

NATURAL Version 3.1.1

Release Notes for OpenVMS

Manual Order Number: NAT311-008VMS

This document applies to NATURAL Version 3.1.1 for OpenVMS, and to all subsequent releases.

Specifications contained herein are subject to change, and these changes will be reported in subsequent release notes or new manual editions.

© April 1998, SOFTWARE AG. All rights reserved.

SOFTWARE AG documentation often refers to numerous hardware and software products by their trade names. In most, if not all cases, these designations are claimed as trademarks or registered trademarks by their respective companies.

TABLE OF CONTENTS

1. GENERAL INFORMATION	1
Introduction	1
Prerequisite	1
Documentation	2
Example Library for New Features	2
Compatibility	3
Overview of Intended Incompatibilities	3
Known Problems	4
2. PROGRAMMING LANGUAGE	5
New Statements	6
DEFINE WORK FILE	6
Statements for NATURAL Remote Procedure Call	6
Statements for NaturalX	6
New System Variables	7
New Date System Variables	7
New System Function	8
SORTKEY	8
Enhanced Statements	9
CALLNAT and PERFORM	9
COMPRESS	10
COMPRESS and MOVE	12
DEFINE DATA	13
DIVIDE	16
EJECT	17
ESCAPE	17
FIND	18
FIND, GET, HISTOGRAM, READ and STORE	20
FIND and READ	20
HISTOGRAM and READ	20
MOVE	22

NATURAL Version 3.1.1 Release Notes for OpenVMS

PRINT	22
MASK Option (Logical Condition)	23
SUBSTRING Option (Various Statements)	23
Incomplete Statement Blocks (Various Statements)	23
Database Field Names (any Database Statement)	24
3. SYSTEM COMMANDS AND UTILITIES	25
New System Commands	26
COMPOPT — Compilation Options	26
DEBUG — Invoke Remote Debugging Utility	27
LASTMSG — Information on Last Error Situation	27
LIST COUNT — Count Number of Objects in Library	27
NATLOAD, NATUNLD, SYSUNLD — Unloading and Loading Utilities	27
REGISTER and UNREGISTER — NaturalX	28
Enhanced System Commands	28
CATALL	28
INPL	28
LIST	28
TECH	28
XREF	28
New SYSNCP Utility — Command Processor	29
New SYSUNLD Utilities — Unloading and Loading	30
New NATTERMCAP Utility — Terminal Capabilities	31
New Remote Debugging Utility	31
NATPARAM Utility	32
Revised User Interface	32
SYSDDM Utility	32
Generation of Null-Value Indicator Fields	32
SYSTRANS Utility	32

4. PROFILE PARAMETERS	33
File Numbers	33
New Profile Parameters	34
ACTPOLICY, AUTOREGISTER, COMSERVERID — NaturalX	34
CM — Suppress Command Mode	34
DBSHORT — Interpretation of Database Short Names	34
DFOUT — Date Format for Output	34
DFSTACK — Date Format for Stack	34
DFTITLE — Date Format for Default Report Title	34
ESCAPE — Disable Terminal Commands “%%” and “%.”	35
FCDP — Suppress Filler Character for Dynamically Protected Input Fields	35
GFID — Generation of Global Format Identifiers	35
KEY — Value Assignments to PA, PF and CLEAR Keys	35
MSGSF — Avoid Truncation of Message Texts in Windows	35
OPF — Suppress Overwriting of Protected Fields by Help routines	36
RDACTIVE, RDNODE and RDPORT — Remote Debugging	36
REINP — Automatic REINPUT for Invalid Data	36
V22COMP — Version 2.2 Compatibility	36
YSLW — Year Sliding Window	36
Parameters for Remote Procedure Call (RPC)	36
Enhanced Profile Parameters	37
DD	37
FDIC, FNAT, FSEC, FUSER, LFILE	37
TD	37
TF	37
USIZE	37
XREF	37
5. MISCELLANEOUS	39
Support of Year 2000	40
The “Year 2000” Problem	40
Handling Date Information with IS Option and VAL Function	40
Default Edit Mask for Date — The DF Parameter	40
Date System Variables	41
Date as Selection Criterion in Utilities	41

NATURAL Version 3.1.1 Release Notes for OpenVMS

“Sliding Window” — The YSLW Parameter	41
Date Format for Output — The DFOUT Parameter	42
Date Format for Stack — The DFSTACK Parameter	42
Date Format for Default Report Title — The DFTITLE Parameter	43
NATURAL Remote Procedure Call (RPC)	44
Programming Language Enhancements for Conversational RPCs	44
RPC Without Stub	45
Remote Directory	45
Remote Error Handling	46
Reduced Data Transfer Load	46
Logon Handling and Support of NATURAL SECURITY	46
Passing Floating-Point Parameters to/from Version 2.2	48
Sharing the FUSER System File Between Versions 2.2 and 3.1	48
ADABAS	48
SQL Support	48
Search Sequence for Objects to be Executed	49
Array Operations with Variable Index Ranges	50
Interception of Mismatching Array Ranges	50
Comparison and Assignment of Variable Array Ranges	50
Error Messages	51
Enhanced Message Texts	51
NAT1117 and NAT0924 Replaced by NAT0082	51
Assignment of Negative Numbers to Date/Time Intercepted	52
Suppressing of Zero Display for Time Fields	52
Assignments Between Numeric Variables of Same Length	52
Dump Generation with CATALL	53
Invoking NATURAL Subprograms from 3GL Programs	53
Invoking 3GL Programs using Version 2.1 Call Structure	53
6. NEW NATURAL-RELATED PRODUCTS	55
Natural@Web	55
NaturalX (Support of DCOM)	56

GENERAL INFORMATION

Introduction

These *Release Notes* inform you of the enhancements and new features that are provided with Version 3.1.1 of NATURAL for OpenVMS.

In addition to providing the enhancements and new features described in these *Release Notes*, NATURAL Version 3.1.1 also consolidates all error corrections, modifications and enhancements provided with previous patch-level releases of Version 2.2.3.

All enhancements and new features described in these *Release Notes* are fully documented in the NATURAL Version 3.1.1 documentation set.

Some of these enhancements lead to intentional minor incompatibilities between Version 2.2.3 and Version 3.1 (see page 3).

Prerequisite

NATURAL Version 3.1.1 requires Version 6.1 (or above) of OpenVMS on AXP or VAX.

Documentation

A completely revised set of NATURAL manuals is provided with the release of NATURAL Version 3.1.1 for OpenVMS.

The basic NATURAL Version 3.1.1 documentation set for OpenVMS consists of the following manuals:

- *NATURAL User's Guide for OpenVMS and UNIX*
- *NATURAL Programming Guide* (*)
- *NATURAL Reference Manual* (*)
- *NATURAL Statements Manual* (*)
- *NATURAL Debugging Manual*
- *NATURAL Installation and Operations Manual for OpenVMS and UNIX*

(*) These manuals also apply to Version 2.3.1 on mainframe computers and Version 3.1.1 on Windows NT and UNIX. They are also available in German.

The manuals are available on CD-ROM (the *Installation and Operations Manual* is also available in printed form).

Example Library for New Features

The library SYSEXV31 contains several example programs which illustrate some of the new features of NATURAL Version 3.1.

When you log on to library SYSEXV31 and then execute the program MENU, a menu will be displayed from which you can select the example programs.

Compatibility

Applications that were created with NATURAL Version 2.2.3 can be executed with Version 3.1 without any conversion procedure being required, and without your having to make any adjustments to the programs — except in the few cases of intentional minor incompatibilities listed below.

Overview of Intended Incompatibilities

The following list provides an overview of the intentional incompatibilities introduced with Version 3.1 (for details on each of the topics listed, refer to the pages indicated).

When a Version 2.2 application is executed with Version 3.1, these incompatibilities will cause the application to produce better, but slightly different, results. If in these cases you wish to get the same results as with Version 2.2, you have to adjust your applications accordingly.

- **DIVIDE** — with both **GIVING** and **REMAINDER**, different results for the **REMAINDER** field may occur (see page 16).
- **FIND** — the comparison logic for multiple-value fields in the **WITH** clause has been changed (see page 18).
- **FIND** — incorrect results for complex search condition with connected search criteria (see page 19).
- Incomplete statement blocks — no longer allowed (see page 23).
- **CATALL** — error NAT4867 issued instead of NAT0082 (see page 28).
- Interception of mismatching array ranges — incorrect results will lead to runtime error (see page 50).
- Assignment of negative numbers to date and time intercepted (see page 52).
- Comparison and assignment of variable array ranges — no longer allowed if an array range is actually a scalar (see page 50).
- Error messages NAT1117 and NAT0924 replaced by NAT0082 (see page 51).
- Assignments between numeric variables of the same length — different internal handling (see page 52).

Known Problems

For information on problems that are known to SOFTWARE AG, but have not yet been solved with this version of NATURAL, please refer to the section “Known Problems” in the README file supplied on the NATURAL installation tape.

PROGRAMMING LANGUAGE

This chapter contains information on:

- new statements,
- new system variables,
- new system function,
- enhanced statements.

New Statements

DEFINE WORK FILE

This statement can be used to assign a file name to a NATURAL work file.

Note: This statement was available with NATURAL Version 2.1.7, but not with Version 2.2.3. With Version 3.1, it now provides the same functionality again as with Version 2.1.7.

Statements for NATURAL Remote Procedure Call

The following statements are used in conjunction with the NATURAL remote procedure call:

- **CLOSE CONVERSATION** — This statement enables the client to close conversations. You can close a specific conversation, the current conversation (as identified by the new system variable *CONVID), or all active conversations.
- **OPEN CONVERSATION** — This statement enables the client to open a conversation and specify the remote subprograms to be included in the conversation.

For further information, see page 44.

Statements for NaturalX

The following statements are used in conjunction with NaturalX:

- **CREATE OBJECT**
- **DEFINE CLASS**
- **INTERFACE**
- **METHOD**
- **PROPERTY**
- **SEND METHOD**

For further information on NaturalX, see page 56.

New System Variables

The following new system variables are available:

System Variable	Function
*CONVID	Contains the conversation ID of a conversational RPC (see page 44).
*OCCURRENCE	Contains, at runtime, the actual number of occurrences of an array defined with a variable index range “(1:V)” in a parameter data area.
*ROWCOUNT	Contains the number of rows deleted, updated or inserted by the last NATURAL SQL statement (searched DELETE, searched UPDATE, or INSERT with <i>select-expression</i> respectively).

New Date System Variables

The following new date system variables are available: ***DAT4D**, ***DAT4E**, ***DAT4I**, ***DAT4J** and ***DAT4U**, all of which provide the year information as 4 digits. Otherwise their date representation corresponds to that of the date system variables ***DATD**, ***DATE**, ***DATI**, ***DATJ** and ***DATU** respectively.

New System Function

SORTKEY

Several national languages contain characters (or combinations of characters) which are not sorted in the correct alphabetical order by a sort program or database system, because the sequence of the characters in the character set used by the computer does not always correspond to the alphabetical order of the characters.

For example, the Spanish letter “CH” would be treated by a sort program or database system as two separate letters and sorted between “CG” and “CI” — although in the Spanish alphabet it is in fact a letter in its own right and belongs between “C” and “D”.

Or it may be that, contrary to your requirements, lower-case and upper-case letters are not treated equally in a sort sequence, that letters are sorted after numbers (although you may wish them to be sorted before numbers), or that special characters (for example, hyphens in double names) lead to an undesired sort sequence.

In such cases, you can use the new system function `SORTKEY(character-string)` to convert “incorrectly sorted” characters into other characters that are “correctly sorted” alphabetically by the sort program or database system. The values computed by `SORTKEY` would then only be used as sort criterion, while the original values would be used for the interaction with the end-user.

When you specify the `SORTKEY` function in a `NATURAL` program, the user exit `NATUSK nn` will be invoked (nn being the current language code as in the system variable `*LANGUAGE`). The *character-string* specified with `SORTKEY` will be passed to the user exit. The user exit has to be programmed so that it converts “incorrectly sorted” characters in this string into corresponding “correctly sorted” characters. The converted character string is then used in the `NATURAL` program for further processing.

The user exit is described in your *NATURAL Installation and Operations Manual*.

For further information on the `SORTKEY` function, see the *NATURAL Reference Manual*.

Enhanced Statements

CALLNAT and PERFORM

New Option AD=A

With Version 2.2, you can mark a parameter to be passed to a subprogram with AD=O (non-modifiable) or AD=M (modifiable).

With Version 3.1, you can also mark a CALLNAT parameter with AD=A (input-only): For remote subprograms executed via NATURAL remote procedure call (RPC) in a client/server environment, such a parameter will not be passed *to* the subprogram, but receive a value *from* the subprogram (see also page 46). If a subprogram is executed locally, the AD=A field will be reset to empty before the subprogram is invoked.

For subroutines, AD=A is also possible: A PERFORM parameter marked with AD=A will be reset before the subroutine is invoked and can be used to receive a value *from* the subroutine.

Internal Handling of AD=O

With Version 3.1, the internal handling of AD=O has changed. A CALLNAT/PERFORM parameter marked with AD=O is no longer passed to the subprogram/subroutine “by reference” (that is, via its address) but “by value”.

For details on the passing of parameters, see the CALLNAT and PERFORM descriptions in the *NATURAL Statements Manual*.

COMPRESS

The COMPRESS statement provides the following new options:

FULL	<p>With this option, the values of the source operands in their actual lengths — that is, including leading zeros and trailing blanks — are transferred to the target field.</p> <p>Without this option, leading zeros and trailing blanks are suppressed before the values are transferred.</p>
NUMERIC	<p>With this option, decimal points and signs in numeric source values are also transferred to the target field.</p> <p>Without this option, decimal points and signs are suppressed before the values are transferred.</p>
ALL	<p>This option can be used in conjunction with the option WITH DELIMITER(S):</p> <p>Without ALL, a delimiter is placed in the target field only between values actually transferred.</p> <p>With ALL, a delimiter is also placed in the target field for each blank value that is not actually transferred. This means that the number of delimiters in the target field corresponds to the number of source fields minus 1. This may be useful, for example, if the content of the target field is to be separated again with a subsequent SEPARATE statement.</p>
SUBSTRING	<p>This option, previously available in several other statements, is now also available in the COMPRESS statement for both the source fields and the target field. It allows you to transfer only parts of source fields and/or transfer them into a specific part of the target field.</p>

Example of FULL Option:

1. COMPRESS 'ABC ' 001 INTO #TARGET WITH DELIMITER '*'
Content of #TARGET is: **ABC*1**
2. COMPRESS **FULL** 'ABC ' 001 INTO #TARGET WITH DELIMITER '*'
Content of #TARGET is: **ABC *001**

Example of NUMERIC Option:

1. COMPRESS -123 1.23 INTO #TARGET WITH DELIMITER '*'
Content of #TARGET is: **123*123**
2. COMPRESS **NUMERIC** -123 1.23 INTO #TARGET WITH DELIMITER '*'
Content of #TARGET is: **-123*1.23**

Example of ALL Option:

1. COMPRESS 'A' ' ' 'C' ' ' INTO #TARGET WITH DELIMITER '*'
Content of #TARGET is: **A*C**
 2. COMPRESS 'A' ' ' 'C' ' ' INTO #TARGET WITH **ALL** DELIMITERS '*'
Content of #TARGET is: **A**C***
-

COMPRESS and MOVE

In order to support languages whose writing direction is from right to left, you can specify the option **PM=I** in the statements **COMPRESS** and **MOVE** so as to transfer the value of a source operand in inverse (right-to-left) direction.

Example 1:

```
MOVE 'XYZ' TO #A  
MOVE #A (PM=I) TO #B
```

Content of #B is: ZYX

Example 2:

```
MOVE 'XYZ' TO #A  
COMPRESS #A (PM=I) 'ABC' INTO #B
```

Content of #B is: ZYX ABC

DEFINE DATA

The DEFINE DATA statement provides the following new options:

- DEFINE DATA CONTEXT,
- DEFINE DATA OBJECT and HANDLE OF OBJECT,
- BY VALUE and BY VALUE RESULT (in DEFINE DATA PARAMETER).

Moreover, an enhancement has been implemented which affects:

- decimal digits of constant values.

DEFINE DATA CONTEXT

DEFINE DATA CONTEXT is used in conjunction with NATURAL remote procedure call (RPC). It is used to define variables that are to be available to multiple remote subprograms within one conversation, without having to explicitly pass the variables as parameters with the corresponding CALLNAT statements. See also page 44.

Only level-1 variables and redefinitions can be specified within DEFINE DATA CONTEXT; group and view definitions, however, are not possible. The variables can also be defined in a separate data area, that is, DEFINE DATA CONTEXT USING *local/parameter-data-area* is also possible.

DEFINE DATA OBJECT and HANDLE OF OBJECT

DEFINE DATA OBJECT and HANDLE OF OBJECT are used in conjunction with NaturalX. For further information on NaturalX, see page 56.

BY VALUE and BY VALUE RESULT (in DEFINE DATA PARAMETER)

With Version 2.2, parameters are passed to a subprogram/subroutine via their addresses (that is, by reference); therefore the format/length of a field specified as parameter in a CALLNAT/PERFORM statement have to be the same as the format/length of the corresponding field in the invoked subprogram/subroutine.

Version 3.1 provides the new option “BY VALUE” in the DEFINE DATA PARAMETER statement. With this option, you can pass parameters to a subprogram/subroutine *by value*; that is, the actual parameter values (instead of their addresses) are passed to the subprogram/subroutine. Consequently, the fields in the subprogram/subroutine need not have the same format/length as the CALLNAT/PERFORM parameters (their formats/lengths only have to be data transfer compatible).

With this new option you can, for example, increase the length of a field in a subprogram/subroutine (if this should become necessary due to an enhancement of the subprogram/subroutine) without your having to adjust any of the objects that invoke the subprogram/subroutine.

Example:

```
* Program
DEFINE DATA LOCAL
  1 #FIELDA (P5)
  ...
END-DEFINE
...
CALLNAT 'SUBP01' #FIELDA
...

* Subprogram SUBP01
DEFINE DATA PARAMETER
  1 #FIELDB (P9) BY VALUE
END-DEFINE
...
```

While the “BY VALUE” option applies to parameters being passed *to* a subprogram/subroutine, the new option “BY VALUE RESULT” causes parameters to be passed *by value* in both directions; that is, the actual parameter values are passed from the invoking object to the subprogram/subroutine and, on return to the invoking object, the actual parameter values are passed from the subprogram/subroutine back to the invoking object.

Decimal Digits of Constant Values

If the constant value specified after CONSTANT or INIT has more digits after the decimal point than the corresponding field, this does not lead to an error with Version 2.2. With Version 3.1, such inconsistency leads to error NAT0094 at compilation.

Example:

```
DEFINE DATA LOCAL
1 #FIELD (N2) INIT <12.25> /* no longer possible with Version 3.1
END-DEFINE
```

A compilation option (see page 26) is provided to allow you to temporarily continue to use the old Version 2.2 syntax.

DIVIDE

With Version 3.1, DIVIDE statements using both the GIVING and the REMAINDER option may in some cases — if the dividend (operand2) has more decimal positions than the result field — give different results for the REMAINDER field. However, these results will be of greater precision.

Note: This change will only affect programs that are newly compiled under Version 3.1. Programs compiled under Version 2.2 and executed under Version 3.1 will not be affected.

Example:

```
DEFINE DATA LOCAL
  1 #RESULT   (N2)
  1 #REMAIN   (N4.1)
END-DEFINE
*
DIVIDE 3 INTO 10.5 GIVING #RESULT REMAINDER #REMAIN
*
**                                     #RESULT #REMAIN
** VALUES WITH VERSION 2.2:  3          0.0
** VALUES WITH VERSION 3.1:  3          1.5
**
END
```

EJECT

To enhance the clarity of programs and avoid possible ambiguities in the source code, the keyword `LESS` in Syntax 2 of the `EJECT` statement is no longer optional, but required.

With Version 2.2, the shortest possible form is:

```
EJECT operand1
```

With Version 3.1, it is:

```
EJECT LESS operand1
```

ESCAPE

With Version 2.2, an `ESCAPE TOP` or `ESCAPE BOTTOM` statement within an `ON ERROR` statement block leads to an error at runtime. With Version 3.1, this invalid coding is already intercepted at compilation.

Moreover, it is no longer allowed to place an `ESCAPE TOP` statement within an `AT START OF DATA` statement block.

FIND

Comparison Logic for Multiple-Value Fields in WITH Clause

The comparison logic for multiple-value fields in the WITH clause of the FIND statement has been changed so as to be in line with the comparison logic in other statements (e.g. IF).

Four different forms of the FIND statement can be distinguished (the field MU in the following examples is assumed to be a multiple-value field):

1. `FIND XYZ-VIEW WITH MU = 'A'`

With Versions 2.2 and 3.1, this statement returns records in which *at least one occurrence* of MU has the value "A".

2. `FIND XYZ-VIEW WITH MU NOT EQUAL 'A'`

With Versions 2.2 and 3.1, this statement returns records in which *at least one occurrence* of MU does *not* have the value "A".

3. `FIND XYZ-VIEW WITH NOT MU NOT EQUAL 'A'`

With Version 2.2, this statement returns records in which *at least one occurrence* of MU has the value "A" (same as 1.).

With Version 3.1, this statement returns records in which *every occurrence* of MU has the value "A".

4. `FIND XYZ-VIEW WITH NOT MU = 'A'`

With Version 2.2, this statement returns records in which *at least one occurrence* of MU does *not* have the value "A" (same as 2.).

With Version 3.1, this statement returns records in which *none of the occurrences* of MU has the value "A".

This means that if in Version 3.1 you newly compile existing Version 2.2 programs containing FIND statements of the forms 3. and 4., they will return different results.

Should you in these cases wish to continue to get the same results as with Version 2.2, you have to change the statements as follows:

3. `FIND XYZ-VIEW WITH NOT MU NOT EQUAL 'A'` change to: `MU = 'A'`

4. `FIND XYZ-VIEW WITH NOT MU = 'A'` change to: `MU NOT EQUAL 'A'`

Complex Search Condition with Connected Search Criteria

With Version 2.2, a complex search condition which combines search criteria as follows

```
FIND ... WITH ((search-criterion1) OR (search-criterion2)) AND
              ((search-criterion3) OR (search-criterion4))
```

OR

```
FIND ... WITH ((search-criterion1) OR (search-criterion2)) AND
              ((search-criterion3) AND (search-criterion4))
```

leads to incorrect results if:

- *search-criterion1* and *search-criterion2* are both of the form:
field operator value
and at least one of the *operators* is **not** an EQUAL operator; and
- *search-criterion3* and *search-criterion4* are both of the form:
field = value
where *field* is the **same** field in both criteria.

With Version 3.1, this error has been corrected. This means that if in Version 3.1 you newly compile existing Version 2.2 programs containing such FIND statements, they may return different results.

Example:

```
DEFINE DATA LOCAL
  1 EMP-VIEW VIEW OF EMPLOYEES
  2 NAME
  2 BIRTH
END-DEFINE
*
FIND EMP-VIEW WITH ((NAME GT 'LYKOS') OR (NAME = 'BAILLET')) AND
                  ((BIRTH = 610116) OR (BIRTH = 490228))
...
```

With Version 2.2., the above FIND statement returns all employees whose names are greater than LYKOS — regardless of their date of birth —, as well as BAILLET (born 61–01–16).

With Version 3.1, the above FIND statement returns: MAUBERT (born 61–01–16) and BAILLET (born 61–01–16).

FIND, GET, HISTOGRAM, READ and STORE

With Version 2.2, it is possible in a non-NATURAL SECURITY environment in reporting mode to specify an ADABAS file number as *view-name*.

With Version 3.1, it is no longer possible to specify an ADABAS file number as *view-name*. This will lead to a compilation error (NAT0980).

FIND and READ

To support repositioning within database loops, the statements FIND and READ provide a new option “STARTING WITH ISN = *operand*”, which can be used for access to ADABAS databases.

This option may be used for repositioning in a FIND/READ loop whose processing has been interrupted, to easily determine the next record with which processing is to continue. This is particularly useful if the next record cannot be identified uniquely by any of its descriptor values. It can also be useful in a distributed client/server application where the reading of the records is performed by a server program while further processing of the records is performed by a client program, and the records are not processed all in one go, but in batches.

HISTOGRAM and READ

The statements HISTOGRAM and READ provide new options for reading records in descending sequence in order to support the “read backwards” feature of ADABAS and SQL databases.

Two options are possible:

- Static backward reading — by specifying the keyword DESCENDING.
- Variable forward/backward reading — by specifying the keyword VARIABLE followed by a variable (format/length A1) which determines the reading direction. The variable can contain the value “A” (for “ascending”) or “D” (for “descending”). This allows you to change the reading sequence from “ascending” to “descending”, and vice versa.

The default sequence is ascending. The new keyword ASCENDING is provided to allow you to explicitly specify ascending sequence.

Note for ADABAS:

For READ statements, the “read backwards” feature requires ADABAS Version 3.1 (or above) on Windows NT and UNIX, ADABAS Version 3.2 (or above) on OpenVMS, or ADABAS Version 6.1 (or above) on mainframes (for Version 6.1, the ZAPs distributed with early warnings ADA612–007 and ADA613–002 respectively have to be applied).

For HISTOGRAM statements, the “read backwards” feature requires ADABAS Version 3.1 (or above) on Windows NT and UNIX, ADABAS Version 3.2 (or above) on OpenVMS, or ADABAS Version 6.2 (or above) on mainframes.

Please note also that at present the NATURAL Statements Manual does not yet mention the appropriate ADABAS version for OpenVMS; this will be rectified with the next edition of the manual.

Example of DESCENDING Option:

```
READ EMPLOYEES IN DESCENDING SEQUENCE BY NAME = 'SMITH'
```

This statement returns all names in descending sequence, starting with the name “SMITH”.

Example of VARIABLE Option:

```
DEFINE DATA LOCAL
1 #DIRECTION (A1) INIT <'A'> /* 'A' = ASCENDING
1 #EMPVIEW VIEW OF EMPLOYEES
  2 NAME
  ...
END-DEFINE
...
IF *PF-KEY = 'PF7'
  THEN MOVE 'D' TO #DIRECTION
END-IF
READ #EMPVIEW IN VARIABLE #DIRECTION SEQUENCE BY NAME = 'SMITH'
  ...
END-READ
...
```

MOVE

With Version 2.2, if the value to be moved with MOVE RIGHT JUSTIFIED is longer than the target field, the value will be truncated on the *right-hand* side before being placed into the target field.

With Version 3.1, if the value to be moved with MOVE RIGHT JUSTIFIED is longer than the target field, the value (after the removal of trailing blanks) will be truncated on the *left-hand* side before being placed into the target field.

Example:

```
DEFINE DATA LOCAL
  1 #SOURCE (A6) INIT <' ABC  '> /* 1 leading blank and 2 trailing blanks
  1 #TARGET (A3)
END-DEFINE
*
MOVE RIGHT JUSTIFIED #SOURCE TO #TARGET
*
** CONTENTS OF #TARGET - WITH VERSION 2.2: ' AB'
**                      - WITH VERSION 3.1: 'ABC'
END
```

PRINT

It is no longer possible to specify the LS parameter with the PRINT statement (as it has no effect anyway). With Version 2.2, this does not lead to an error; with Version 3.1, it will lead to error NAT0394.

MASK Option (Logical Condition)

With the MASK option, you can specify the new character “/” (slash) to check if a value ends with a specific character or string of characters.

Example:

```
IF #FIELD = MASK ( * 'E' / )
```

This condition will be true if there is either an “E” in the last position of the field, or the last “E” in the field is followed by nothing but blanks.

SUBSTRING Option (Various Statements)

With Version 2.2, invalid or inconsistent values for the starting position and/or the length of the field portion in a SUBSTRING option lead to errors at runtime.

With Version 3.1, such invalid/inconsistent values in a SUBSTRING option are already intercepted at compilation (error NAT0471).

Incomplete Statement Blocks (Various Statements)

With Version 2.2, an empty statement block (for example, a FOR or REPEAT processing loop that does not contain any statements) may in some cases not lead to a compilation error.

With Version 3.1, this inconsistency has been corrected, and any empty — that is, syntactically incomplete — statement block will lead to an error at compilation. If you wish a statement block intentionally to perform no function, insert an IGNORE statement.

Database Field Names (any Database Statement)

With Version 2.2, two-character database field names are interpreted as field short names (as used by the underlying database system), whereas other database field names are interpreted as field long names (as defined for NATURAL in the corresponding DDM). For some database systems, this may lead to long names erroneously interpreted as short names.

With Version 3.1, it is possible to set an option with the new system command COMPOPT (see page 26) so that database field names will always be interpreted as long names, regardless of their length. This will avoid possible misinterpretations of database field names in programs.

SYSTEM COMMANDS AND UTILITIES

This chapter contains information on:

- new system commands,
- enhanced system commands,
- the new SYSNCP utility — command processor,
- the new SYSUNLD utilities — unloading and loading,
- the new NATTERMCAP utility — terminal capabilities,
- the new remote debugging utility,
- the NATPARAM utility,
- the SYSDDM utility,
- the SYSTRANS utility.

New System Commands

COMPOPT — Compilation Options

This new system command allows you to set various options which will affect the way in which a NATURAL object is compiled. The following options are available:

Database Short Field Names (DBSHORT)

This option determines how database field names are interpreted in NATURAL programs. Two options are possible:

- Database field names are considered long names (as defined in the corresponding DDM) — except 2-character field names, which are considered short names (as used by the underlying database system). This is the default.
- All database field names are considered long names, regardless of their length. This avoids possible misinterpretations of database field names in programs.

Note: It is also possible to set this option with the new profile parameter DBSHORT.

Global Format IDs (GFID)

This option allows you to control NATURAL's generation of global format IDs so as to influence ADABAS's performance for format buffer translations.

Note: It is also possible to set this option with the new profile parameter GFID.

Compatibility Option — Allow Old Version 2.2 Syntax (V22COMP)

The following inconsistent syntax construction, which is not intercepted by Version 2.2, leads to a syntax error with Version 3.1: DEFINE DATA — inconsistent number of decimal digits in constant value (see page 15).

To allow you a smooth transition from Version 2.2 to Version 3.1, the compatibility option of COMPOPT can be set: the above syntax construction will then *not* lead to a syntax error. Thus you will be able to compile your existing programs under Version 3.1 until you have adjusted them to the Version 3.1 requirements.

Notes:

It is also possible to set this option with the new profile parameter V22COMP.

The compatibility option (and the V22COMP parameter) will be available only for a limited period of time to allow you a smooth transition to the Version 3.1 syntax. It will be removed again with one of the next releases of NATURAL.

DEBUG — Invoke Remote Debugging Utility

With this new system command, you invoke the new remote debugging utility (see page 31).

LASTMSG — Information on Last Error Situation

With this new system command, you can display additional information about the error situation which has occurred last.

When NATURAL displays an error message, it may in some cases be that this error is not the actual error, but an error caused by another error (which in turn may have been caused by yet another error, etc.) In such cases, the LASTMSG command allows you to trace the issued error back to the error which has originally caused the error situation.

When you enter the LASTMSG command, you will get — for the error situation that has occurred last — the error message that has been displayed, as well as all preceding (not displayed) error messages that have led to this error.

When you select one of these messages, you will get the following information on the corresponding error: error number; number of the line in which the error occurred; name, type and level of the object that caused the error; name, database ID and file number of the library containing the object; error class (error issued by NATURAL or by user application); error type (runtime, syntax, command execution, session termination, program termination, remote procedure call); date and time at which the error occurred.

LIST COUNT — Count Number of Objects in Library

With this new system command, you can have the number of NATURAL objects in your current library listed. The following command options are available:

LIST COUNT	will display the total number of objects.
LIST COUNT *	will display the number of objects broken down by object types.
LIST COUNT name<	will display the number of objects whose names are less/equal <i>name</i> .
LIST COUNT name>	will display the number of objects whose names are greater/equal <i>name</i> .
LIST COUNT name*	will display the number of only those objects whose names begin with <i>name</i> .

NATLOAD, NATUNLD, SYSUNLD — Unloading and Loading Utilities

These new system commands are used to invoke the new utilities of the same names and the library in which they are contained; see page 30.

REGISTER and UNREGISTER — NaturalX

These two new system commands are used in conjunction with NaturalX. For information on NaturalX, see page 56.

Enhanced System Commands

CATALL

With Version 2.2, error NAT0082 is issued if no object is found that meets the CATALL selection criteria.

With Version 3.1, error NAT4867 is issued in this situation.

INPL

The INPL command has been revised entirely. It now provides the same functionality as on Windows NT, UNIX and mainframes.

Note: With Version 3.1, you can only use INPL to load datasets into the file system which are identified as official Software AG INPL system datasets.

For unloading and loading other datasets, you use the new SYSUNLD utilities (see page 30).

LIST

The LIST command provides the new option DIRECTORY, which allows you to display directory information about an object.

TECH

The TECH command has been enhanced to provide additional error information.

XREF

Apart from the existing values, you can specify the new value "DOC". XREF=DOC corresponds to XREF=FORCE, except that no cross-reference data will be generated.

New SYSNCP Utility — Command Processor

The NATURAL command processor is used to define and control navigation within an application. It consists of two parts:

- The utility SYSNCP. With this utility, you define commands (that is, combinations of keywords) and the actions to be performed in response to these commands. From your definitions, SYSNCP generates decision tables which determine what happens when a user enters a command.
- The statement PROCESS COMMAND. This statement is used to invoke the command processor within a NATURAL program. In the statement you specify the name of the processor to be used to handle the command input by a user at that point.

The benefits of the NATURAL command processor may be summarized as follows:

- **Less Coding** — Instead of having to repeatedly program lengthy and identically structured statement blocks to handle the processing of commands, you only have to specify a PROCESS COMMAND statement that invokes the command processor; the actual command handling need no longer be specified in the source code. This considerably reduces the amount of coding required.
- **More Efficient Command Handling** — As command handling is defined in a standardized way and in one central place, the work involved in creating and maintaining the command-processing part of an application can be done much faster and much more efficiently.
- **Improved Performance** — The NATURAL command processor has been designed with particular regard to performance aspects: it allows NATURAL to process commands as fast as possible and thus contributes to improving the performance of your NATURAL applications.

The various features of the NATURAL command processor and the functions of the SYSNCP utility are described in the chapter **Command Processor Maintenance** of the *NATURAL User's Guide for OpenVMS and UNIX*. The PROCESS COMMAND statement is described in the *NATURAL Statements Manual*.

New SYSUNLD Utilities — Unloading and Loading

The new library SYSUNLD contains two new utilities: NATUNLD and NATLOAD.

- NATUNLD is used to *unload* NATURAL programming objects, error messages and DDMs from system files onto a work file.
- NATLOAD is used to *load* NATURAL programming objects, error messages and DDMs from a work file into system files.

NATUNLD generates variable-length records with a maximum of 252 characters per record. These records are written onto NATURAL work file 1 in a format which can be used for loading with NATLOAD. NATLOAD reads the records from NATURAL work file 1. NATLOAD can only load work files created by NATUNLD.

In addition, NATUNLD allows you to write delete instructions for specific objects to the work file. When the work file is read with NATLOAD, these instructions cause the objects concerned to be deleted *from the target environment*.

The utilities NATUNLD and NATLOAD are described in the *NATURAL Installation and Operations Manual for OpenVMS and UNIX*.

New NATTERMCP Utility — Terminal Capabilities

The new utility NATTERMCP allows you to create, modify and test the terminal capabilities described in the terminal database SAGtermcap.

The NATTERMCP utility is described in the *NATURAL Installation and Operations Manual for OpenVMS and UNIX*.

New Remote Debugging Utility

NATURAL Version 3.1.1 provides a new utility which allows you to debug your NATURAL applications remotely.

You install this utility on a Windows NT 4.0 computer in your network, and then start your NATURAL on OpenVMS with the profile parameters RDACTIVE, RDNOTE and RDPORT. Then you can invoke the debugging utility from your OpenVMS computer with the NATURAL system command DEBUG.

The debugging utility provides a comfortable graphical user interface.

For further information see the *NATURAL Debugging Manual*.

NATPARM Utility

Revised User Interface

The user interface of the NATPARM utility has been revised to be more consistent and user-friendly.

SYSDDM Utility

Generation of Null-Value Indicator Fields

The SYSDDM utility has been enhanced to support the generation of null-value indicator fields for ADABAS files (see page 48).

SYSTRANS Utility

The SYSTRANS utility command provides the following enhancements:

- The use of SYSTRANS can now be controlled and restricted via NATURAL SECURITY.
- In addition to the existing objects, you can now also transfer command processors.
- Instead of making the selection of objects to be transferred by object types, you can now also transfer objects by library.
- Within SYSTRANS, you can now also invoke functions via direct commands.
- A new user exit allows you to invoke SYSTRANS functions from within your NATURAL applications.

PROFILE PARAMETERS

This chapter describes the changes concerning the NATURAL profile parameters. The following information is provided:

- file numbers,
- new profile parameters,
- enhanced profile parameters.

File Numbers

In accordance with the enhancements of ADABAS Version 3.1 for Windows NT and UNIX and Version 4.1 for OpenVMS, you can now specify values above 255 for file numbers (that is, file numbers can now be from 1 to 5000).

This applies to all profile parameters in which you can specify an ADABAS file number — with the following exception: if you use the “active cross-references” function of PREDICT Version 3.4 (or below), file numbers specified for FNAT and FUSER cannot be above 255 (due to restrictions inherent in that PREDICT version). Therefore it is at present not possible, to specify such values with the parameters FNAT and FUSER.

New Profile Parameters

The following new profile parameters are available with Version 3.1. The default values of these parameters are set so as to be compatible with the behaviour of Version 2.2.

ACTPOLICY, AUTOREGISTER, COMSERVERID — NaturalX

These new parameters are used in conjunction with NaturalX. For information on NaturalX, see page 56.

CM — Suppress Command Mode

This new parameter allows you to suppress the NATURAL command mode (MORE). As a result, the MORE line will be write-protected (no input possible).

DBSHORT — Interpretation of Database Short Names

This new parameter corresponds to an option of the new system command COMPOPT (see page 26).

DFOUT — Date Format for Output

This new parameter determines the format of date values for output; it is used in conjunction with year 2000 (see page 40).

DFSTACK — Date Format for Stack

This new parameter determines the format of date values placed on the stack; it is used in conjunction with year 2000 (see page 40).

DFTITLE — Date Format for Default Report Title

This new parameter determines the format of date values output in NATURAL default report titles as produced by DISPLAY, WRITE or PRINT statements; it is used in conjunction with year 2000 (see page 40).

ESCAPE — Disable Terminal Commands “%%” and “%. ”

This new parameter allows you to disable the terminal commands “%%” and “%. ”. These terminal commands will then be ignored; that is, it will not be possible to leave the currently active NATURAL program by entering “%%” or “%. ”.

FCDP —

Suppress Filler Character for Dynamically Protected Input Fields

This new parameter allows you to suppress the display of filler characters for input fields that have been made write-protected dynamically (that is, to which the attribute AD=P has been assigned via a control variable).

With Version 2.2, a dynamically protected input field is displayed filled with filler characters — which may suggest to the users that they could enter something in the field.

With Version 3.1, you can avoid this by setting FCDP=OFF: dynamically protected input fields will then be displayed filled with blanks instead of filler characters.

GFID — Generation of Global Format Identifiers

This new parameter corresponds to an option of the new system command COMPOPT (see page 26).

KEY — Value Assignments to PA, PF and CLEAR Keys

This new parameter allows you to assign values to the keys PA1 to PA3, PF1 to PF24 and to the CLEAR key on video terminals. The value assigned to each key can be a NATURAL system command or a user command (user program).

Assignments made with this parameter are only valid from within the NATURAL Direct Command window.

MSGSF — Avoid Truncation of Message Texts in Windows

By default, a NATURAL system error message consists of: the name of the program and the number of the line that caused the error, followed by the actual text of the message. Depending on the size of the window in which the message is displayed, the actual text may be truncated.

To avoid this truncation, the new parameter MSGSF can be set so that only the actual message text — without program name and line number — itself will be displayed.

OPF — Suppress Overwriting of Protected Fields by Helproutines

With Version 2.2, a helproutine assigned to a field can overwrite the field's content, even if the field is write-protected (AD=P).

With Version 3.1, the new parameter OPF allows you to suppress this, so that helproutines cannot overwrite the contents of write-protected fields.

RDACTIVE, RDNODE and RDPORT — Remote Debugging

These new parameters are used in conjunction with the new remote debugging utility (see page 31).

REINP — Automatic REINPUT for Invalid Data

By default, NATURAL automatically issues an internal REINPUT statement if invalid data have been entered.

With the new parameter REINP, you can switch this mechanism off. This allows you to handle such input errors yourself in your application.

V22COMP — Version 2.2 Compatibility

This new parameter corresponds to an option of the new system command COMPOPT (see page 26).

YSLW — Year Sliding Window

This new parameter is used to define a year sliding window; it is used in conjunction with year 2000 (see page 41).

Parameters for Remote Procedure Call (RPC)

In conjunction with RPC, several new profile parameters are available. These are grouped under "Remote Procedure Call"; see the *NATURAL Installation and Operations Manual for OpenVMS and UNIX* for details.

Enhanced Profile Parameters

DD

The possible range of values is now from -32767 to $+32767$.

FDIC, FNAT, FSEC, FUSER, LFILE

In addition to the existing values, you can specify the new option “Read Only” to disable modifications in the file. Thus it is possible to set read-only access for individual system files. (With Version 2.2, read-only access can only be set for *all* system files with the profile parameter ROSY.)

TD

With Version 2.2, you can specify the time differential in intervals of 30 minutes. With Version 3.1, you can specify it in intervals of 1 minute.

TF

For the *production-DBID* and/or *production-FNR*, you can specify an asterisk (*):

- If you specify it for both, all production DBIDs and FNRs will be translated to the specified *test-DBID* and *test-FNR*.
- If you specify it for the *production-FNR* only, all FNRs in the specified *production-DBID* will be translated to the specified *test-DBID* and *test-FNR*.

The test and production databases affected must be of the same database type.

USIZE

With Version 2.2, the possible range of values is 500 to 2048 KB.

With Version 3.1, the possible range of values is 1 to 1024 MB.

XREF

Apart from the existing values, you can specify the new value “DOC”. XREF=DOC corresponds to XREF=FORCE, except that no cross-reference data will be generated.

MISCELLANEOUS

This chapter contains information on:

- support of year 2000,
- NATURAL remote procedure call (RPC),
- ADABAS,
- SQL support,
- search sequence for objects to be executed,
- array operations with variable index ranges,
- error messages,
- assignment of negative numbers to date/time intercepted,
- suppressing of zero display for time fields,
- assignments between numeric variables of same length,
- dump generation with CATALL,
- invoking NATURAL subprograms from 3GL programs,
- invoking 3GL programs using Version 2.1 call structure.

Support of Year 2000

The “Year 2000” Problem

Numerous applications use a 2-digit format instead of a 4-digit format for the representation of year information. This means that, for example, 24th December 1996 is represented as “24-12-96”. This might lead to misinterpretations of dates in the next century, because 24th December 2000 would be represented as “24-12-00” — which could not be distinguished from the representation of 24th December 1900.

The following sections describe what is provided with NATURAL Version 3.1 to help solve this problem.

In addition, Chapter 5 of the *NATURAL Programming Guide* contains a new section on the processing of date information.

Handling Date Information with IS Option and VAL Function

With Version 2.2, the IS option, which is used to check whether the contents of an alphanumeric field can be converted to another format, can only be applied to format D if the value checked contains 2-digit year information. The mathematical function VAL, which is used to extract a numeric value from an alphanumeric field, only accepts date information that corresponds to the rules for the IS option; this means that a date value which contains 4-digit year information cannot be extracted.

With Version 3.1, the IS option and the VAL function accept both 2- and 4-digit year information.

Default Edit Mask for Date — The DF Parameter

If the value of a date field is converted to alphanumeric format (for example, in a MOVE, DISPLAY, PRINT, WRITE or INPUT statement) and no edit mask is specified for the conversion, the default date format as determined by the profile parameter DTFORM is used as edit mask, but only providing 2 digits for the year information. This means that even if the date value contained the century, this information would be lost during the conversion.

The same is true for the input validation of a date variable used in an INPUT statement. If no edit mask is specified, the input is validated according to the default date format determined by the DTFORM parameter, with 2 digits for the year information.

With Version 2.2, both the above effects can be avoided by explicitly specifying edit masks. However, a change of the format of alphanumeric date representations would then no longer be possible by simply changing the setting of the DTFORM parameter; instead, it would necessitate the adjustment of the specified edit masks in the programs.

With Version 3.1, the new session parameter DF allows you to specify whether the length of a date when converted to alphanumeric representation is to be 8 or 10 characters (and with 2- or 4-digit year information). The DF parameter can be specified with the FORMAT statement and various other statements (at statement and field level), and it is evaluated at compilation. This allows you to gradually change your applications to use 4-digit year representations and at the same time continue to make use of the flexibility provided by the DTFORM parameter.

Date System Variables

The NATURAL date system variables *DATD, *DATE, *DATI, *DATJ and *DATU, which contain the current date in various formats, all provide the year information as 2-digits.

With Version 3.1, corresponding new date system variables providing 4-digit year information are available (see page 7).

Date as Selection Criterion in Utilities

The NATURAL utilities NATUNLD, NATLOAD and SYSMAIN allow you to specify a date as selection criterion.

With Version 2.2, the date can only be specified with 2 digits for the year, and the current century is internally appended to the specified date.

With Version 3.1, the above utilities accept dates with both a 2- and a 4-digit year specification as selection criterion.

“Sliding Window” — The YSLW Parameter

With the new profile parameter YSLW, you can set a so-called “sliding window”, which allows you to continue using 2-digit year representations and at the same time ensure that any 2-digit year value can be uniquely related to a specific century.

Date Format for Output — The DFOUT Parameter

The new session/profile parameter DFOUT allows you to control the format in which date fields are output with INPUT, DISPLAY, PRINT and WRITE statements.

For date fields which are displayed with the above statements and for which neither an edit mask is specified nor a DF parameter applies, the DFOUT parameter determines the format in which the field values are displayed:

- with a 2-digit year component and delimiters,
- with a full 4-digit-year component and no delimiters.

The lengths of the date fields are not affected by the DFOUT setting, as either date value representation fits into an 8-byte field.

The DFOUT parameter can be set in the NATURAL parameter file, dynamically when NATURAL is invoked, or with the system command GLOBALS. It is evaluated at runtime.

Date Format for Stack — The DFSTACK Parameter

The new session/profile parameter DFSTACK allows you to control the format in which the values of date variables are placed on the stack (via a STACK, RUN or FETCH statement). Three options are possible:

- Date values are stacked with a full 4-digit year component and no delimiters.
- Date values are stacked with a 2-digit year component and delimiters.
- Date values are stacked with a 2-digit year component and delimiters; in addition, a change in the century will be intercepted: when a value is read from the stack, the century is either assumed to be the current one or determined by the YSLW parameter (see above); if this would lead to the century being different from that of the original date value, a runtime error will be issued (during the stacking process).

The DFSTACK parameter only applies to date fields for which no DF parameter is specified. It can be set in the NATURAL parameter file, dynamically when NATURAL is invoked, or with the system command GLOBALS. It is evaluated at runtime.

Date Format for Default Report Title — The DFTITLE Parameter

The new session/profile parameter DFTITLE allows you to control the format in which the date values is output in NATURAL default report titles as produced by DISPLAY, WRITE or PRINT statements. For details, see your *NATURAL Installation and Operations Manual*.

Note: At present, this parameter is only described in the NATURAL Installation and Operations Manual, but not in other parts of the NATURAL documentation where it ought to be described, too; this inconsistency will be rectified with the next edition of the NATURAL documentation.

NATURAL Remote Procedure Call (RPC)

Programming Language Enhancements for Conversational RPCs

With Version 3.1, conversational RPC support is provided for client/server communication (support of remote CALLNAT).

For conversational RPCs, the following enhancements to the NATURAL programming language are available:

- **OPEN CONVERSATION** — This new statement allows the client to get a server for exclusive use to execute a number of services (subprograms) within one server process. This exclusive use is called conversation. With **OPEN CONVERSATION**, you open a conversation and specify the names of the subprograms which are to be involved in this conversation. **OPEN CONVERSATION** will assign a unique ID that identifies the conversation to the system variable ***CONVID** (see below). Several conversations can be open at the same time. To switch from one open conversation to another, you assign the corresponding conversation ID to the ***CONVID**.
- **CLOSE CONVERSATION** — This new statement allows the client to close conversations. You can close the current conversation, one specific other open conversation, or all open conversations (see also page 6).
- **DEFINE DATA CONTEXT** — The new **CONTEXT** clause of the **DEFINE DATA** statement is used to define the data that are to be available to several subprograms within a conversation; see page 45 for details.
- ***CONVID** — This new system variable contains the conversation ID of a conversational RPC. The value of ***CONVID** is set by the **OPEN CONVERSATION** statement. NATURAL RPC will examine the current conversation for the subprogram to be executed remotely and will pass it to the appropriate server process.

DEFINE DATA CONTEXT

With Version 3.1, it is possible for several remote subprograms within one conversation to share the same data without having to explicitly pass these data as parameters. The fields to be available to the subprograms are defined as *context variables* in a DEFINE DATA CONTEXT statement in each subprogram in which they are to be available.

A context variable is referenced by its name, and its content is shared by all subprograms referring to that name within one conversation.

Each conversation has its own set of context variables. Context variables cannot be shared by different conversations.

The context variables will be reset to their initial values when an OPEN CONVERSATION statement is executed or a single (non-conversational) remote CALLNAT is performed.

Note: One of the great advantages of using NATURAL for client/server applications is that you can develop and test your applications locally, and then distribute them for production. This requires that subprograms using context variables behave the same way whether they be executed locally or remote. For remote conversations this is given, as one server may have only one open conversation at a time and consequently all context variables belong to this conversation; for local conversations, NATURAL precludes any confusion between context variables of the same names belonging to different conversations (by internally identifying the variables not only by their names but also by their respective conversation IDs).

RPC Without Stub

With Version 2.2, you have to generate a stub for every subprogram to be executed remotely.

With Version 3.1, if a stub for a remote subprogram does not exist, NATURAL RPC will automatically generate the necessary data which would normally be supplied by the stub — and invoke the remote subprogram as if a stub existed for it. This means that you no longer need to generate stubs for remote subprograms.

Nonetheless it is possible to continue to use existing stubs.

Remote Directory

With Version 2.2, the locations (server addresses) of the remote subprograms have to be defined in a local directory for each RPC client application. For multiple clients using the same remote subprograms this means that identical directory information appears — and has to be maintained — on every single client.

With Version 3.1, you can define one central remote directory on the server; this remote directory can be accessed by all clients. This drastically reduces the maintenance effort of directory information. Moreover, the remote directory provides a single central place of reference to all services available in your client/server environment.

The remote directory server is implemented as a NATURAL subprogram. A sample of this subprogram, named RDSSCDIR, is provided in the library SYSRPC. It reads the required directory information from a work file. The interface of the subprogram is documented so that you can develop your own remote directory service.

If a remote subprogram is not found in the local directory, it will be sought in the remote server directory (by executing an internal remote CALLNAT). An internal directory cache minimizes the access to the remote directory. The cache information is controlled by an expiration date set by the remote directory server.

Remote Error Handling

With Version 2.2, NATURAL errors that occur in remote subprograms on the server are handled directly by the error transaction at level 1 of the application, to which the actual error information is not always available.

With Version 3.1, errors that occur in remote subprograms on the server are handled in the same way as errors in subprograms invoked locally; that is, the same ON ERROR handling mechanisms apply.

Reduced Data Transfer Load

With Version 3.1, the AD= specifications of CALLNAT parameters (see also page 9) are also evaluated for remote subprograms in a client/server environment:

- AD=M parameters are passed from the client to the server and back again.
- AD=O parameters are only passed from the client to the server, but not back.
- AD=A parameters are only passed from the server back to the client.

This reduces the load of data to be sent.

Logon Handling and Support of NATURAL SECURITY

With Version 2.2, a remote subprogram will always be sought in the current library on the server; however, the client has no control over which library is the current one on the server when the remote CALLNAT is performed.

With Version 3.1, NATURAL RPC also supports NATURAL SECURITY in client/server environments, and at the same time allows clients to set the current library on a server.

NATURAL RPC's Server Maintenance on the client provides a new option "Logon", which can be set for individual servers, or for a node so as to apply to all servers belonging to that node.

If the "Logon" option is set to "yes" and a remote CALLNAT is performed, the user's ID and password as well as the current library ID from the client will be passed to the server along with the CALLNAT request.

The user ID and password are established as follows:

- If the client runs under NATURAL SECURITY, the user ID and password from the NATURAL SECURITY logon on the client will be encrypted into a security token and passed to the server.
- For non-NATURAL SECURITY clients, a user exit will be provided which the user has to execute and which will prompt the user to specify a user ID and password — which will then be passed to the server.

The server will verify the user ID and password; and before executing the requested subprogram, NATURAL will then perform a logon on the server using the current library ID from the client.

If the server runs under NATURAL SECURITY, the user ID and password from the client will be verified against the corresponding user security profile on the server, and the logon to the requested library and the execution of the subprogram will be performed according to the corresponding NATURAL SECURITY library and user profile definitions on the server.

After the execution of the subprogram, the library used before the CALLNAT request will be made current again on the server.

In the case of a conversational RPC (see also page 44), the first CALLNAT request within the conversation will set the library ID on the server; and the CLOSE CONVERSATION statement will reset the library ID on the server to the one before the first request of the conversation.

For compatibility, Version 3.1 servers still support remote CALLNATs from clients where the "Logon" option is not set.

To enforce the "Logon" option — that is, if you want a server to accept only requests from clients where the "Logon" option *is* set — the new profile parameter LOGONRQ is provided, which will be evaluated when you start the server.

Passing Floating-Point Parameters to/from Version 2.2

If floating-point parameters are passed in a remote procedure call to/from a partner (client or server) whose NATURAL version is 2.2, the profile/session parameter DC in this Version 2.2 NATURAL must be set to “.” (period); otherwise, conversion errors will occur.

Sharing the FUSER System File Between Versions 2.2 and 3.1

If you use an existing FUSER system file for both NATURAL Versions 2.2 and 3.1, and any of the user libraries on that FUSER system file contains a Version 2.2 program NATCLT3, you have to move this program to the library SYSTEM on the FUSER system file. Otherwise, NATURAL Version 3.1 RPC will not function correctly.

ADABAS

The following ADABAS features are supported with NATURAL Version 3.1:

- Support of null values in analogy to SQL databases: for database fields that can contain null values, corresponding null-value indicator fields are generated in the DDM. These null-value indicator fields are evaluated when a FIND or READ statement is executed, and can be queried in a FIND statement.
- Support of the “read backwards” feature (see also page 20).
- Support of repositioning within database loops (see also page 20).
- The maximum number of occurrences of a periodic group has been increased from 99 to 191.
- Support of the “security by value” feature.
- Support of file numbers from 1 to 5000.

SQL Support

NATURAL Version 3.1 for OpenVMS provides access to SQL databases via ENTIRE ACCESS.

When ENTIRE ACCESS for OpenVMS becomes available, you can use the NATURAL SQL statements (which are described in the *NATURAL Statements Manual*) to access SQL databases from your NATURAL applications on OpenVMS.

Search Sequence for Objects to be Executed

Search for Objects to be Executed from User Libraries

The sequence in which user-written NATURAL objects that are to be executed from the FUSER system file are searched for, has been enhanced: if an object cannot be found on the FUSER file, the library SYSTEM on the system file FNAT is also searched for it. The search sequence is:

- ① current library (as defined by system variable *LIBRARY-ID),
- ② steplib (in sequence as specified in NATURAL SECURITY profile of current library),
- ③ default steplib (as defined by system variable *STEPLIB),
- ④ library SYSTEM on FUSER system file,
- ⑤ library SYSTEM on FNAT system file (← *new!*).

Search for Objects to be Executed from System Libraries

The sequence in which NATURAL objects that are to be executed from the FNAT system file are searched for, has been enhanced: if an object cannot be found on the FNAT file, the library SYSTEM on the system file FUSER is also searched for it. The search sequence is:

- ① current “SYS” library (as defined by system variable *LIBRARY-ID),
- ② steplib (in sequence as specified in NATURAL SECURITY profile of current library),
- ③ default steplib (as defined by system variable *STEPLIB),
- ④ library SYSTEM on FNAT system file,
- ⑤ library SYSTEM on FUSER system file (← *new!*).

Thus, it is possible to make user exits generally available without having to keep copies of them on both system files. It will be sufficient to provide them in one location, namely on the FUSER system file.

Array Operations with Variable Index Ranges

With Version 3.1, the handling at runtime of operations involving arrays with variable index ranges has been improved to avoid incorrect/inconsistent results. This affects the following:

Interception of Mismatching Array Ranges

With Version 2.2, some array operations using variable index ranges may lead to incorrect results if the array ranges do not match at runtime.

With Version 3.1, once the actual values are assigned to the variables in the index at runtime and the actual index ranges are thus determined, the actual range is compared with the syntax rules for *constant* index ranges: if the same construction using constant index ranges would lead to an error at compilation, an error will be issued at runtime (NAT1317).

Consequently, possible incorrect results that may have gone unnoticed with Version 2.2 will be intercepted by a runtime error with Version 3.1.

Comparison and Assignment of Variable Array Ranges

With Version 3.1, a comparison or assignment involving arrays with variable indexes will lead to an error at runtime (NAT1317) if an array range turns out to be actually a scalar once the actual values are assigned to the index variables.

With Version 2.2, such a comparison or assignment is allowed, but it is not consistent with the handling of constant scalars (as shown in the following example).

Example:

Version 2.2:

1. IF #A(i:j) = #B(m) is resolved as: IF #A(i) = #B(m) OR #A(j) = #B(m)
2. IF #A(i:j) = #B(m:n) is resolved as: IF #A(i) = #B(m) AND #A(j) = #B(n)

This means that if the values of “m” and “n” are equal, comparison **2.** is resolved inconsistently.

Version 3.1:

If the values of “m” and “n” are equal, comparison **2.** will cause a runtime error.

Error Messages

Enhanced Message Texts

Descriptions of various error messages have been extended to provide a more precise indication as to why an error condition has occurred.

Example:Version 2.2:

```
NAT0082 Invalid command, or object does not exist in library.
```

Version 3.1:

```
NAT0082 Invalid command, or object-type object-name does not exist in library.
```

```
NAT0082 Invalid command, or subprogram XYZ does not exist in library.
```

NAT1117 and NAT0924 Replaced by NAT0082

In situations where Version 2.2 displays error message NAT1117 (requested map not available) or NAT0924 (subroutine, GDA or external report not found), Version 3.1 displays message NAT0082. This will lead to different results if you interrogate these message numbers in your applications.

Assignment of Negative Numbers to Date/Time Intercepted

It is not allowed to assign a negative value to a date field (format D) or a time field (format T).

With Version 2.2, however, such invalid assignment at runtime may in some cases not be intercepted.

With Version 3.1, this has been corrected: the assignment of a negative value to a date or time field will always lead to an error (NAT1319).

Suppressing of Zero Display for Time Fields

With Version 2.2, the profile/session parameter ZP, which can be used to suppress the display of field values that consist of all zeros, applies only to numeric fields (formats N, P, I and F).

With Version 3.1, the ZP parameter also applies to time fields (format T).

Assignments Between Numeric Variables of Same Length

The internal handling of assignments of a value from one variable with format N to another variable with format N — where both variables have the same length — has been changed (so as to be consistent with the internal handling of assignments between format N variables of different lengths). When these variables are redefined, this may in some cases lead to different results.

A different result will also occur if the first three bytes of the content of the system variable *CURSOR represent a negative number and the content of *CURSOR is assigned to a user-defined variable of format/length N6.

Dump Generation with CATALL

With Version 3.1, the profile/session parameter DU=ON (dump generation) also takes effect when the system command CATALL is executed.

Invoking NATURAL Subprograms from 3GL Programs

With Version 3.1, it is possible to invoke NATURAL subprograms from a program written in a third-generation programming language (3GL).

For details, see the description in the file READMEMORY3GL.TXT in the directory NATDIR:[V311.samples.sysexuex].

Invoking 3GL Programs using Version 2.1 Call Structure

With Version 3.1, it is possible to invoke a program written in a third-generation programming language (3GL) from a NATURAL program via the call interface MYAPI, using a call structure that corresponds to the one used in Version 2.1. This means that NATURAL programs invoking 3GL programs need not be modified.

For details about the call interface, see the description in the file READMEMORYAPI.TXT in the directory NATDIR:[V311.samples.sysexuex].

NEW NATURAL-RELATED PRODUCTS

Together with NATURAL Version 3.1.1, two new SOFTWARE AG products are available:

- Natural@Web,
- NaturalX.

Natural@Web

With NATURAL 3.1.1 and Natural@Web, you can create web pages via a NATURAL subprogram.

This enables you to:

- return dynamic web pages generated by NATURAL subprograms,
- access the HTTP interface of your HTTP server (cookies),
- return different kinds of documents containing alphanumeric data,
- use predefined programs for HTML generation.

NATURAL provides three new libraries containing the NATURAL web interface, which is called by the EntireX web adapter.

For details on Natural@Web, please refer to the Natural@Web documentation.

NaturalX (Support of DCOM)

With NATURAL 3.1.1 and NaturalX, you can write distributed object-based applications and distribute them with DCOM (distributed component object model).

This enables you to:

- allow your components to be accessed by other object-oriented components,
- execute these components on local or remote servers,
- access object-oriented components written in a variety of programming languages across process and machine boundaries from within NATURAL programs,
- wrap existing NATURAL applications into object-oriented components.

The following concepts have been introduced into NATURAL: classes, objects, interfaces, methods, and properties.

To integrate the new concepts smoothly into NATURAL, existing NATURAL concepts are used. Existing object types, like local data area and subprogram, are used in new contexts. One new NATURAL object type is introduced: the class.

The NATURAL programming language has been extended to include object-oriented instructions. For this purpose, the following new statements are available: CREATE OBJECT, DEFINE CLASS, SEND METHOD, INTERFACE, METHOD and PROPERTY.

The new system commands REGISTER and UNREGISTER, as well as the new profile parameters ACTPOLICY, AUTOREGISTER and COMSERVERID are available in conjunction with NaturalX.

For details on NaturalX, please refer to the NaturalX documentation.

Note: With NATURAL Version 3.1.1, NaturalX classes can only be used in a local NATURAL session. The distribution of NaturalX applications via DCOM will be made available with one of the next releases.

NATURAL Version 3.1.1 Release Notes for OpenVMS

NATURAL Version 3.1.1 Release Notes for OpenVMS