

COMPOPT

The COMPOPT command is available on all platforms, however, there are platform-specific differences.

- COMPOPT for Mainframes
 - COMPOPT for Windows and UNIX
-

COMPOPT for Mainframes

```
COMPOPT [option=value]
```

This system command is used to set various compilation options. The options are evaluated when a Natural programming object is compiled.

option=value

The keywords for the individual options are shown on the Compilation Options screen and are described in the section Options.

The settings assigned to a compiler option are in effect until you issue the next LOGON command to another library. At LOGON, the default settings set with the macro NTCMPO and/or profile parameter CMPO will be resumed.

This section covers the following topics:

- General Information on Compiler Options
- Options

General Information on Compiler Options

You can specify compiler parameters on different levels:

- The default settings of the individual parameters are set with the macro NTCMPO in the Natural parameter module NATPARAM.
- At session start, you can override the compiler parameters with the profile parameter CMPO.
- During an active Natural session, there are two ways to change the compiler parameters with the COMPOPT command: either directly using command assignment (COMPOPT *option=value*) or by issuing the COMPOPT command without parameters which displays the Compilation Options screen. For further information, see the section Options. The settings assigned to a compiler option are in effect until you issue the next LOGON command to another library. At LOGON, the default settings set with the macro NTCMPO and/or the profile parameter CMPO (see above) will be resumed.

Example:

```
COMPOPT DBSHORT=ON
```

- In a Natural programming object (for example: program, subprogram), you can set compiler parameters with the OPTIONS statement.

Example:

```
0010 OPTIONS KCHECK=ON
0020 WRITE 'Hello World'
0030 END
```

The compiler options defined in an **OPTIONS** statement will only affect the compilation of this programming object, but do not update settings set with the command **COMPOPT**.

Options

If you issue the **COMPOPT** command without parameters, the Compilations Options screen appears.

- **KCHECK** - Keyword Checking
- **PCHECK** - Parameter Checking for **CALLNAT** Statements
- **DBSHORT** - Interpretation of Database Short Field Names
- **PSIGNF** - Internal Representation of Positive Sign of Packed Numbers
- **TSENABL** - Applicability of TS Profile Parameter
- **GFID** - Generation of Global Format IDs
- **LOWSRCE** - Allow Lower-Case Source
- **TQMARK** - Translate Quotation Mark

Version Compatibility Options:

- **FINDMUN** - Detect Inconsistent Comparison Logic in **FIND** Statements
- **MASKCME** - **MASK** Compatible with **MOVE EDITED**
- **NMOVE22** - **MOVE** Assignment like **NAT22**
- **V31COMP** - Disable New Version 4.1 Syntax

KCHECK - Keyword Checking

ON	Programming objects will be checked for Natural statement keywords. Variable names which are reserved Natural keywords are rejected. The section Keywords and Reserved Words in the Natural Programming Guide contains a list of all Natural keywords and reserved words, in which the statement keywords affected by this option are marked.
OFF	No keyword check is performed. This is the default.

PCHECK - Parameter Checking for **CALLNAT** Statements

ON	<p>The compiler checks the number, format, length and array index bounds of the parameters that are specified in a CALLNAT statement. Also, the OPTIONAL feature of the DEFINE DATA PARAMETER statement is considered in the parameter check.</p> <p>The parameter check is based on a comparison of the CALLNAT parameters with the DEFINE DATA PARAMETER definitions in the subprogram to be invoked.</p> <p>It requires that</p> <ul style="list-style-type: none"> ● the name of the subprogram to be invoked is defined as a numeric constant (not as an alphanumeric variable), ● the subprogram to be invoked is available as a cataloged object. <p>Otherwise, PCHECK=ON will have no effect.</p>
OFF	No parameter check is performed. This is the default.

DBSHORT - Interpretation of Database Short Field Names

ON	<p>Database field names in programming objects 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.</p> <p>Note: Even if DBSHORT=ON, the use of a short database field name is not allowed in the following cases:</p> <ul style="list-style-type: none"> ● if DEFINE DATA LOCAL is specified in a program, ● in Natural for non-mainframe platforms, ● when Natural Security is installed.
OFF	<p>All database field names in programming objects are considered long names, regardless of their length. This avoids possible misinterpretations of database field names in programs.</p> <p>You can use this setting to disallow the use of short names in general.</p>

Background Information (DBSHORT=ON)

When the Natural compiler resolves a database field (that is, a field defined in a DDM) and DBSHORT=ON, the length of the field name is used to decide if the identifier represents a "db-short-name" or a "db-long-name". When the field name length is 2 characters a "db-short-name" reference is assumed, whereas all other identifiers are treated as "db-long-names".

Natural Rules that Always Apply

According to the general Natural rules, you must not use "db-short-names" in your program if a DEFINE DATA LOCAL was specified; not to create the field list in the view definition nor in a search expressions of a FIND statement nor to specify a read sequence control field in a READ or HISTOGRAM statement. All these restrictions are controlled by the Natural compiler, whether the option DBSHORT is ON or OFF.

Purpose of DBSHORT=OFF

The purpose of DBSHORT is to change the compiler's behavior as follows:

if set to ON , everything works as described above under Background Information.

if set to OFF, every database field identifier is regarded as a "db-long-name", no matter of how many characters it consist. In other words, only the long name column in the DDM (captioned as "Name" in a DDM display) is considered to locate the referenced field and the DDM short names (captioned as "DB" in a DDM display) are completely disregarded.

Purpose of DBSHORT=ON

The main reason for using DBSHORT=ON is when you have long names defined in a DDM with only 2-byte identifier length. At DDM generation, you may only create 2-byte long-names if the underlying database you access with this DDM is SQL (e.g. DB2). For all other database types, the attempt to define a long-field with 2-byte name length results in the error SYSDDM4219 (SYSDDM utility).

However, when DBSHORT=OFF is set, the compiler does not check db-short-names in a DDM. This leads to the syntax error NAT0981 if a db-short-field is used in a program.

PSIGNF - Internal Representation of Positive Sign of Packed Numbers

ON	The positive sign of a packed number is represented internally as H'F'. This is the default.
OFF	The positive sign of a packed number is represented internally as H'C'.

TSENABL - Applicability of TS Profile Parameter

This option determines whether the profile parameter TS (translate output for locations with non-standard lower-case usage) is to apply only to Natural system libraries (that is, libraries whose names begin with "SYS", except SYSTEM) or to all user libraries as well.

ON	The TS parameter applies to all libraries.
OFF	The TS parameter only applies to Natural system libraries. This is the default.

GFID - Generation of Global Format IDs

This option allows you to control Natural's internal generation of global format IDs so as to influence Adabas's performance concerning the re-usability of format buffer translations.

ON	Global format IDs are generated for all views. This is the default.
VID	Global format IDs are generated only for views in local/global data areas, but not for views defined within programs.
OFF	No global format IDs are generated.

For details on global format IDs, see the Adabas documentation.

Rules for generating GLOBAL FORMAT-IDs in Natural Version 3.1.

Note: STOD is the return value of the store clock machine instruction (STCK).

For Natural nucleus internal system-file calls

$GFID = abccdde$

where	equals
<i>a</i>	x'F9'
<i>b</i>	x'22' or x'21' depending on DB statement
<i>cc</i>	physical database number (2 bytes)
<i>dd</i>	physical file number (2 bytes)
<i>ee</i>	number created by runtime (2 bytes)

For user programs or Natural utilities

a) GFID=*abbbbbc* for file number less than or equal to 255 and Adabas Version lower than 6.2 (see NTDB macro).

where	equals
<i>a</i>	x'F8' or x'F7' or x'F6'
<i>bbbbb</i>	Bytes 1-6 of STOD value
<i>c</i>	physical file number

b) GFID=*axbbbbc* for file number greater than 255 and Adabas Version lower than 6.2.

where	equals
<i>a</i>	x'F8' or x'F7' or x'F6'
<i>x</i>	physical file number - high order byte
<i>bbbb</i>	Bytes 2-6 of STOD value
<i>c</i>	physical file number - low order byte

c) GFID=*abbbbb* for Adabas Version 6.2 or higher.

where	equals
<i>a</i>	x'F8' or x'F7' or x'F6' where: F6=UPDATE SAME F7=HISTOGRAM F8=all others
<i>bbbbbb</i>	Bytes 1-7 of STOD value

LOWSRCE - Allow Lower-Case Source

This option supports the use of lower or mixed-case program sources on mainframe platforms. It facilitates the transfer of programs written in mixed/lower-case characters from other platforms to a mainframe environment.

ON	Allows any kind of lower/upper-case characters in the program source.
OFF	Allows upper-case mode only. This requires keywords, variable names and identifiers to be defined in upper case. This is the default.

When you use lower-case characters with LOWSRCE=ON consider the following:

- The syntax rules for variable names allow lower-case characters in subsequent positions. Therefore, you can define two variables, one written with lower-case characters and the other with upper-case characters.
Example:

```
DEFINE DATA LOCAL
1 #Vari (A20)
1 #VARI (A20)
```

With LOWSRCE=OFF, these variables are treated as different variables.

With LOWSRCE=ON, the compiler is **not** case sensitive and does not make a distinction between lower/upper-case characters. This will lead to a syntax error because a duplicate definition of a variable is not allowed.

- Using the **session parameter** EM (Edit Mask) in an I/O statement or in a MOVE EDITED statement, there are characters which influence the layout of the data setting assigned to a variable (EM control characters), and characters which insert text fragments into the data setting.

Example:

```
#VARI := '1234567890'
WRITE #VARI (EM=XXXXXXxxXXXXXX)
```

With LOWSRCE=OFF, the output is 12345xx67890, because for alpha-format variables only upper-case **X**, **H** and circumflex accent (^) sign can be used.

With LOWSRCE=ON, the output is 1234567890, because an **x** character is treated like an upper-case **X** and, therefore, interpreted as an EM control characters for that field format. To avoid this problem, enclose constant text fragments in apostrophes (').

Example:

```
WRITE #VARI (EM=XXXXX'xx'XXXXX)
```

The text fragment is **not** considered an EM control character, regardless of the LOWSRCE settings.

- Since all variable names are converted to upper-case characters with LOWSRCE=ON, the display of variable names in I/O statements (INPUT, WRITE or DISPLAY) differs.

Example:

```
MOVE 'ABC' to #Vari
DISPLAY #Vari
```

With LOWSRCE=OFF, the output is:

```
      #Vari
-----
ABC
```

With LOWSRCE=ON, the output is:

```
      #VARI
-----
ABC
```

TQMARK - Translate Quotation Mark

ON	Each double quotation mark within a text constant is output as a single apostrophe. This is the default.
OFF	Double quotation marks within a text constant are not translated; they are output as double quotation marks.

Example:

```
RESET A(A5)
A:= 'AB"CD'
WRITE '12"34' / A / A (EM=H(5))
END
```

With TQMARK ON, the output is:

```
12'34
AB'CD
C1C27DC3C4
```

With TQMARK OFF, the output is:

```
12"34
AB"CD
C1C27FC3C4
```

FINDMUN - Detect Inconsistent Comparison Logic in FIND Statements

With Natural Version 2.3, the comparison logic for multiple-setting fields in the WITH clause of the FIND statement has been changed. This means that when Version 2.2 programs containing certain forms of FIND statements are compiled under Version 3.1, they will return different results. This option can be used to search for FIND statements whose WITH clauses use multiple-setting fields in a way that is no longer consistent with the enhanced Version 3.1 comparison logic.

ON	Error NAT0998 will be returned for every FIND statement of such form detected at compilation.
OFF	No search for such FIND statements will be performed. This is the default value.

The comparison logic for multiple-value fields in the WITH clause of the FIND statement has been changed with Natural Version 2.3 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 Version 2.2 and above, 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 Version 2.2, this statement returns records in which no occurrence of MU has the value "A" (same as 4.). With Version 2.3 and above, 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 2.3 and above, 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 and above, this statement returns records in which **no occurrence** of MU has the value "A". This means that if you newly compile under Version 2.3 existing Version 2.2 programs containing FIND statements of the forms **2.** and **3.**, they will return different results.

If you specify FINDMUN=ON, error NAT0998 will be returned for every FIND statement of form **2.** or **3.** detected at compilation.

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:

In Form 2:

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

into

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

In Form 3:

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

into

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

MASKCME - MASK Compatible with MOVE EDITED

MASKCME=ON	The range of valid year values that match the YYYY mask characters is 1582 - 2699 to make the MASK option compatible to MOVE EDITED.
MASKCME=OFF	The range of valid year values that match the YYYY mask characters is 0000 - 2699. This is the default value.

NMOVE22 - Assignment of Numeric Variables of Same Length and Precision

NMOVE22=ON	Assignments of numeric variables where source and target have the same length and precision is performed as with Natural Version 2.2.
NMOVE22=OFF	Assignments of numeric variables where source and target have the same length and precision is performed as with Natural Version 2.3 and above, that is they are processed as if source and target would have different length or precision. This is the default value.

V31COMP - Disable New Version 4.1 Syntax



This compiler option will be available only with Natural Version 4.1 to allow a smooth transition. It will be removed again with a subsequent release of Natural after Version 4.1.

A number of functions and programming features introduced with Natural Version 4.1 for Mainframes would give rise to problems when a program developed and compiled with Version 4.1 is to be recompiled for putting into operation in a Version 3.1 environment. The relevant functions or features are listed below.

The V31COMP option has been provided to detect such incompatibilities and trigger an error message that supplies a reason code for why the recompilation failed. The following values are possible:

V31COMP=ON	When a program is compiled under Version 4.1, every attempt to use a syntax construction that is supported by Version 4.1, but not by Version 3.1, is rejected and a NAT0647 syntax error and a corresponding reason code (see below) will be output.
V31COMP=OFF	The Version 3.1 compatibility option is disabled. This is the default.

Compilation Relevant Differences between Version 4.1 and 3.1

The following table gives an overview of the compilation relevant differences between Version 4.1 and 3.1 and indicates the number of the reason code that will be supplied when incompatible syntax is detected:

Function or Feature	Version 4.1	Version 3.1	Reason Code (see below)
Possible length for ALPHA field	1 byte - 1 GB	1 - 253 bytes	001
Possible length for BINARY field	1 byte - 1 GB	1 - 126 bytes	
Possible index range for array	-1 GB : +1 GB	-32 KB : +32 KB	002
Possible length of data group	1 byte - 1 GB	1 byte - 32 KB	003
Dynamic fields	possible	unknown	004
New Statements EXPAND / REDUCE / RESIZE ESCAPE MODULE [IMMEDIATE] ESCAPE TOP REPOSITION CALL .. INTERFACE4 DEFINE WORK FILE TYPE ' '	possible	unknown	005
New Adabas Features - FIND / READ / HISTO with MULTI-FETCH - New READ / HISTO comparators GT, GE, LT, LE - Ending-at check with keyword TO in READ / HISTO - New READ .. IN DYNAMIC SEQUENCE clause	possible	unknown	006
New System Variables *CPU-TIME, *DATV, *DATVS, *HOSTNAME, *LINE, *NATVERS, *PARM-USER, *PATCH-LEVEL, *PID, *LENGTH(..)	possible	unknown	007
IF BREAK with break variable of type B, F, I, D, T, L, H IF/AT BREAK .. with /n/ clause for type Binary	possible possible	NAT0623 error NAT0001 error	008
Max length of SORT record	10 KB	4KB (NAT0328 error)	009
SUBSTR for Binary fields MOVE ALL for Binary fields	possible possible	NAT0471 error NAT0101 error	010
New logical condition predicate IF SPECIFIED .. THEN	possible	unknown	011
Advanced arithmetic with Date / Time allow Multiply / Divide	possible	NAT0392	012
OPTIONAL parameter Definition: #P1 (A10) OPTIONAL Usage: CALLNAT 1X PERFORM .. 1X	possible	unknown	013
New COMPOPT parameter TQMARK, NMOVE22, MASKCME, (V31COMP)	possible	unknown	014

Reason Codes Supplied with Syntax Error NAT0647

With the compiler parameter V31COMP=ON set, the Version 4.1 features or functions listed above will result in the following syntax error:

NAT0647 - Program code not V31 compatible due to reason :1:

Reason Code Supplied under Version 3.1	Cause
001	Length for alpha/binary fields may not exceed 253/126 bytes.
002	Index range for array field must not exceed 32KB.
003	Length of data group must not exceed 32KB.
004	Field of type Dynamic is unknown.
005	ESCAPE TOP REPOSITION is unknown; ESCAPE MODULE is unknown; CALL .. INTERFACE4 is unknown; DEFINE WORK .. TYPE=.. clause is unknown.
006	FIND/READ/HISTOGRAM .. MULTI-FETCH clause is unknown; READ/HISTOGRAM comparator NE, LT, LE, GT, GE are unknown; READ/HISTOGRAM in DYNAMIC sequence is unknown.
007	System variables *CPU-TIME, *DATV, *DATVS, *HOSTNAME, *LINE, *NATVERS, *PARM-USER, *PATCH-LEVEL, *PID, *LENGTH(..) are unknown.
008	IF BREAK only allowed for field types A, N, P. IF/AT BREAK .. with /n/ option only allowed for field types A, N, P.
009	Length of a SORT record must not exceed 4 KB.
010	MOVE ALL for binary field is not allowed.
011	Logical condition 'IF .. SPECIFIED..' is unknown.
012	Operation Divide or Multiply not allowed for date/time fields.
013	Definition of OPTIONAL parameters is unknown.
014	OPTIONS parameters TQMARK, NMOVE22 and MASKCME are unknown.

Notes Concerning the Data Area Editor

No error will be reported if V31COMP=ON and the system commands READ and SAVE are executed from the NEXT prompt for a data area containing one of the above mentioned new features.

For the SAVE command in the data area editor, only those features that cannot be stored using the old Version 3.1 compatible data structure are checked and are not allowed. The following features are rejected when an attempt is made to save them in the old format:

- The first position of the L (level number) field contains values other than blank or 0.
- The length definition of the Length field contains more than 4 bytes. This includes the definition of dynamic variables.
- Array bounds that are defined by using the Array Index Definition function of the Extended Field Definition Editing.
- Definition of optional parameters.

The following error messages may be output:

- NAT4483 V31 source format error, LEVEL greater than 9
- NAT4483 V31 source format error, LENGTH field greater than 4 digits
- NAT4483 V31 source format error, ARRAY INDEX defined via '.e' command
- NAT4483 V31 source format error, OPTIONAL parameter definition

COMPOPT for Windows and UNIX

```
COMPOPT [option=value]
```

This command is used to set various compilation options. The options are evaluated when a Natural programming object is compiled.

option=value

When you issue the COMPOPT command without parameters, a screen is displayed on which you can set the options described below.

Instead of changing an option on the screen, you can also specify it directly with the COMPOPT command. The keywords for the individual options are shown (in parentheses on the COMPOPT screen) and in the above description.

Example:

```
COMPOPT DBSHORT=ON
```

Note:

The default settings of the individual options are set with the corresponding profile parameters in the Natural parameter module.

DBSHORT - Interpretation of Database Short Field Names

ON	Database field names in programming objects 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).
OFF	All database field names in programming objects are considered long names, regardless of their length. This avoids possible misinterpretations of database field names in programs.

GFID - Generation of Global Format IDs

This option allows you to control Natural's internal generation of global format IDs so as to influence Adabas's performance concerning the re-usability of format buffer translations.

ON	Global format IDs are generated for all views.
VID	Global format IDs are generated only for views in local/global data areas, but not for views defined within programs.
OFF	No global format IDs are generated.

For details on global format IDs, see the Adabas documentation.