COMPOPT

COMPOPT [*option=value ...*]

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

If you enter the COMPOPT command without any options, a screen is displayed where you can enable or disable the options described below.

The default settings of the individual options are set with the corresponding keyword subparameters of the parameter macro NTCMPO in the Natural parameter module or in the profile parameter CMPO. When you change the library, the COMPOPT options are reset to their default values.

This chapter covers the following topics:

- Syntax Explanation
- Specifying Compiler Keyword Parameters
- General Compilation Options
- Compilation Options for Ensuring Version Compatibility

Syntax Explanation

COMPOPT	If you issue the COMPOPT system command without options, the **Compilations Options** screen appears. The keywords available there are described below.
COMPOPT ***option=value***	The keywords for the individual options are described below. The setting assigned to a compiler option is 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.

Specifying Compiler Keyword Parameters

You can specify compiler keyword parameters on different levels:

1. The default settings of the individual keyword parameters are specified in the macro NTCMPO in the Natural parameter module NATPARM.
2. At session start, you can override the compiler keyword parameters with the profile parameter CMPO.
3. During an active Natural session, there are two ways to change the compiler keyword parameters with the COMPOPT system command: either directly using command assignment (COMPOPT *option=value*) or by issuing the COMPOPT command without keyword parameters which

displays the **Compilation Options** screen. 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:

```
OPTIONS KCHECK=ON
DEFINE DATA LOCAL
1 #A (A25) INIT <'Hello World'>
END-DEFINE
WRITE #A
END
```

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

   ```
   OPTIONS KCHECK=ON
   WRITE 'Hello World'
   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.

General Compilation Options

The following options are available:

- KCHECK - Keyword Checking
- PCHECK - Parameter Checking for Object Calling 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
- THSEP - Dynamic Thousands Separator
- CPAGE - Code Page Support for Alphanumeric Constants
- DB2ARRY - Support DB2 Arrays in SQL SELECT and INSERT Statements
- CHKRULE - Validate INCDIR Statements in Maps

These options correspond to the keyword subparameters of the CMPO profile parameter and/or the NTCMPO parameter macro.

KCHECK - Keyword Checking

ON	Field declarations in a programming object will be checked against a set of critical Natural keywords. If a variable name defined matches one of these keywords, a syntax error is reported when the programming object is checked or cataloged.
OFF	No keyword check is performed. This is the default value.

The section *Performing a Keyword Check* (in the *Programming Guide*) contains a list of the keywords that are checked by the KCHECK option.

The section *Alphabetical List of Natural Reserved Keywords* (in the *Programming Guide*) contains an overview of all Natural keywords and reserved words.

PCHECK - Parameter Checking for Object Calling Statements

ON	The compiler checks the number, format, length and array index bounds of the parameters that are specified in an object calling statement, such as CALLNAT, PERFORM, INPUT USING MAP, PROCESS PAGE USING, helproutine call. Also, the OPTIONAL feature of the DEFINE DATA PARAMETER statement is considered in the parameter check. The parameter check is based on a comparison of the parameters of the object calling statement with the DEFINE DATA PARAMETER definitions for the object to be invoked. It requires that • the name of the object to be called is defined as an alphanumeric constant (not as an alphanumeric variable), • the object to be called is available as a cataloged object. Otherwise, PCHECK=ON will have no effect. **Problems in Using the CATALL Command with PCHECK=ON** When a CATALL command is used in conjunction with PCHECK=ON, you should consider the following: If a CATALL process is invoked, the order in which the programming objects are compiled depends primarily on the type of the object and secondarily on the alphabetical name of the object. The object type sequence used is: GDAs, LDAs/PDAs, external subroutines, subprograms, helproutines, maps/adapters, programs/classes. Within objects of the same type, the alphabetical order of the name determines the sequence in which they are cataloged. As mentioned above, the parameters of the object calling statement are checked against the compiled form of the called object. If the calling object (the one which is being compiled and includes the object calling statement) is cataloged before the invoked object, the PCHECK result may be wrong if the parameters in the invoking statement and in the called object were changed. In this case, the new object image of the called object has not yet been produced by the CATALL command. This causes the *new* parameter layout in the object calling statement to be compared with the *old* parameter layout of the DEFINE DATA PARAMETER statement of the called subprogram. Solution: • Set compiler option PCHECK to OFF. • Perform a general compile with CATALL on the complete library, or if just one or a few objects were changed, perform a separate compile on these objects. • Set compiler option PCHECK=ON. • On the complete library, perform a general compile with CATALL, selecting function CHECK.
OFF	No parameter check is performed. This is the default value.

DBSHORT - Interpretation of Database Short Field Names

A database field defined in a DDM is described by two names:

- the short name with a length of 2 characters, used by Natural to communicate with the database (especially with Adabas);
- the long name with a length of 3-32 characters (1-32 characters, if the underlying database type accessed is DB2/SQL), which is supposed to be used to reference the field in the Natural programming code.

Under special conditions, you may reference a database field in a Natural program with its short name instead of the long name. This applies if running in Reporting Mode without Natural Security and if the database access statement contains a reference to a DDM instead of a view.

The decision if a field name is regarded as a short-name reference depends on the name length. When the field identifier consists of two characters, a short-name reference is assumed; a field name with another length is considered as a long-name reference. This standard interpretation rule for database fields can additionally be influenced and controlled by setting the compiler option `DBSHORT` to `ON` or `OFF`:

ON	The usage of a short name is allowed for referencing a database field. However, a data base short name is *not permitted* in general (even if `DBSHORT=ON`) • for the definition of a field when a view is created; • when a view field is used in the programming code; • when a `DEFINE DATA LOCAL` statement was previously used to defines variables; • when running under Natural Security. This is the default value.
OFF	A database field may only be referenced via its long name. Every database field identifier is considered as a long-name reference, regardless of its length. If a two character name is supplied which can only be found as a short name but not as a long name, syntax error NAT0981 is raised at compile time. This makes it possible to use long names defined in a DDM with 2-byte identifier length. This option is essential if the underlying database you access with this DDM is SQL (DB2) and table columns with a two character name exist. For all other database types (for example, Adabas), however, any attempt to define a long-field with a 2-byte name length will be rejected at DDM generation. Moreover, if no short-name references are used (what can be enforced via `DBSHORT=OFF`), the program becomes independent of whether it is compiled under Natural Security or not.

Examples:

Assume the following data base field definition in the DDM EMPLOYEES:

Short Name	Long Name
AA	PERSONNEL-ID

Example 1:

```
OPTIONS DBSHORT=ON
READ EMPLOYEES
  DISPLAY AA      /* data base short name AA is allowed
END
```

Example 2:

```
OPTIONS DBSHORT=OFF
READ EMPLOYEES
  DISPLAY AA      /* syntax error NAT0981, because DBSHORT=OFF
END
```

Example 3:

```
OPTIONS DBSHORT=ON
DEFINE DATA LOCAL
1 V1 VIEW OF EMPLOYEES
  2  PERSONNEL-ID
END-DEFINE
READ V1 BY PERSONNEL-ID
  DISPLAY AA     /* syntax error NAT0981, because PERSONNEL-ID is defined in view;
                 /* (even if DBSHORT=ON)
END-READ
END
```

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 value.
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.

Natural objects cataloged with TSENABL=ON determine the TS parameter even if they are located in a non-system library.

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

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 value.
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

- **For Natural nucleus internal system-file calls:**

```
GFID=abccddee
```

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

- **For user programs or Natural utilities:**
  - `GFID=abbbbbbc` for file number less than or equal to 255 and Adabas Version lower than 6.2 (see `NTDB` macro).

where	equals
a	x'F8' or x'F7' or x'F6'
bbbbbb	bytes 1-6 of STOD value
c	physical file number

  - `GFID=axbbbbbc` for file number greater than 255 and Adabas Version lower than 6.2.

where	equals
a	x’F8’ or x’F7’ or x’F6’
x	physical file number - high order byte
bbbbb	Bytes 2-6 of STOD value
c	physical file number - low order byte

- `GFID=abbbbbb` for Adabas Version 6.2 or higher.

where	equals
a	x’F8’ or x’F7’ or x’F6’ where: F6=`UPDATE SAME` F7=`HISTOGRAM` F8=all others
bbbbbbb	bytes 1-7 of STOD value

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

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 value.

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=XXXXXxxXXXXX)
  ```

  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 character 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 value.
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
```

THSEP - Dynamic Thousands Separator

This option can be used to enable or disable the use of thousands separators at compilation time. See also the profile and session parameter THSEPCH and the section *Customizing Separator Character Displays* (in the *Programming Guide*).

ON	Thousands separator used. Every thousands separator character that is not part of a string literal is replaced internally with a control character.
OFF	Thousands separator not used, i.e. no thousands separator control character is generated by the compiler. This is the compatibility setting.

CPAGE - Code Page Support for Alphanumeric Constants

The CPAGE option can be used to activate a conversion routine which translates all alphanumeric constants (from the code page that was active at compilation time into the code page that is active at runtime) when the object is started at runtime.

See also *CPAGE Compiler Option* in the *Unicode and Code Page Support* documentation.

ON	Code page support for alpha strings is enabled.
OFF	Code page support for alpha strings is disabled. This is the default value.

DB2ARRY - Support DB2 Arrays in SQL SELECT and INSERT Statements

The DB2ARRY option can be used to activate retrieval and/or insertion of multiple rows from/into DB2 by a single SQL SELECT or INSERT statement execution. This allows the specification of arrays as receiving fields in the SQL SELECT and as source fields in the SQL INSERT statement. If DB2ARRY is ON, it is no longer possible to use Natural alphanumeric arrays for DB2 VARCHAR/GRAPHIC columns. Instead of these, long alphanumeric Natural variables have to be used.

ON	DB2 array support is enabled.
OFF	DB2 array support is not enabled. This is the default value.

CHKRULE - Validate INCDIR Statements in Maps

The CHKRULE option can be used to enable or disable a validation check during the catalog process for maps.

ON	INCDIR validation is enabled. If the file (DDM) or field referenced in the INCDIR control statement does not exist, syntax error NAT0721 is raised at compile time. When a Natural map is created, you may include fields which are already defined inside another existing programming object. This works with nearly all kinds of objects which allow you to define variables and also with DDMs. When the included field is a database variable, it is a map editor built-in behavior to automatically add (besides the included field) an additional INCDIR statement in the map statement body to trigger a Predict rule upload and incorporation when the map is compiled (STOW). The function is similar to what is happening when an INCLUDE statement is processed. However, instead of getting the source lines from a copycode object, they are received from Predict. The search key to find the rule(s) are the DDM name (which is regarded as the file name) and the field name. Both are indicated in the INCDIR statement. An INCDIR rule requested at compile time has not got to be found on Predict, as there is absolutely no requirement for its existence. That implies, it is by no means an error situation if a searched rule is not found. When fields are incorporated from a DDM into a map, the corresponding INCDIR statements are created, including the current DDM and field name as "search key" to request existent rules from Predict. However, if the DDM is renamed after the copy process, the old DDM name (which is not valid anymore) still continues to be used in the INCDIR statement. This causes that no rule is loaded and the programmer is not informed about this. Moreover, it is not only a DDM rename causing this situation. The more likely situation effecting this consequence is to have a wrong FDIC file assigned, by any mistake. In this case, the DDM name is valid, but it cannot be found on the current Predict system file. Then the result is same as when the DDM does not exist at all; the processing rules supposed to be added from Predict are not included.
OFF	INCDIR validation is disabled. This is the default value.

Compilation Options for Ensuring Version Compatibility

The following options are available:

- FINDMUN - Detect Inconsistent Comparison Logic in FIND Statements
- MASKCME - MASK Compatible with MOVE EDITED
- NMOVE22 - Assignment of Numeric Variables of Same Length and Precision
- V41COMP - Disable New Version 4.2 Syntax

These options correspond to the keyword subparameters of the CMPO profile parameter and/or the NTCMPO parameter macro.

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

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. If the profile parameter MAXYEAR is set to 9999, the range of valid year values is 1582 - 9999.
OFF	The range of valid year values that match the YYYY mask characters is 0000 - 2699. This is the default value. If the profile parameter MAXYEAR is set to 9999, the range of valid year values is 0000 - 9999.

NMOVE22 - Assignment of Numeric Variables of Same Length and Precision

ON	Assignments of numeric variables where source and target have the same length and precision is performed as with Natural Version 2.2.
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.

V41COMP - Disable New Version 4.2 Syntax

Important:
This compiler option will be available only with Natural Version 4.2 to allow a smooth transition. It will be removed again with a subsequent release of Natural after Version 4.2.

A number of functions and programming features introduced with Natural Version 4.2 would give rise to problems when a program developed and compiled with Version 4.2 is to be recompiled for putting into operation in a Version 4.1 environment. The relevant functions or features are listed below.

The V41COMP 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:

ON	When a program is compiled under Version 4.2, every attempt to use a syntax construction that is supported by Version 4.2, but not by Version 4.1, is rejected and a NAT0647 syntax error and a corresponding reason code (see below) will be output.
OFF	A test for Version 4.1 compatibility is not performed. This is the default value.

Compilation Relevant Differences between Version 4.2 and 4.1

The following table gives an overview of the compilation relevant differences between Version 4.2 and 4.1 and indicates the reason code that will be supplied when incompatible syntax is detected:

Function or Feature	Version 4.2	Version 4.1	Reason Code
New format U (Unicode)	possible	unknown	001

Array with variable number of occurences
 	
X-array, for example:
 	
`DEFINE DATA LOCAL`
`1 #ARR (A10/1:*)`
	possible	unknown	002

Possible length of alpha and literals (constants)	1 byte - 1 GB	1 byte - 253 bytes (NAT0264)	003
New compiler parameters: `THSEP` Thousands separator character in edit mask `CPAGE` Make alphunmeric constants sensitive for code page translation	possible	unknown	004

New statements:
 	
`MOVE NORMALIZED`
`MOVE ENCODED`
`PARSE`
`REQUEST DOCUMENT`
`EXPAND / REDUCE / RESIZE ARRAY`
	possible	unknown	005

Statement `SET GLOBALS`: • session parameter `CPCVERR=ON/OFF` • allowed when in structured mode (`SM=ON`)	possible	unknown	006

Function or Feature	Version 4.2	Version 4.1	Reason Code

New system variables:
 	
`*PARSE-COL`
`*PARSE-LEVEL`
`*PARSE-NAMESPACE-URI`
`*PARSE-ROW`
`*PARSE-TYPE`
`*CODEPAGE`
`*LOCALE`
`*TYPE`
`*CURRENT-UNIT`
`*UBOUND`
`*LBOUND`
	possible	unknown	007

Not used	-	-	008

Length and type of source parameters supplied with `INCLUDE`
 	
Example:
 	
`INCLUDE COPY01 'WRITE *LINE'`
`'WRITE *PROGRAM'`
	any length and format U (Unicode) allowed	only alpha with a length of max. 80 bytes	009

Definition of an Adabas LA-field in a data view • with a size greater than 253 bytes or • of type `DYNAMIC`	possible	unknown	010