access when the statement is executed. The MULTI-FETCH option will be 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 will not be affected; that is, the number of HISTOGRAM processing loops executed will be the same as without MULTI-FETCH, and the records will still be processed one by one.

ENDING AT Controlled by Database

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

With Natural 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 will be 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, will be provided for the specification of the database-controlled end value. The existing ENDING AT clause will not be affected and will continue to yield the same results as before.

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

INPUT Statement

The INPUT statement will provide the following enhancement:

Selection Boxes

Natural 4.1 will provide 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 will provide 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 will be 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" will be 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 will then be 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 will then be copied into the input field.

The field attribute SB will only be available for alphanumeric fields.

INTERFACE Statement

The new option EXTERNAL will allow you to declare a NaturalX interface definition to be external.

METHOD Statement

The new ID clause will allow you to specify a dispatch ID for a NaturalX interface definition.

PERFORM Statement

The PERFORM statement will provide the following enhancements:

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

PROPERTY Statement

The new ID clause will allow you to specify a dispatch ID for a NaturalX interface definition.

READ Statement

The READ statement will provide the following enhancements:

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

Dynamic Change of Reading Direction

With Natural 3.1, the records to be retrieved by a READ statement can 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 4.1, the new keyword DYNAMIC will be provided for the *sequence/range-specification*: It will allow 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 will specify a variable to which the values "A" (for "ascending") or "D" (for "descending") can be assigned. The DYNAMIC option will be 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 4.1. will provide the possibility to specify start/end values with the following options:

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

These new comparators will be 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 will provide a new MULTI-FETCH option. With this option, you will be able to specify the number of records to be retrieved per database access when the statement is executed. The MULTI-FETCH option will be 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 will not be affected; that is, the number of READ processing loops executed will be the same as without MULTI-FETCH, and the records will still be processed one by one.

ENDING AT Controlled by Database

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

With Natural 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 will be 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, will be provided for the specification of the database-controlled end value. The existing ENDING AT clause will not be affected and will continue to yield the same results as before.

The new keyword TO will be 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 will no longer be restricted to VSAM databases, but will also be available for Adabas and DL/I databases.

SEND METHOD Statement

The SEND METHOD statement will provide the following enhancement:

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

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 will make 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 exactly be known 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 will be 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 having to define a length at development time.

The new Natural system variable *LENGTH will be provided to obtain the value space (number of bytes) currently used 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 and REDUCE will be introduced. If the space allocated for a dynamic variable is no longer needed, the REDUCE statement can be used to reduce 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 used space for the dynamic variable to this specific size.

Dynamic variables can be used, for example, in CALLNAT or PERFORM statements.

Optional Parameters

Natural Version 4.1 will support the use of optional parameters in subprograms, external subroutines and dialogs.

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.

For example:

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

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 will be available.

SPECIFIED Option in Logical Condition

With the new SPECIFIED option to be specified in a logical condition, you will be 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.

For 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 Version 4.1, it will be 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.

New System Variables

The following new Natural system variables will be available with Natural Version 4.1:

System Variable	Description
*CPU-TIME	Contains the CPU time used by the Natural process.
*DATV	Contains the current date in the format <i>dd-mon-yyyy</i> (where <i>mon</i> is the name of the month, abbreviated to 3 characters).
*DATVS	Contains the current date in the format <i>ddmonyyyy</i> (where <i>mon</i> is the name of the month, abbreviated to 3 characters).
*HOSTNAME	Contains the name of the machine on which Natural is running.
*LENGTH(<i>field</i>)	Contains the currently used length (in bytes) of a <i>field</i> defined as a dynamic variable. See also Dynamic Variables.
*NATVERS	Contains the Natural version number.
*PARM-USER	Contains the name of the parameter module currently in use.
*PATCH-LEVEL	Contains the Natural patch-level number.
*PID	Contains the current process ID.