Application Related System Variables

This chapter covers the following topics:

- *APPLIC-ID
- *APPLIC-NAME
- *COM
- *CONTROL
- *CONVID
- *COUNTER (r)
- *CPU-TIME
- *CURRENT-UNIT
- *DATA
- *DIALOG-ID
- *ERROR-LINE
- *ERROR-NR
- *ERROR-TA
- *ETID
- *EVENT
- *ISN (r)
- *LBOUND
- *LENGTH (field)
- *LEVEL
- *LIBRARY-ID
- *LINE
- *NUMBER (r)
- *OCCURRENCE
- *PAGE-EVENT

- *PAGE-LEVEL
- *PROGRAM
- *ROWCOUNT
- *STARTUP
- *STEPLIB
- *SUBROUTINE
- *THIS-OBJECT
- *TYPE
- *UBOUND

---

*APPLIC-ID

Format/length:	A8
Content modifiable:	No

This system variable contains the ID of the library to which the user is currently logged on.

*APPLIC-NAME

Format/length:	A32
Content modifiable:	No

Under Natural Security

If Natural Security is installed, this system variable contains the name of the library to which the user is logged on. If the user is logged on via a special link, it contains the link name instead. If Natural Security is not installed, this system variable contains the name `SYSTEM`.

The general option `Set *APPLIC-NAME always to library name` can be set so that `*APPLIC-NAME` always contains the library name, regardless of whether the user is logged on via a special link or not. See *Set *APPLIC-NAME always to library name* in the *Natural Security* documentation.

*COM

Format/length:	A128
Content modifiable:	Yes

This system variable designates a communication area which can be used to process data from outside a screen window.

Normally when a window is active, no data can be entered on the screen outside the window. However, if a map contains `*COM` as a modifiable field, that field will still be available for the user to enter data when a window is currently on the screen. Further processing can then be made dependent on the content of `*COM`. This allows you to implement user interfaces where a user can always enter data in the command line, even when a window with its own input fields is active.

Note:
Although `*COM` can be used as a modifiable field in an `INPUT` statement, it is *not* treated as an input field, but as a system variable; that is, any input entered into the `*COM` field will be taken as it is, without any input processing (e.g. conversion to upper case) being performed on it. Once `*COM` has been displayed on the screen via an `INPUT` statement, every subsequent `INPUT` or `REINPUT` statement will cause the current content of `*COM` to be displayed.

See also:

- *Dialog Design* in the *Programming Guide*
  - *Processing Data Outside an Active Window*
  - *Positioning the Cursor to *COM - the %T* Terminal Command*
  - *Copying Data from a Screen*

*CONTROL

Format/length:	Handle
Content modifiable:	No

This system variable contains the handle of the dialog element for which the current event has been triggered.

For details on events, see *Event-Driven Programming Techniques* in the *Programming Guide*.

*CONVID

Format/length:	I4
Content modifiable:	Yes

This system variable contains the conversation ID of the current conversational remote procedure call (RPC). This ID is set by an `OPEN CONVERSATION` statement.

Via an `OPEN CONVERSATION` statement, a client can get a server for exclusive use to execute a number of services (subprograms) within one server process. This exclusive use is called conversation. The `OPEN CONVERSATION` statement is used to open a conversation and specify the subprograms to be involved in this conversation. When an `OPEN CONVERSATION` statement is executed, it assigns a unique ID which identifies the conversation to the system variable `*CONVID`.

Several conversations can be open at the same time. To switch from one open conversation to another, you assign the corresponding conversation ID to *CONVID.

For further information on Natural RPC, see the *Natural Remote Procedure Call (RPC)* documentation.

*COUNTER (*r*)

Format/length:	P10
Content modifiable:	Yes

This system variable contains the number of times a processing loop initiated by a FIND, READ, HISTOGRAM or PARSE statement has been entered.

(*r*) notation after *COUNTER is used to indicate the statement label or source-code line number of the FIND, READ, HISTOGRAM or PARSE statement. If (*r*) is not specified, *COUNTER represents the number of times the currently active processing loop has been entered.

*COUNTER is not incremented if a record is rejected as a result of the criteria specified in a WHERE clause. *COUNTER is incremented if a record is rejected as a result of an ACCEPT/REJECT statement.

*CPU-TIME

Format/length:	I4
Content modifiable:	No

*CPU-TIME contains the CPU time currently used by the Natural process in units of 10 ms.

*CURRENT-UNIT

Format/length:	A32
Content modifiable:	No

This system variable contains the name of the currently executed unit. This is

- the function name in case of the object type "function",
- the inline subroutine name if an inline subroutine is performed,
- the external subroutine name in case of the object type "subroutine", see also *SUBROUTINE,
- the object name in case of all other object types (program, subprogram, map, dialog, etc.); see also *PROGRAM.

The contents of *CURRENT-UNIT will always be in upper case.

*DATA

Format/length:	N3
Content modifiable:	No

This system variable contains the number of data elements in the Natural stack which are available to the next `INPUT` statement as input data. `*DATA` will contain `0` when the stack is empty. A value of `-1` indicates the next element in the stack is a command or the name of a Natural transaction.

The settings of the Natural profile/session parameters `IA` (Input Assign Character) and `ID` (Input Delimiter Character) at the time of execution of the `STACK` statement are used to determine the `*DATA` value.

*DIALOG-ID

Format/length:	I4
Content modifiable:	No

This system variable contains the ID of the current instance of a dialog.

For details on dialogs, see *Event-Driven Programming Techniques* in the *Programming Guide*.

*ERROR-LINE

Format/length:	N4
Content modifiable:	No

This system variable contains the source-code line number of the statement that caused an error.

*ERROR-NR

Alternatively, you may specify `*ERROR`.

Format/length:	N7
Content modifiable:	Yes

This system variable contains the error number of the error which caused an `ON ERROR` condition to be entered.

Only error numbers in the range from 0 to 9999 are supported.

Normally, `*ERROR-NR` contains the Natural *system* error number which caused an error condition to be entered; however, when a `REINPUT WITH TEXT *`*`nnnn`* statement is executed, the *application-specific* message number *nnnn* is placed into `*ERROR-NR`.

You may modify the content of this system variable via a Natural program; however, not within an `ON ERROR` statement block.

`*ERROR-NR` is reset to `0` when another Level 1 program is executed.

*ERROR-TA

Format/length:	A8
Content modifiable:	Yes

This system variable contains the name of the error transaction program which is to receive control in the event of an error condition.

For further information, see *Using an Error Transaction Program* in the *Programming Guide*.

*ETID

Format/length:	A8
Content modifiable:	No

This system variable contains the current identifier of transaction data for Adabas. The default value is one of the following:

- the value of the Natural profile parameter `ETID`,
- the value from the security profile of the currently active user (applies only under Natural Security).

*EVENT

Format/length:	A32
Content modifiable:	No

This system variable contains the name of the current event.

For details on events, see *Event-Driven Programming Techniques* in the *Programming Guide*.

*ISN (*r*)

Format/length:	P10
Content modifiable:	Yes

This system variable contains the Adabas internal sequence number (ISN) of the record currently being processed within a processing loop initiated by a `FIND` or `READ` statement.

(r) notation after *ISN is used to indicate the label or statement number of the statement in which the FIND or READ was issued. If *(r)* is not specified, *ISN represents the ISN of the record currently being processed in the currently active processing loop.

For the HISTOGRAM statement, *ISN contains the number of the occurrence in which the descriptor value last read is contained (*ISN = 0 if the descriptor is not contained within a periodic group).

Database-Specific Information:

SQL Databases	*ISN cannot be used.
Tamino	*ISN contains the XML object ID.

*LBOUND

Format/length: I4

Content modifiable: No

*LBOUND contains the current lower boundary (index value) of an array for the specified dimension(s) (1, 2 or 3) or for all dimensions (asterisk (*) notation).

Syntax:

***LBOUND** (*operand1* [,*dim*])

Operand Definition Table:

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

operand1 is the array for which the lower boundary is specified. The index notation of the array is optional. As index notation only the complete range notation * is allowed for each dimension.

dim is the dimension number for which the current lower boundary is returned:

dim = { 1 | 2 | 3 | * }

If no dimension is specified, the lower bound of the first dimension is returned.

If 1, 2 or 3 is specified, the lower bound of the first, second or third dimension is returned.

If * is specified, the lower bound of all the defined dimensions are returned, that is

- 1 in case of an one dimensional array,
- 2 in case of a two dimensional array,
- 3 in case of three dimensional array.

If an X-array is not allocated and the lower bound of the specified dimension of this X-array is the variable index bound, that is, it is represented by an asterisk (*) character in the index definition, the lower bound of the specified dimension is undefined, and access to `*LBOUND` leads to a runtime error. In order to avoid the runtime error, `*OCCURRENCE` may be used to check against zero occurrences:

```
DEFINE DATA LOCAL
  1 #XA(A5/1:*)
END-DEFINE
IF *OCCURRENCE (#XA) NE 0 AND *LBOUND(#XA) > 10
  THEN ...
```

Examples:

```
DEFINE DATA LOCAL
  1 #I  (I4)
  1 #J  (I4/1:3)
  1 #XA (A5/10:*,20:*)
END-DEFINE
#I  := *LBOUND(#XA)       /* lower bound of 1st dimension is 10
#I  := *LBOUND(#XA,1)     /* lower bound of 1st dimension is 20
#I  := *LBOUND(#XA,2)     /* lower bound of 2nd dimension is 20
#J(1:2):= *LBOUND(#XA,*)  /* lower bound of all dimensions
                          /* (1st and 2nd)
                          /* #J(1) is 10 and #J(2) is 20
```

See also `*UBOUND` and `*OCCURRENCE`.

*LENGTH (*field*)

Format/length:	I4
Content modifiable:	No

This system variable returns the currently used length of a field defined as dynamic variable in terms of code units; for A and B format the size of one code unit is 1 byte and for U format the size of one code unit is 2 bytes (UTF-16). `*LENGTH(field)` applies to dynamic variables only.

See also *Value Space Currently Used for a Dynamic Variable* in the *Programming Guide*.

*LEVEL

Format/length:	N2
Content modifiable:	No

This system variable contains the level number of the program, subprogram, external subroutine, map, helproutine or dialog which is currently active. Level 1 is a main program.

`*LEVEL` does not apply to inline subroutines.

*LIBRARY-ID

Format/length:	A8
Content modifiable:	No

This system variable contains the current library ID (as specified by the user in the `LOGON` command).

This variable is the equivalent of the variable `*APPLIC-ID`.

*LINE

Format/length:	I4
Content modifiable:	No

It contains the number of the line currently executed in a Natural object.

*NUMBER (*r*)

Format/length:	P10
Content modifiable:	Yes

This system variable contains either the number of records which were selected as a result of a `FIND` statement (as a result of the `WITH` clause) or the number of values selected as a result of a `HISTOGRAM` statement.

`(`*`r`*`)` notation after `*NUMBER` is used to indicate the statement label or source-code line number of the `FIND` or `HISTOGRAM` statement. If `(`*`r`*`)` is not specified, `*NUMBER` represents the number of records selected from the `FIND` or `HISTOGRAM` used to initiate the currently active processing loop.

Note:
When the Adabas file accessed is protected by the Adabas facility Security By Value, `*NUMBER` will contain `9999999999`, if more than 1 record was found. If 1 record was found, `*NUMBER` will contain 1. If no record was found, `*NUMBER` will contain 0.

Database-Specific Information:

SQL Databases	For SQL databases, `*NUMBER` only contains the number of rows found, when used with a `FIND NUMBER` or with a `HISTOGRAM` statement without a `WHERE` clause. In all other cases, `*NUMBER` will not contain the number of rows found: `*NUMBER` will be `0` if no rows have been found; any value other than `0` indicates that rows have been found, but the value will have no relation to the number of rows actually found.
Tamino	When used with a `FIND NUMBER` statement without a `WHERE` clause, `*NUMBER` will contain the number of rows found. Otherwise, when applied to an XML database, `*NUMBER` will not contain the number of rows found: `*NUMBER` will be `0` if no rows have been found. Any value other than `0` indicates that rows have been found. However, the value will have no relation to the number of rows actually found. If a `FIND NUMBER` with a `WHERE` clause is used, the number of rows found is returned in `*COUNTER`.

*OCCURRENCE

Format/length: I4

Content modifiable: No

This system variable contains the current number of occurrences of an array for the specified dimension(s).

Syntax:

***OCCURRENCE** (*operand1* [,*dim*])

Operand Definition Table:

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

operand1 is the array for which the number of occurrences is returned. The index notation of the array is optional. As index notation only the complete range notation * is allowed for each dimension.

dim is the dimension number for which the current number of occurrences is returned:

dim = { 1 | 2 | 3 | * }

Explanation:

1	One-dimensional array. This is the default, if *dim* is not specified.
2	Two-dimensional array.
3	Three-dimensional array.
*	All dimensions defined for the corresponding array apply.

In a parameter data area, you can use the index notation `1:V` to define an array with a variable number of occurrences (see the `DEFINE DATA` statement). The current number of occurrences of such an array is determined at runtime. With `*OCCURRENCE`, you can ascertain the current number of array occurrences.

Examples:

```
DEFINE DATA
  PARAMETER
    1 #ARRAY (A5/1:V)
  LOCAL
    1 #I (I4)
    ...
END-DEFINE
...
FOR #I = 1 TO *OCCURRENCE(#ARRAY)
  ...
END-FOR
...
```

See also the example programs `OCC1P` and `OCC2P`.

Concerning X-arrays, `*OCCURRENCE` contains the current number of occurrences:

```
DEFINE DATA LOCAL
  1 #I   (I4)
  1 #J   (I4/1:3)
  1 #XA  (A5/1:*,1:*)
END-DEFINE
EXPAND ARRAY #XA TO (1:10,1:20)
#I := *OCCURRENCE(#XA)          /* number of occurrences of 1st dimension is 10
#I := *OCCURRENCE(#XA,1)        /* number of occurrences of 1st dimension is 10
#I := *OCCURRENCE(#XA,2)        /* number of occurrences of 2nd dimension is 20
#J(1:2) := *OCCURRENCE(#XA,*)   /* number of occurrences of all dimensions
                                /* (1st and 2nd)
                                /* #J(1) is 10 and #J(2) is 20
END
```

*PAGE-EVENT

Format/length:	U(dynamic)
Content modifiable:	No

This system variable contains the name of the current event delivered from Natural for Ajax.

It is used for rich GUI programming with the PROCESS PAGE statement. For further information, see the *Natural for Ajax* documentation.

*PAGE-LEVEL

Format/length:	I4
Content modifiable:	No

This system variable contains the level of the active PROCESS PAGE MODAL statement blocks.

If no PROCESS PAGE MODAL is active, the value of *PAGE-LEVEL is 0.

Note:
If the value of *PAGE-LEVEL is greater than 0, no output to Report 0 via an INPUT, PRINT, WRITE or DISPLAY statement is possible.

*PROGRAM

Format/length:	A8
Content modifiable:	No

This system variable contains the name of the Natural object that is currently being executed.

*ROWCOUNT

Format/length:	I4
Content modifiable:	No

This system variable contains the number of rows that were deleted, updated or inserted by one of the Natural SQL statements "searched" DELETE, "searched" UPDATE or INSERT (with *select-expression*) respectively. *ROWCOUNT always refers to the last executed one of these statements.

*STARTUP

Format/length:	A8
Content modifiable:	Yes

The program whose name is contained in this system variable will be executed whenever Natural would otherwise display the command input prompt (NEXT prompt or direct command line/window).

*STARTUP contains the name of the program which has been entered in Natural Security as startup transaction in the security profile of the respective library (except in batch mode; see also the *Natural Security* documentation).

If no startup transaction is specified or if Natural Security is not used, *STARTUP contains the value of the profile parameter STARTUP.

Via a Natural program, you can assign to *STARTUP a program name which will always overwrite its previous content.

Note:
A startup program used in batch mode must contain a FETCH or STACK COMMAND statement; otherwise error NAT9969 may occur.

If you invoke the command input prompt by entering the Natural terminal command %% (or any equivalent command) - either in a non-security environment or in a security environment in which command mode is not prohibited for the current library - the startup mechanism will be deactivated. To subsequently re-activate it, log on to the library again or execute a program which re-assigns the name of a program to *STARTUP.

Important:
To deactivate the program that is contained in *STARTUP, set the system variable to blank value, for example, by means of the statement RESET *STARTUP.

Under Natural Security:

In a Natural Security environment in which command mode is prohibited for the current library, %% will cause the program whose name is contained in *STARTUP to be invoked.

When a Natural runtime error occurs which is caused by a startup transaction (*STARTUP), Natural's error processing might lead to the startup transaction being executed again. This would cause an error-loop situation. To prevent such a loop, the general option Logoff in error case if *STARTUP is active is available. See *Logoff in Error Case if *STARTUP is Active* in the *Natural Security* documentation.

*STEPLIB

Format/length:	A8
Content modifiable:	No

This system variable contains the name of the steplib library which has been concatenated to the Natural library to which the user is currently logged on.

If Natural Security is not active, *STEPLIB contains the *STEPLIB name specified with the profile parameter STEPLIB in the parameter file used.

If Natural Security is active, the value may be defined in the security profile of a given library.

Note:
The database ID and file number of the *STEPLIB library are derived from its name. Apart from the library SYSTEM, libraries with the name SYS*xxx* are assumed to be in FNAT and other libraries are assumed to be in FUSER.

*SUBROUTINE

Format/length:	A32
Content modifiable:	No

This system variable contains the name of the external subroutine that is currently being executed. The contents of `*SUBROUTINE` will always be in upper case.

*THIS-OBJECT

Format/length:	HANDLE OF OBJECT
Content modifiable:	No

This system variable contains a handle to the currently active object. The currently active object uses `*THIS-OBJECT` to either execute its own methods or pass a reference to itself to another object.

`*THIS-OBJECT` only contains an actual value when a method is being executed. Otherwise it contains `NULL-HANDLE`.

*TYPE

Format/length:	A32
Content modifiable:	No

This system variable contains the type of the Natural object which is currently executed.

Valid values of `*TYPE`:

Value	Object Type
`PROGRAM`	Program
`FUNCTION`	Function
`SUBPROGRAM`	Subprogram
`SUBROUTINE`	Subroutine
`HELPROUTINE`	Helproutine
`MAP`	Map
`ADAPTER`	Adapter
`DIALOG`	Dialog

*UBOUND

Format/length:	I4
Content modifiable:	No

UBOUND contains the current upper boundary (index value) of an array for the specified dimension(s) (1, 2 or 3) or for all dimensions (notation).

Syntax:

***UBOUND** (*operand1* [,*dim*])

Operand Definition Table:

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

operand1 is the array for which the upper boundary is specified. The index notation of the array is optional. As index notation only the complete range notation * is allowed for each dimension.

dim is the dimension number for which the current upper boundary is returned:

dim = { 1 | 2 | 3 | * }

If no dimension is specified, the upper bound of the first dimension is returned.

If 1, 2 or 3 is specified, the upper bound of the first, second or third dimension is returned.

If * is specified, the upper bound of all the defined dimensions are returned, that is

- 1 in case of an one dimensional array,
- 2 in case of a two dimensional array,
- 3 in case of three dimensional array.

If an X-array is not allocated and the upper bound of the specified dimension of this X-array is the variable index bound, that is, it is represented by an asterisk (*) character in the index definition, the upper bound of the specified dimension is undefined, and access to *UBOUND leads to a runtime error. In order to avoid the runtime error, *OCCURRENCE may be used to check against zero occurrences:

```
DEFINE DATA LOCAL
  1 #XA(A5/1:*)
END-DEFINE
IF *OCCURRENCE (#XA) NE 0 AND *UBOUND(#XA) > 10
  THEN ...
```

Examples:

```
DEFINE DATA LOCAL
  1 #I  (I4)
  1 #J  (I4/1:3)
  1 #XA (A5/*:10,*:20)
END-DEFINE
#i  := *UBOUND(#XA)        /* upper bound of 1st dimension is 10
#i  := *UBOUND(#XA,1)      /* upper bound of 1st dimension is 10
#i  := *UBOUND(#XA,2)      /* upper bound of 2nd dimension is 20
#j(1:2):= *UBOUND(#XA,*)   /* upper bound of all dimensions
                           /* (1st and 2nd)
                           /* #J(1) is 10 and #J(2) is 20
```

See also *LBOUND and *OCCURRENCE.