software AG

Natural

System Commands

Version 6.3.13 for Windows

October 2012

Natural

Table of Contents

Preface

This documentation describes the Natural system commands.

Natural system commands perform functions you need to create, maintain or execute Natural programming objects. In addition, Natural system commands are used to monitor and administer your Natural environment.

This documentation is organized under the following headings:

Issuing System Commands	Describes the general rules that apply when you enter a Natural system command.
System Command Syntax	Explains the symbols that are used within the syntax descriptions of Natural system commands.
System Commands Grouped by Function	Provides an overview of the Natural system commands grouped according to their functions.
System Commands in Alphabetical Order	Descriptions of the system commands in alphabetical order.

Notation *vrs* **or** *vr*

When used in this documentation, the notation *vrs* or *vr* represents the relevant product version (see also *Version* in the *Glossary*).

1   Issuing System Commands

Command Input

You can issue a system command by entering it in one of the following ways:

- In the **command line**;
- At the Natural NEXT or MORE prompt.

The following rules apply:

- Command input is not case-sensitive.
- Commands are context-sensitive.
- Some Natural commands affect objects other than the currently active object.

For an explanation of the symbols that are used within the syntax descriptions, see *System Command Syntax*.

Command Line

The functionality of system commands is available via various menus. You can also enter system commands in the command line. See *Issuing Commands in the Command Line* in *Using Natural Studio*.

NEXT Prompt

The NEXT prompt appears in a Natural application or program when no more output is pending.

MORE Prompt

The MORE prompt is displayed at the bottom of an output screen to signal that more output is pending. When a system command is entered in response to a MORE prompt, program execution is interrupted and the system command is executed.

2 System Command Syntax

Syntax Elements

The following symbols are used within the syntax descriptions of system commands:

Element	Explanation
ABCDEF	Upper-case letters indicate that the term is either a Natural keyword or a Natural reserved word that must be entered exactly as specified.
ABCDEF	If an optional term in upper-case letters is completely underlined (not a hyperlink!), this indicates that the term is the default value. If you omit the term, the underlined value applies.
ABCDEF	If a term in upper-case letters is partially underlined (not a hyperlink!), this indicates that the underlined portion is an acceptable abbreviation of the term.
abcdef	Letters in italics are used to represent variable information. You must supply a valid value when specifying this term.
[ ]	Elements contained within square brackets are optional. If the square brackets contain several lines stacked one above the other, each line is an optional alternative. You may choose at most one of the alternatives.
{ }	If the braces contain several lines stacked one above the other, each line is an alternative. You must choose exactly one of the alternatives.
|	The vertical bar separates alternatives.
...	A term preceding an ellipsis may optionally be repeated. A number after the ellipsis indicates how many times the term may be repeated. If the term preceding the ellipsis is an expression enclosed in square brackets or braces, the ellipsis applies to entire bracketed expression.
,...	A term preceding a comma-ellipsis may optionally be repeated; if it is repeated, the repetitions must be separated by commas. A number after the comma-ellipsis indicates how many times the term may be repeated. If the term preceding the comma-ellipsis is an expression enclosed in square brackets or braces, the comma-ellipsis applies to entire bracketed expression.
:...	A term preceding a colon-ellipsis may optionally be repeated; if it is repeated, the repetitions must be separated by colons. A number after the colon-ellipsis indicates how many times the term may be repeated. If the term preceding the colon-ellipsis is an expression enclosed in square brackets or braces, the colon-ellipsis applies to entire bracketed expression.
Other symbols (except [ ] { } | ... ,... :...)	All other symbols except those defined in this table must be entered exactly as specified. **Exception:** The SQL scalar concatenation operator is represented by two vertical bars that must be entered literally as they appear in the syntax definition.

Example of Command Syntax

```
CATALOG [object-name [library-id]]
```

- CATALOG is a Natural keyword which you must enter as specified. The underlining indicates that you may also enter it in abbreviated form as CAT.

- *object-name* and *library-id* are user-supplied operands for which you specify the name of the program you wish to deal with and the ID of the library in which that program is contained.

- The square brackets indicate that *object-name* and *library-id* are optional elements which you can, but need not, specify. The grouping of the brackets indicate that you can specify CATALOG alone, or CATALOG followed either by a program name only or by a program name and a library ID; however, you cannot specify a library ID if you do not also specify a program name.

3 System Commands Grouped by Function

This chapter provides an overview of the Natural system commands grouped according to their functions.

Navigating within Natural

Command	Function
FIN	Terminates a Natural session.
LOGOFF	Causes the library ID to be set to SYSTEM and the Adabas password to be set to blanks. The contents of the source program work area are not affected by this command.
LOGON	Establishes a library ID for the user. In the specified library, all source or object programs saved during the session will be stored (unless you explicitly specify another library ID in a SAVE, CATALOG or STOW command).
RETURN	Returns to a return point set by a SETUP command.
SETUP	Establishes a return point to which control can be returned using a RETURN command. This allows you to easily transfer from one application to another during a Natural session.

Environment Settings

Command	Function
COMPOPT	Sets various compilation options that affect the way in which Natural programming objects are compiled.
GLOBALS	Changes the settings of various Natural session parameters.
MAP	Establishes a connection to a remote development server.
PROFILE	Only available if Natural Security has been installed. Displays the security profile currently in effect. This profile informs you of the conditions of use in effect for you in your current Natural environment.
SYSPROD	Displays a list of the products installed at your site, and some information on these products.
SYSPROF	Displays the current definitions of the Natural system files.
UNMAP	Disconnects the currently active remote environment.

Editing and Storing Programming Objects

Command	Function
CATALL	Catalogs *all* objects or selected objects in the current library.
CATALOG	Compiles the Natural programming object currently in the source work area of an editor, and if the syntax has been found to be correct, stores the resulting object module in the Natural system file.
CHECK	Checks that the source code of a programming object does not contain any syntax errors. The checking process varies according to the type of object being checked. Syntax checking is also performed as part of the system commands RUN, CATALL, CATALOG and STOW.
CLEAR	Closes the currently active object and opens a new editor window without content and without a name. The type of editor is the same as for the currently active object. If the currently active object has been modified since the last save, you are prompted to save any changes.
EDIT	Edits the source form of a programming object.
LIST	Lists one or more objects which are contained in the current library.
READ	Transfers an object in source form from the Natural system file to the source work area.
RENUMBER	Renumbers the source code which is currently held in the source work area.
SAVE	Stores the *source form* of the programming object currently in the work area of the editor in the Natural system file.
SCAN	Searches for a string of characters within an object, with an option to replace the string with another string.
STOW	Compiles and stores a Natural programming object (in both source and object form) in the Natural system file.
SYSWIZDW	Invokes the Natural Dialog Wizard, a tool for creating dialogs for specific purposes. The defined dialogs can have several layouts that adapt to desired requirements.

Executing Programs

Command	Function
EXECUTE	Executes a program that has been compiled and stored in object form. You can EXECUTE a program only if it has been stored in compiled form.
RUN	Compiles and executes the source program currently in the work area of the editor.

Maintenance Utilities

Command	Function
SYSERR	Creates and maintains the messages you wish your Natural applications to display to the users.
SYSNCP	Creates and maintains the command processors to be used in your Natural applications.
SYSRPC	Creates and maintains remote procedure calls, that is, provides the settings necessary to execute a subprogram located on a remote server.

Transfer of Programming Objects

Command	Function
SYSMAIN	Transfers objects within the Natural system from one library to another.
SYSOBJH	Processes Natural and non-Natural objects for distribution in Natural environments.

Monitoring and Debugging

Command	Function
DEBUG	This command is used to invoke the Natural debugger.
RPCERR	Displays the last Natural error number and message if related to Remote Procedure Call (RPC), and the last Broker reason code and associated message.
TECH	Displays technical and other information on your Natural session.

Commands Used with NaturalX

Command	Function
REGISTER	Registers Natural classes. They are registered for the server ID under which Natural was started.
UNREGISTER	Unregisters Natural classes.

Miscellaneous

Command	Function
HELP	Invokes the Natural help system.
INPL	Invokes the INPL utility. It is *only* used for the loading of Software AG installation datasets into the system files.
LAST	Displays the system commands that were last executed, and allows you to execute them again.
LASTMSG	Displays additional information on the error situation which occurred last.
LIST XREF	Only available if Predict has been installed. Displays all active cross-reference data for the current library.
MAIL	Only available if Natural Security has been installed. Invokes a mailbox to modify its contents and/or expiration date. A mailbox is used as a notice board to broadcast messages to Natural users.
SYSAPI	Invokes the SYSAPI utility, which is used to locate Application Programming Interfaces (APIs) provided by Natural add-on products.
SYSEXT	Invokes the library SYSEXT, which contains various Natural user application interfaces.
SYSEXV	Invokes the SYSEXV application with examples of the new features of the current Natural versions.
SYSFILE	Invokes the function Natural Print/Work Files of the SYSFILE utility. This utility provides information on the work files and print files available.
SYSWIZDB	Invokes the Natural Data Browser, a development tool wizard within Natural Studio. It enables you to display and print or store file structures.
UNLOCK	Enables you to view locked objects and unlock them if required.
UPDATE	Prevents database updating being carried out by a program.
XREF	Only available if Predict has been installed. Controls the usage of the Predict function "active cross-references". This function automatically creates documentation in Predict about the objects which a program/data area references.

4 CATALL

This command is used to catalog, check, save or stow all objects or selected objects in the current library.

CATALL in Interactive Mode

```
CATALL
```

When you issue this command, the **Catalog Objects in Library** dialog box appears. In this dialog box, you specify which types of objects are to be processed. Objects are processes in the order in which the object types are listed in the dialog box. Additionally, you can choose which action is to be performed and which objects are to be processed.

See also *Cataloging the Objects in a Library* in *Using Natural Studio*.

You can make the following specifications in the dialog box:

Starting from	Enter an asterisk (*) if you want to process all objects of the selected types in the current library. If you want to restrict the number of objects to a certain range, you can use asterisk notation for the name.
Apply action only to existing modules	If you mark this option, only those objects for which object modules already exist in the current library will be cataloged again; objects that exist only in source form will not be processed.
Apply action to all sources	If you mark this option, *all* selected objects will be processed.
Action	You can select one of the following actions to be applied to the selected objects: ■ **Catalog** ■ **Check** ■ **Save** ■ **Stow** These actions correspond to the system commands of the same names. **Note:** Under Natural Security, certain actions may be disallowed.
Renumber source lines	By default, the source-code lines of sources that were saved or stowed are also renumbered. If you wish no automatic renumbering of lines, deactivate this checkbox.

Object types	By default, CATALL applies to objects of all types in the current library (all object types are activated). If you wish objects of a certain type *not* to be affected by CATALL, deactivate the corresponding option.
	In addition, command buttons are available to select all options or to clear all check boxes.
	Note: When you are working in a remote mainframe development environment using SPoD, the options **DDMs** and **Generate new map source** cannot be used and will be dimmed.
Generate new map source	Maps created with previous Natural versions are not necessarily compatible with Natural Version 3.1 and above. Mark this option to ensure that maps are converted during the catalog operation. This option converts and stores maps in source *and* object form.
	When you are working in a remote mainframe development environment using SPoD, this option cannot be used and will be dimmed.

CATALL in Batch Mode

$$
\text{CATALL } \textit{object-name} \left[\begin{array}{c} \underline{\text{RECAT}} \\ \text{ALL} \end{array} \right] [\text{TYPES } \textit{types}] \left[\begin{array}{c} \text{CATALOG} \\ \text{CHECK} \\ \text{SAVE} \\ \text{STOW} \end{array} \right] [\textit{options} \dots]
$$

For the various specifications you can make in the **Catalog Objects in Library** dialog box, there are also corresponding options which you can specify directly with the system command CATALL:

object-name	The name of the object to be cataloged.
	Enter an asterisk (*) if you want to catalog all objects of the specified types in the current library.
	If you want to restrict the number of objects to a certain range, you can use asterisk notation for the name.
RECAT / ALL	Corresponds to the options **Apply action only to existing modules**, or **Apply action to all sources** of the **Catalog Objects in Library** dialog box. RECAT is the default.
TYPES *types*	Corresponds to the object types marked in the **Catalog Objects in Library** dialog box. Possible *types* are:
	G Global data areas
	A Parameter data areas
	L Local data areas

	D DDM		
	S Subroutine		
	N Subprogram		
	H Helproutine		
	M Map		
	P Program		
	3 Dialog		
	4 Class		
	7 Function		
	8 Adapter		
	* All types (this is the default)		
	The *types* have to be specified as *one* character string, for example, `LAG` for local, parameter and global data areas. By default, `CATALL` applies to objects of all types in the current library.		
`CATALOG /` `CHECK / SAVE` `/ STOW`	Corresponds to the actions of the same names on the **Catalog Objects in Library** dialog. `CATALOG` is the default.		
options	`NOREN`		No automatic renumbering of source-code lines of source objects.

> **Note:** The individual command components must be separated from one another either by a blank or by the input delimiter character (as defined with the session parameter `ID`).

5 CATALOG

```
CATALOG [object-name [library-id]]
```

Related commands: SAVE | STOW | UNCATALOG.

This command is used to compile the Natural programming object currently in the source work area of an editor and (if the syntax has been found to be correct) store the resulting object module in the Natural system file.

See also:

Cataloging Objects in *Using Natural Studio*
Object Naming Conventions in *Using Natural Studio*

⚠ **Important:** The CATALOG command cannot be used if the profile parameter RECAT has been set to ON; in this case, use the STOW command to compile and store the object.

CATALOG	If you do not specify an *object-name*, the object is cataloged in the library under the name of the object last read into the source work area (for example, with EDIT or READ).
CATALOG *object-name*	A new object is created. As *object-name*, you specify the name under which the new object is to be cataloged. It is stored in the current library. If the object exists, the command is rejected.
CATALOG *object-name* *library-id*	If you want the new object to be cataloged into another library, you must specify the *library-id* of that library. If the object exists, the command is rejected.

📄 **Note:** If an FDIC system file is specified in the parameter file which is not valid, Natural will display an appropriate error message when the CATALOG command is issued.

6 CHECK

```
CHECK
```

This command is used to check if the syntax of the source code currently in the editor work area contains any errors.

If a syntax error is detected, syntax checking is suspended and the line containing the error is displayed. You can then either correct the line (whereupon verification continues) or press ENTER without modifying the line displayed. This stops the verification procedure and opens the editor.

> **Note:** Syntax checking is also performed as part of the RUN, STOW, CATALOG and CATALL commands.

See also *Checking Objects* in *Using Natural Studio*.

7 CLEAR

```
CLEAR
```

This command is used to close the currently active object and to open a new editor window without content and without a name. The type of editor is the same as for the currently active object.

If the currently active object has been modified since the last save, you are prompted to save any changes.

See also *Clearing Editor Windows* in *Using Natural Studio*.

8 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 profile parameters in the Natural parameter file.

Syntax Explanation

COMPOPT	If you issue the COMPOPT system command without options, a dialog box appears. The keywords available there are described below. See also *Compiler Options* in *Using Natural Studio*.
COMPOPT option=value	Instead of changing an option in the dialog box, you can also specify it directly with the COMPOPT command. Example: `COMPOPT DBSHORT=ON`

Compiler Options

The following compiler options are available. For details on the purpose of these options and the possible settings, see the description of the corresponding Natural profile parameter:

DBSHORT | GFID | KCHECK | MASKCME | PCHECK | PSIGNF | THSEP | TQMARK

Specifying Compiler Parameters

You can specify compiler parameters on different levels:

1. **As Default Settings**

   The default settings of the individual compiler parameters are specified using the **Compiler Options** category of the Configuration Utility and are stored in the Natural parameter file NATPARM.

2. **At Session Start**

   At session start, you can override the compiler option settings by specifying the corresponding profile parameters.

3. **During an Active Natural Session**

   During an active Natural session, there are two ways to change the compiler parameter values with the COMPOPT system command: either directly using command assignment (COMPOPT *option=value*) or by issuing the COMPOPT command without options which displays the **Compiler Options** dialog box. The settings assigned to a compiler option are in effect until you issue the next LOGON command to another library. At LOGON to a different libary, the default settings (see item 1 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**

   In a Natural programming object (for example: program, subprogram), you can set compiler parameters 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.

COMPOPT in a Remote Mainframe Environment

The topics provided below apply when using the COMPOPT command in a remote mainframe environment.

Specifying Compiler Keyword Parameters (Remote Mainframe Environment)

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 (Remote Mainframe Environment)

- CHKRULE - Validate INCDIR Statements in Maps
- CPAGE - Code Page Support for Alphanumeric Constants
- DBSHORT - Interpretation of Database Short Field Names
- DB2ARRY - Support DB2 Arrays in SQL SELECT and INSERT Statements
- DB2BIN – Generate SQL Binary Data Types for Natural Binary Fields
- DB2TSTI – Generate SQL TIMESTAMP Data Type for Natural TIME Fields

- ECHECK - Existence Check for Object Calling Statements
- GDASC - GDA Signature Check
- GFID - Generation of Global Format IDs
- KCHECK - Keyword Checking
- LOWSRCE - Allow Lower-Case Source
- MAXPREC – Maximum Number of Digits after Decimal Point
- MEMOPT - Memory Optimization for Locally Declared Variables
- PCHECK - Parameter Check for Object Calling Statements
- PSIGNF - Internal Representation of Positive Sign of Packed Numbers
- THSEP - Dynamic Thousands Separator
- TQMARK - Translate Quotation Mark
- TSENABL - Applicability of TS Profile Parameter

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

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.

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.

ON	Code page support for alpha strings is enabled.
OFF	Code page support for alpha strings is disabled. 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 DEFINE DATA LOCAL statement was specified; - 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
```

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.

DB2BIN – Generate SQL Binary Data Types for Natural Binary Fields

The `DB2BIN` option can be used to support the DB2 data types BINARY and VARBINARY.

If `DB2BIN` is set to `OFF`, Natural binary fields (format B(n)) are generated as SQL data type CHAR (n<= 253) or VARCHAR (253<n<=32767) like it was in previous releases. `DB2BIN=OFF` is good for those who used Natural binary fields like SQL CHAR fields. B2 and B4 are treated as SQL SMALLINT and INTEGER.

If `DB2BIN` is set to `ON`, Natural binary fields (format B(n)) are generated as SQL data type BINARY (n<=255) or VARBIN (255<n<=32767). `DB2BIN=ON` is good for those who want to use SQL binary columns. B2 and B4 are also treated as SQL BINARY(2) and BINARY(4).

> **Note:** The setting of `DB2BIN` at the end of the compilation is used for the complete Natural object. It cannot be changed for parts of a Natural object.

ON	SQL types BINARY and VARBIN are generated for Natural binary fields.
OFF	SQL types CHAR and VARCHAR are generated for Natural binary fields, except B2 and B4. The latter are treated as SQL data types SMALLINT and INTEGER. This is the default value.

DB2TSTI – Generate SQL TIMESTAMP Data Type for Natural TIME Fields

This option is used to map Natural `TIME` variables to the SQL TIMESTAMP data type instead of the default SQL TIME data type.

ON	SQL type TIMESTAMP is generated for Natural `TIME` fields of Natural data format T.
	This applies to the entire Natural object. You cannot generate only part of an object with the `DB2TSTI` setting.
OFF	SQL type TIME is generated for Natural `TIME` fields of Natural data format T.
	This is the default value.

> **Note:** A Natural `TIME` field only contains tenth of seconds as precision while a SQL TIMESTAMP column can contain a much greater precision. Thus, the TIMESTAMP value read from the SQL database may be truncated if `DB2TSTI=ON` is set.

ECHECK - Existence Check for Object Calling Statements

ON	The compiler checks for the existence of a programming object that is specified in an object calling statement, such as `FETCH [RETURN/REPEAT]`, `RUN [REPEAT]`, `CALLNAT`, `PERFORM`, `INPUT USING MAP`, `PROCESS PAGE USING`, function call, helproutine call.
	The existence check is based on a search for the cataloged object or for the source of the object when it is invoked by a `RUN [REPEAT]` statement.
	It requires that the name of the object to be called/run is defined as an alphanumeric constant (not as an alphanumeric variable).
	Otherwise, `ECHECK=ON` will have no effect.
	Error Control for ECHECK=ON
	The existence check is executed only when the object does not contain any syntax errors. The existence check is executed for every object calling statement.
	The existence check is controlled by the `PECK` profile parameter (see the *Parameter Reference* documentation).
	Problems in Using the CATALL Command with ECHECK=ON
	When a `CATALL` system command is used in conjunction with `ECHECK=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, functions, subprograms, external subroutines, help routines, 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 success of the object calling statement is 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 ECHECK result may be wrong if the called object was not cataloged beforehand. In this case, the object image of the called object has not yet been produced by the CATALL command. Solution: - Set compiler option ECHECK 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 ECHECK=ON. - On the complete library, perform a general compile with CATALL, selecting function CHECK.
OFF	No existence check is performed. This is the default setting.

GDASC - GDA Signature Check

This option is used to store information on the structure of a GDA (global data area) to determine whether a Natural error is to be issued when an unchanged GDA is cataloged.

The GDA information (GDA signature) only changes when a GDA is modified. The GDA signature does not change when a GDA is (accidentally) cataloged but was not modified.

The signature of the GDA and the GDA signatures stored in all Natural objects referencing this GDA are compared at execution time, in addition to the time stamps of the objects.

ON	GDA signatures are stored and compared during execution. Natural only issues an error message if the signatures are not identical.
OFF	GDA signatures are not stored. 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=abbbbbbb` 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).

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.

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
```

MAXPREC – Maximum Number of Digits after Decimal Point

This option determines the maximum number of digits after the decimal point that the Natural compiler generates for results of arithmetic operations.

7,...,29	The value denotes the maximum number of digits after the decimal point that the Natural compiler generates for results of arithmetic operations.
	The default value 7 provides upwards compatibility for existing applications. If such applications are cataloged with MAXPREC=7, they will deliver the same results as before. Objects cataloged with a Natural version that did not support the MAXPREC option are executed as if MAXPREC=7 had been set.
	If higher precision is desired for intermediate results, the value should be increased.
	The setting of MAXPREC does not limit the number of digits after the decimal point that can be specified for user defined fields and constants. However, the precision of such fields and constants influences the precision of results of arithmetic operations. This makes it possible to benefit from enhanced precision in selected computations without having the need to set the compiler option MAXPREC to a value that unintentionally affects other computations. So even if MAXPREC=7 is in effect, the following example program can be cataloged and executed:
	```DEFINE DATA LOCAL 1 P (P1.15) END-DEFINE P := P + 0.1234567890123456 END```
	See also *Precision of Results of Arithmetic Operations* in the *Programming Guide*.

⚠ **Caution:** Changing the value of the MAXPREC option that is being used to catalog a Natural object may lead to different results, even if the object source has not been changed. See example below.

Example:

```
DEFINE DATA LOCAL
1 #R (P1.7)
END-DEFINE
#R := 1.0008 * 1.0008 * 1.0008
IF #R = 1.0024018 THEN ... ELSE ... END-IF
```

The value of #R after the computation and the execution of the IF statement depend on the setting of MAXPREC:

Setting of MAXPREC Effective at Compile Time	Value of #R	Executed Clause of IF Statement
MAXPREC=7	1.0024018	THEN clause
MAXPREC=12	1.0024019	ELSE clause

MEMOPT - Memory Optimization for Locally Declared Variables

This option determines whether or not memory is allocated for unused level-1 fields or groups defined locally (DEFINE DATA LOCAL).

ON	Storage is allocated only for ■ level-1 field, if the field or a redefinition thereof is accessed; ■ group, if the group or at least a group-field is accessed.
OFF	Data storage is allocated for all groups and fields declared locally. This is the default setting.

PCHECK - Parameter Check 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, function call, 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. **Error Control for PCHECK=ON** The parameter check is executed only when the object does not contain any syntax errors. The parameter check is executed for every object calling statement.

<table>
<tr>
<td></td>
<td>

The parameter check is controlled by the PECK profile parameter (see the *Parameter Reference* documentation).

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, functions, subprograms, external subroutines, help routines, 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.

</td>
</tr>
<tr>
<td>OFF</td>
<td>No parameter check is performed. This is the default setting.</td>
</tr>
</table>

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'.

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 parameter THSEP and 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.

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
```

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.

Compilation Options for Ensuring Version Compatibility (Remote Mainframe Environment)

- 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 Version 4.2 and 8.2 Syntax
- V42COMP - Disable Version 8.2 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

This option can be used to detect FIND statements containing a WITH clause that uses a multiple-value field in a search criterion for which the evaluation differs, depending on whether the Natural object has been cataloged with Natural Version 2.2, or with Natural Version 2.3 or above.

With Natural Version 2.3, the evaluation of search criteria for multiple-value fields in the WITH clause of the FIND statement has been changed.

As a consequence, if a Natural object cataloged with Natural Version 2.2 is recataloged with Natural Version 2.3 or above, execution of the Natural object may lead to different results if it contains a FIND statement using a WITH clause that uses a multiple-value field.

ON	Error NAT0998 will be returned for every `FIND` statement for which the evaluation of the search criterion has been changed.
OFF	No search for such `FIND` statements will be performed. This is the default value.

The evaluation of search criteria using multiple-value fields in the `WITH` clause of the `FIND` statement has been changed with Natural Version 2.3 so as to comply with the comparison evaluation in other statements (e.g. `IF`).

Four different variants of the search criterion in the `WITH` clause of a `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'`

    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 Natural Version 2.2, this statement returns records in which no occurrence of field `MU` has the value "A" (same as 4.). With Natural Version 2.3 or above, this statement returns records in which at least one occurrence of field `MU` does not have the value "A".

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

    With Natural Version 2.2, this statement returns records in which *at least one occurrence* of field `MU` has the value "A" (same as 1.). With Natural Version 2.3 or above, this statement returns records in which *every occurrence* of field `MU` has the value "A".

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

    This statement returns records in which *no occurrence* of `MU` has the value "A".

If you recatalog a Natural object that has been cataloged with Natural Version 2.2 and that contains `FIND` statements of variants 2. and 3. with Natural Version 2.3 or above, it will return different results.

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

In order to obtain the same results as with Natural 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 with `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 Version 4.2 and 8.2 Syntax

⚠️ **Important:** This compiler option will be available only with Natural Version 8.2 to allow a smooth transition. It will be removed again with a subsequent release of Natural after Version 8.2.

The compiler option `V41COMP` allows only functions and programming features which are compatible with Natural Version 4.1.

A number of functions and programming features introduced with Natural Version 4.2 or 8.2 would give rise to problems when a program developed and compiled with Version 4.2 or 8.2 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 or 8.2, 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. **Note:** `V41COMP=ON` automatically implies `V42COMP=ON`. Functions and programming features newly introduced with Natural Version 8.2 are controlled by `V42COMP`. If such version 8.2 features are found, the syntax error NAT0599 will be indicated.
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 features newly introduced with Version 4.2 (see also `V42COMP` with the new features in Version 8.2), 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`	possible	unknown	006

Function or Feature	Version 4.2	Version 4.1	Reason Code
▪ allowed when in structured mode (SM=ON)			

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

V42COMP - Disable Version 8.2 Syntax

⚠ **Important:** This compiler option will be available only with Natural Version 8.2 to allow a smooth transition. It will be removed again with a subsequent release of Natural after Version 8.2.

A number of functions and programming features introduced with Natural Version 8.2 would give rise to problems when a program developed and compiled with Version 8.2 is to be recompiled for putting into operation in a Version 4.2 environment. The relevant functions or features are listed **below**.

The `V42COMP` 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 8.2, every attempt to use a syntax construction that is supported by Version 8.2, but not by Version 4.2, is rejected and a NAT0599 syntax error and a corresponding reason code (see **below**) will be output.
OFF	A test for Version 4.2 compatibility is not performed. This is the default value.

Compilation Relevant Differences between Version 8.2 and 4.2

The following table gives an overview of the features newly introduced with Version 8.2, and indicates the reason code that will be supplied when incompatible syntax is detected:

Function or Feature	Version 8.2	Version 4.2	Reason Code

A variable of format P/N or a numeric constant with more than 7 precision digits was defined.
 	
Example:
 	
`DEFINE DATA LOCAL`
`  1 #P(P5.8) ↵`
	possible	not possible (NAT0169)	001

Size of a cataloged program exceeds 64 KB.	possible	not possible (NAT0881)	002
New object type "Function" was used, either by ■ a reference to a function; or ■ a `DEFINE PROTOTYPE` statement.	possible	unknown	003
New compiler option was referenced: ■ `MEMOPT` ■ `MAXPREC` Example: `OPTIONS MAXPREC=10`	possible	unknown	004
One of the new field options was used in an I/O statement (`WRITE`, `DISPLAY`, `PRINT`, `INPUT`): ■ `EMU=` ■ `ICU=` ■ `LCU=` ■ `TCU=` Example:	possible	unknown	005

Function or Feature	Version 8.2	Version 4.2	Reason Code
`DISPLAY #TEXT (LCU='<' TCU='>')`			
An arithmetic expression was used to specify the start-, end- or step-value in a FOR statement. Example: `FOR #I := (#J + 1) TO (#K - 1) STEP (#L + 1)`	possible	not possible	006
In a `MOVE ALL` statement, a `SUBSTR` option was applied for the source or target field. Example: `MOVE ALL 'X' TO SUBSTR(#A, 3, 5)`	possible	not possible	007
One of the following new system functions was used: ■ `*MAXVAL` ■ `*MINVAL` ■ `*TRIM` ■ `*TRANSLATE` Example: `COMPUTE #I := *MAXVAL(1, #J, #K) + 1`	possible	unknown	008

A `SUBSTR` option was applied
 	
■ for the selection field in a `DECIDE ON` statement, or
■ within a `VALUE(S)` clause
 	
Example:
 	
`DECIDE ON FIRST VALUE OF SUBSTR(#A, 2,3)`
`  VALUE SUBSTR(#VAL,1,3) ..`
	possible	unknown	009

Within an `EXAMINE` statement, ■ the `DIRECTION` clause or ■ the `POSITION` clause or ■ a multi-valued (array) field for the search value was used. Example:	possible	unknown	010

Function or Feature	Version 8.2	Version 4.2	Reason Code
`EXAMINE #TEXT FOR #ARR(1:3) GIVING NUMBER #I`			
The `ADJUST` option was used in a `READ WORK FILE` statement to auto resize a X-array field at access. Example: `READ WORK FILE 1 #XARR(*) AND ADJUST`	possible	unknown	011
The field referenced in the `REINPUT .. MARK` clause is supplied with a `(CV=..)` option. Example: `REINPUT 'text' MARK *#FLD (CV=#C)`	possible	unknown	012
System variables were referenced in the field list of a `WRITE WORK FILE` statement.	possible	unknown	013
Within a `READ` or `FIND` statement, ■ an `IN SHARED HOLD` clause or ■ a `SKIP RECORDS IN HOLD` clause was used.	possible	unknown	014
New statements: `READLOB` and `UPDATELOB`.	possible	unknown	015

9 DEBUG

```
DEBUG object-name
```

This command is used to invoke the Natural debugger. With the command, you specify the name of the object to be debugged.

See the *Debugger* documentation for detailed information on the debugger.

> **Note:** This command is not executable in batch mode.

10 EDIT

This command is used to invoke a Natural editor for the purpose of editing the source form of a Natural programming object.

Three different forms of command syntax exist. These are documented in the following sections.

Related command: READ.

See also *Object Naming Conventions* in the *Using Natural Studio* documentation.

See also *Invoking Editors* in *Using Natural Studio*.

Syntax 1

```
EDIT [object-type] [object-name [library-id]]
```

object-type

The following object types can be edited:

```
      {  CLASS    }
      {  4        }
      COPYCODE
      {  DIALOG   }
      {  3        }
      GLOBAL
      HELPROUTINE
      LOCAL
      MAP
      PARAMETER
      PROGRAM
      {  SUBPROGRAM }
      {  N          }
      SUBROUTINE
      TEXT
      VIEW
      7 (for Function)
```

Which editor is invoked depends on the type of object to be edited:

■ Local data areas, global data areas or parameter data areas are edited with the data area editor.

- Maps are edited with the map editor.

- Dialogs are edited with the dialog editor.

- Classes are edited with the Class Builder.

- `EDIT VIEW` only works in the current library and when an *object-name* is specified. If the object to be viewed is a DDM, the DDM editor is invoked.

- All other types of objects - program, subprogram, subroutine, `7` (for function), helproutine, copycode, text, description - are edited with the program editor.

  Note: The text object "description" is available on mainframes only. A description is a program description as stored and maintained in the Predict Data Dictionary; an object of this type can only be edited if Predict is installed.

The object types are described in the *Programming Guide*. The editors are described in the *Editors* documentation.

If you specify the name of the object you wish to edit, you need not specify its object type.

object-name

With the `EDIT` command, you specify the name of the object you wish to edit. The maximum length of the object name is 8 characters.

  Note: For DDMs, the maximum length is 32 characters.

Natural will then load the object into the edit work area of the appropriate editor and set the object name for a subsequent `SAVE`, `CATALOG`, `STOW` command.

If you do not specify an *object-name* and there is no object in the source work area, the empty program editor screen will be invoked where you can create a program. If the source work area is not empty, the object will be loaded in the appropriate editor.

  Note: If the source work area is not empty and contains an object that has been opened in an editor session, the corresponding editor window is displayed and gets the input focus. Any changes that are applied to the source work area in the meantime (for example, by running Natural programs) will not be displayed.

library-id

If the object you wish to edit is not contained in the library you are currently logged on to, you must specify the *library-id* of the library in which the object to be edited is contained.

If Natural Security is active, a *library-id* must not be specified, which means that you can only edit objects which are in your current library.

Syntax 2

```
EDIT [    *       ]{    *       }
     [ object-type ]{ object-name }
```

If you do not remember the name of the object you wish to edit, you can use this form of the EDIT command to display a list of objects, and then select from the list the desired object.

EDIT *	displays a list of all objects in your current library.
EDIT object-type *	displays a list of all objects of that type in your current library.

To select an object from a certain range of objects, you can use asterisk notation and wildcard notation for the *object-name* in the same manner as described for the system command LIST.

Syntax 3

```
EDIT FUNCTION subroutine-name
```

The EDIT FUNCTION command may be used to edit a subroutine using the subroutine name (not the object name) with maximally 32 characters.

> **Note:** Please note that the keyword FUNCTION used in this syntax is not identical with the Natural **object type** 7 (for function) listed above. See the description of object type Function in the *Programming Guide*.

Example:

```
DEFINE SUBROUTINE CHECK-PARAMETERS
  ...
END-SUBROUTINE
END
```

Assuming that the above subroutine has been saved under the object name CHCKSUB, you may edit subroutine CHECK-PARAMETERS either by issuing the command:

```
EDIT S CHKSUB
```

or by

```
EDIT F CHECK-PARAMETERS
```

11 EXECUTE

```
{ EXECUTE  [REPEAT]     program-name       [library-id] }
{ program-name [parameter ...]                          }
```

This command is used to execute a Natural object module of type program or of type dialog. The object module must have been cataloged (that is, stored in object form) in the Natural system file or linked to the Natural nucleus. The execution of an object module does not affect the source code currently in the editor work area.

See also *Executing Objects* in *Using Natural Studio*.

Syntax Explanation

EXECUTE	The keyword EXECUTE is optional; it is sufficient to specify *program-name*, i.e. the name of the program or dialog to be executed.
REPEAT	If the program or dialog being executed produces multiple screen output and you wish the screens to be output one after another without intervening prompts, you specify the keyword REPEAT together with the keyword EXECUTE.
program-name	The name of the program or dialog to be executed. If you do not specify a *library-id*, Natural can only execute the specified program or dialog if it is stored either in your current library or in the current steplib library (the default steplib is SYSTEM).
library-id	If the program or dialog is stored in another library, specify the *library-id* of that library. In this case, the program or dialog can only be executed if it is actually stored in the specified library. A *library-id* that begins with SYS must not be specified (except SYSTEM).
parameter	When you execute a program by specifying the program name without the keyword EXECUTE, you may pass parameters to the program. These parameters will be read by the first INPUT statement in the executed program. You can specify the parameters as positional parameters or as keyword parameters, with the individual specifications separated from one another by blanks or the input delimiter character (as specified with the session parameter ID). **Note:** If one of the parameters passed contains blanks or is a string which contains blanks, the transfer will only be executed if directly after the program name an input delimiter is set.

Examples of EXECUTE Command

```
EXECUTE PROG1
```

```
EXECUTE PROG1 ULIB1
```

```
PROG1
```

```
PROG1 VALUE1 VALUE2 VALUE3
```

```
PROG1 VALUE1, VALUE2, VALUE3
```

```
PROG1 PARM1=VALUE1, PARM2=VALUE2, PARM3=VALUE3
```

```
PROG1 PARM3=VALUE3 PARM1=VALUE1 VALUE2
```

```
PROG1,ab cd ef,gh,de fg,ab
```

12 FIN

```
FIN
```

This command is used to terminate a Natural session. It applies to online sessions as well as batch mode sessions.

A batch mode session is also terminated when an end-of-file condition is detected in the command input dataset.

13 GLOBALS

```
GLOBALS [parameter=value ...]
```

This command is used to set Natural session parameters.

> **Note:** In batch mode, this command is only executable, if parameters are specified. For example, `GLOBALS SM=ON` can be executed in batch mode.

Syntax Explanation

GLOBALS	If the `GLOBALS` command is entered without parameters, a window appears where you can modify the parameter settings. For detailed information on this window, see *Using Session Parameters* in *Using Natural Studio*.
parameter	Parameter settings can be supplied in any order and must be separated by a blank.
	If more parameters are specified than will fit on one command line, several `GLOBALS` commands must be issued.
	Example:
	`GLOBALS DC=, ID=.`
	Note: Certain session parameters can be modified only using the above mentioned window, but not via the command line.

List of Parameters

The following table contains a list of session parameters that can be specified with the system command `GLOBALS`.

Parameters	Function
CF	Character for Terminal Commands
COMPR	Set RPC Buffer Compression
CPCVERR	Code Page Conversion Error
DBSHORT	Interpretation of Database Short Names
DC	Character for Decimal Point Notation
DFOUT	Date Format for Output
DFSTACK	Date Format for Stack
DFTITLE	Output Format of Date in Standard Report Title
DU	Dump Generation
EJ	Page Eject

Parameters	Function
ENDIAN	Endian Mode for Compiled Objects
FCDP	Filler Character for Dynamically Protected Input Fields
FS	Default Format/Length Setting for User-Defined Variables
GFID	Global Format IDs
IA	Input Assign Character
ID	Input Delimiter Character
IM	Input Mode
LE	Reaction when Limit for Processing Loop Exceeded
LS	Line Size
LT	Limit for Processing Loops
ML	Position of Message Line
NC	Use of Natural System Commands
OPF	Overwriting of Protected Fields by Helproutines
PM	Print Mode
PS	Page Size for Natural Reports
REINP	Issue Internal REINPUT Statement for Invalid Data
SA	Sound Terminal Alarm
SF	Spacing Factor
SM	Programming in Structured Mode
SYMGEN	Generate Symbol Table
THSEPCH	Thousands Separator Character
TIMEOUT	Wait Time for RPC Server Response
TRYALT	Try Alternative Server Address
WH	Wait for Record in Hold Status
XREF	Creation of XRef Data for Natural
ZD	Zero-Division Check
ZP	Zero Printing

Interaction with SET GLOBALS and Other Statements

Statement SET GLOBALS

The system command `GLOBALS` and the statement `SET GLOBALS` offer the same parameters for modification. They can both be used in the same Natural session. Parameter values specified with a `GLOBALS` command remain in effect until they are overridden by a new `GLOBALS` command or `SET GLOBALS` statement, the session is terminated, or you log on to another library.

Other Statements Influencing the Session Parameter Settings

Some parameter values may be overridden during program execution using the `LIMIT`, `EJECT`, and `FORMAT` statements and by format entries specified in `INPUT`, `DISPLAY`, `PRINT`, and `WRITE` statements.

14 HELP

```
{ HELP }  [ [NAT]nnnn      ]
{ ?    }  [ USER[nnnn]     ]
```

This command is used to invoke online help for error messages.

> **Note:** This command is not executable in batch mode.

For further information, see *Using Help* in *Using Natural Studio*.

HELP	Displays the help menu.
HELP [NAT]nnnn	Entering HELP and a number (up to 4 digits, optionally prefixed by "NAT") displays the detailed message text for the Natural error condition associated with that number, that is, the long text of the Natural system error message NATnnnn.
HELP USERnnnn	Displays the long text of the library-specific error message number nnnn in the current library.

15   INPL

```
INPL [R]
```

This command is used to invoke the Natural `INPL` utility. This utility is *only* used for the loading of Software AG installation datasets into the system files as described in the online help and in the platform-specific installation documentation.

Apart from that, you use the Object Handler to load objects into the system files.

INPL	If you enter the `INPL` command without any parameters, the `INPL` utility will be invoked.
INPL R	Invokes the `INPL` utility function Natural Security Recover which is only available if Natural Security is installed.
	It can be used to reset the access to the Natural Security library `SYSSEC`: the user DBA, the library `SYSSEC`, and the link between the two will be redefined as after the initial installation, while all other links to `SYSSEC` will be cancelled. See also *Inaccessible Security Profiles* in the section *Countersignatures* of the *Natural Security* documentation.

For further information, see *INPL Utility* in the *Tools and Utilities* documentation.

16 LAST

```
      ┌       ┐
      │  -    │
LAST  │  -nn  │
      │  *    │
      └       ┘
```
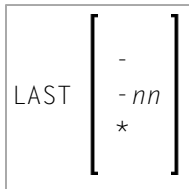
This command is used to display the system command(s) that was/were last executed. Moreover, you can have the displayed command(s) executed again. You can also overwrite them before they are executed.

Only system commands that you actually entered can be displayed via the LAST command; commands issued internally by Natural as a result of a command you entered are not available via LAST.

LAST	The command that was issued last is placed in a dialog box and can be executed.
LAST -	The command that was issued last is placed in a dialog box and can be executed.
	If you enter LAST - again, the last but one command is placed in a dialog box.
	By repeatedly entering LAST -, you can thus "page" backwards command by command.
LAST -nn	Natural "remembers" up to the last 20 commands that were issued; nn must therefore not be greater than 20.
	The last command but nn is placed in a dialog box and can be executed.
LAST *	A dialog box is displayed showing the last 20 system commands that were issued.
	■ To execute the commands again, select the required commands and use the **Copy** button to copy the commands to the **Selected Commands** list box.
	■ The selected commands in the list box can be modified before executing them.

17 LASTMSG

```
LASTMSG
```

This command is used to display additional information about the error situation which has occurred last.

> **Note:** This command is also available in a remote session. All information can be read in batch mode.

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 command `LASTMSG`, 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.

See also *Last Message* in *Using Natural Studio*.

▶ **To display information on the corresponding error**

■ Select one of these messages and choose the **Details** button, or double-click the message.

  The following is displayed:

  ■ 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 (system = error issued by Natural; user = error issued by user application);

- error type (runtime, syntax, command execution, session termination, program termination, remote procedure call);
- date and time of the error.

> **Note:** The library `SYSEXT` contains a user application programming interface `USR2006` which enables you to display in your Natural application the error information supplied by `LASTMSG`.

Natural Remote Procedure Call (RPC):

If an error occurs on the server, the following error information is not displayed: database ID, file number, date and time.

18 LIST

This system command is used to display the source code of a single object or to list one or more objects which are contained in the current library.

Note: This command is not executable in batch mode.

This chapter covers the following topics:

See also *Listing Objects* in *Using Natural Studio*, and the descriptions of the commands LIST XREF and LIST COUNT.

Syntax Overview

```
       ┌ ┌ ┌                                   ┐ ┐
       │ │ │ [object-type] object-name         │ │
       │ │ │ DIRECTORY [object-name]           │ │
LIST   │ │ ⟨ VIEW [view-name]                  ⟩ │
       │ │ │ RESOURCE [name]                   │ │
       │ │ │ ERROR [name]                      │ │
       └ └ └                                   ┘ ┘
```

object-type

```
     *
             {   CLASS   }
             {   4       }
      COPYCODE
      DATA-AREAS
          GLOBAL
          LOCAL
          PARAMETER
             {   DIALOG  }
             {   3       }
      7 (for function)
      8 (for adapter)
      MAP
             {   PROCESSOR }
             {   CP        }
             {   5         }
      PROGRAM
      ROUTINES
          HELPROUTINE
             {   SUBPROGRAM }
             {   N          }
          SUBROUTINE
       TEXT
```

object-name

In place of *object-name*, you may specify the name of an object (8 characters long at maximum). You may also specify asterisk notation (*), see the **examples** below.

Displaying an Individual Source

LIST	If you enter only the LIST command itself, without any parameters, the contents of the source of the object currently selected will be listed.
LIST *object-name* LIST *object-type object-name*	If you enter a single object name with the LIST command, you need not specify the *object-type*. If you specify an *object-type*, you must also specify an *object-name*.

	In both cases, the object's source code will be listed.

Displaying a List of Objects

LIST *object-name* LIST *object-type object-name*	To have all objects in the current library listed, except DDMs, you specify an asterisk (*) for the *object-name*, but no *object-type*. To have all objects of a certain type listed, you specify a certain *object-type* and an asterisk (*) for the *object-name*. If you wish a certain range of objects to be listed, you can use asterisk notation (*) for the *object-name* and/or wildcard notation (?).

Examples

■ List all objects in the current library, except DDMs:

```
LIST *
```

■ List all subroutines in the current library:

```
LIST S *
```

■ List all objects (of any type) whose names begin with SYS:

```
LIST SYS*
```

■ List all maps whose names begin with SYS:

```
LIST M SYS*
```

■ List directory information of object PRG01 in current library:

```
LIST DIR PRG01
```

■ List all objects such as NATAL, NATURAL, NAT*vr*AL (where *vr* represents the relevant product version):

```
LIST N?T*AL
```

Displaying Directory Information

LIST DIRECTORY	Displays the directory information about the last active object currently in the source work area: ■ **Source code:** "Saved-on" date and time, library name, user ID, programming mode (reporting or structured), Natural version, code page information (if available), operating system, size, encoding. ■ **Object code:** "Cataloged-on" date and time, library name, user ID, programming mode, Natural version, code page information (if available), operating system/version, size, Endian mode. Directory information on the saved source code cannot be always exact, because the source code can be edited with non-Natural editors which are not under the control of Natural.
LIST DIRECTORY *object-name*	Displays the directory information about the specified object. If you use asterisk notation (*) for *object-name*, the directory information of the existing objects is displayed sequentially.

> **Note:** The code page information displayed shows the first 32 characters of the code page only.

Displaying Views

LIST VIEW	Displays a list of all views (DDMs).
LIST VIEW *view-name*	If you specify a single view name, the specified view will be displayed. For the *view-name*, you can use asterisk notation to display a list of all views (*) or a certain range of views (for example: A*).

Displaying File Information of Resource Objects

`LIST RESOURCE` *name*	Displays the file information about the specified resource object. For *name*, you may only use asterisk notation (*).

Example - Display the file information of all resource objects whose name starts with a W:

```
LIST RESOURCE W*
```

Displaying File Information of Error Message Containers

`LIST ERROR` *name*	Displays the file information about the specified error message container `N`*nn*`APMSL.MSG`, where *nn* is the language code. For *name*, you may only use asterisk notation (*).

19 LIST COUNT

```
LIST COUNT
```

This command is used to list the number of Natural objects in your current library.

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

> **Note:** If there are objects listed under object type `undefined`, this indicates that the library contains objects whose version is not compatible.

20 LIST XREF

```
LIST XREF
```

This command is only available if Predict has been installed.

It is used to display all active cross-reference data for the current library.

For further information, see *List XREF For Natural* in the Predict documentation.

21 LOGOFF

```
LOGOFF
```

Related command: LOGON.

This command is used to cause the library ID to be set to SYSTEM and the Adabas password to be set to blanks. The contents of the source program work area are not affected by this command.

LOGOFF has no effect on Natural global parameter settings.

For information on LOGOFF processing under Natural Security, see *How to End a Natural Session* in section *Logging On* of the *Natural Security* documentation.

> **Note:** LOGOFF does *not* cause the Natural session to be terminated.

▶ **To terminate the session**

■ Use the system command FIN, or execute a program that contains a TERMINATE statement.

22 LOGON

```
LOGON library-id [password]
```

Related command: LOGOFF.

This command is used to log on to a library in your environment or create a new library. In the specified library, all newly created source or object programs saved during the session will be stored (unless you explicitly specify another library ID in a SAVE, CATALOG or STOW command).

The LOGON command has no direct effect on the source program in the currently active window.

LOGON causes all Natural global data areas and application independent variables (AIVs), all assignments made using the SET KEY statement and retained ISN lists to be released. Data definition modules (DDMs) contained in the DDM buffer area are also released.

LOGON library-id	The library ID can be 1 to 8 characters long and must not contain blanks. It can consist of the following characters:	
	A - Z	upper-case alphabetical characters
	0 - 9	numeric characters
	-	hyphen
	_	underscore
	The first character of a library ID must be an upper-case alphabetical character.	
LOGON library-id password	The Adabas password; see *Session Parameters* in section *Library Maintenance* of the *Natural Security* documentation.	

For information on LOGON processing under Natural Security, see *Logging On* in the *Natural Security* documentation.

23 MAIL

```
MAIL [{ *
        ?
        mailbox-id }]
```

This command is used to invoke a mailbox which is a kind of "notice board" used to broadcast messages under Natural Security. The contents and/or expiration date of the mailbox can be modified.

MAIL	If you enter the `MAIL` command without any parameters, a window is displayed prompting you to enter a mailbox ID.
MAIL *	A list of all mailboxes you may use is displayed, and you may then select a mailbox from the list.
MAIL ?	
MAIL *mailbox-id*	If you specify a *mailbox-id* (maximum 8 characters), the corresponding mailbox is invoked directly. The *mailbox-id* must have been defined in Natural Security.

For further information, see *Mailboxes* in the *Natural Security* documentation.

24 MAP

The `MAP` command enables you to perform the following functions, using the Natural command line:

This chapter covers the following topics:

Related command: `UNMAP`.

Establish a Connection to a Natural Development Server Environment

The following `MAP` command syntax applies if you want to establish a connection to a Natural Development Server, using the Natural command line:

```
MAP ENVIRONMENT=environment-name server-name port-name[userid[password['parm=value;...']]]
```

This method has the same effect as the dialog described in the section *Mapping a Development Server* in the *Remote Development Using SPoD* documentation.

environment-name	Alias name used for the connection. If *environment-name* is not specified, an alias name in the form *server(port)* will be generated. If the environment name contains blanks, it must be enclosed in single quotes ('...').
server-name	The name of the Natural development server on the mainframe, UNIX or OpenVMS server.
port-name	The TCP/IP port number of the development server.
userid	The user ID for access to the development server. If you enter an asterisk (*) as *userid*, the user ID of the client session is used.
password	If Natural Security is installed on the development server, specify the required password. If you enter an asterisk (*) as *password*, an empty password string is sent to the development server.
parm	If dynamic parameters are required for your development server, specify the session parameters using single quotes ('...').

To unmap a session on a Natural Development Server, you can use the `UNMAP` command.

Establish a Connection to a Natural Application

The following `MAP` command syntax applies if you want to establish a connection to a Natural application, using the Natural command line:

```
MAP APPLICATION=application-name[userid[password]
```

This method has the same effect as the dialog described in the section *Mapping and Unmapping Applications* in the *Remote Development Using SPoD* documentation. For information on the term "Natural Application", refer to the Natural Single Point of Development documentation.

application-name	The name of the application.
userid	The user ID for access to the application. If you enter an asterisk (*) as *userid*, the user ID of the client session is used.
password	If Natural Security is installed on the development server, specify the required password. If you enter an asterisk (*) as *password*, an empty password string is sent to the development server.

To unmap an application on a Natural Development Server, you can use the `UNMAP` command or the dialog described in the section *Mapping and Unmapping Applications*.

25 PROFILE

This command is available only if Natural Security is installed.

```
PROFILE
```

This command is used to display the security profile currently in effect. This profile informs you of the conditions of use in effect for you in your current Natural environment.

For further information, see *PROFILE Command* in the *Natural Security* documentation.

26 PURGE

```
PURGE [object-name ...]
```

This command is used to delete one or more source objects.

> **Note:** If the Natural profile parameter `RECAT` is set to `ON`, the `PURGE` command will be rejected for a source for which a corresponding cataloged object exists.

PURGE	If you enter the `PURGE` command without an *object-name*, a list of all objects in the current library will be displayed; on the list, you can then mark the objects to be deleted.
PURGE *object-name*	As *object-name*, you specify the name(s) of the object(s) to be deleted. You can only delete objects that are stored in the library to which you are currently logged on. If you wish to delete all objects whose names begin with a specific string of characters, use asterisk notation (*) for the *object-name*.

27 READ

```
READ object-name [library-id]
```

Related command: EDIT.

This command is used to transfer an object that is stored in source form into the source work area. Any object currently in the source work area will be overwritten by the object read.

See also *Object Naming Conventions* in the *Using Natural Studio* documentation.

READ *object-name*	The name of the object to be read.
	If *object-name* is specified without a library ID, the object will be read only if it is stored in the library to which you are currently logged on.
READ *object-name* *library-id*	The library in which the object to be read is contained.
	If both *object-name* and *library-id* are specified, Natural will only read the object if it is stored under the specified library ID.

28 REGISTER

```
REGISTER { class-module-name }  [ { library-name } [ { ES } ] ] ]
         { *                 }    [ { *          } [ { IM } ] ] ]
                                                   [ { EM } ] ] ]
```

Related command: UNREGISTER.

This command is used to register Natural classes. They are registered for the server ID under which Natural was started.

For further information, see *The REGISTER Command* in the *Administrating NaturalX Applications* part of the *Operations* documentation.

29 RENAME

This command is not available via the command line in a remote development environment.

```
RENAME [old-name [new-name [new-type]]]
```

This command is used to give a Natural programming object another name. In addition, you can change the object type.

You can only rename one object at a time. The object to be renamed must be stored in the library to which you are currently logged on. To ensure consistency, Natural will rename source code or object module or both.

See also *Object Naming Conventions* in the *Using Natural Studio* documentation.

RENAME	If you issue the command without parameters, a **Rename Object** window appears where you can specify the same parameters as in the command line.
old-name	As old-name you specify the existing name of the object to be renamed.
new-name	As new-name you specify the name under which the object is to be stored from now on.
new-type	When you rename an object in source form, you can also change its object type by specifying the corresponding character for new-type. The possible values you can specify for new-type are:

3	Dialog
4	Class
5	Processor
7	Function
8	Adapter
9	Resource
A	Parameter data area
C	Copycode

	G	Global data area
	H	Helproutine
	L	Local data area
	M	Map
	N	Subprogram
	O	Macro
	P	Program
	S	Subroutine
	T	Text
	Y	Rule
	Z	Recording

See also *Renaming Objects* in *Using Natural Studio.*

30 RENUMBER

`RENUMBER [(n)]`

This command is used to renumber the lines in the source program currently in the source work area.

`RENUMBER`	If you enter the command without parameter, the increment to be used for renumbering is 10.
`RENUMBER (n)`	*n* can be used to specify a value between 1 and 10 as increment for renumbering.

31   RETURN

```
RETURN [ { I    } ]
        {  nn  }
        {  *   }
```

This command is used to return to a previous (or initial) Natural application.

RETURN	If `RETURN` is specified without any parameters, control will be returned to the previous application (as defined with the system command `SETUP`). All information about this previous application will be deleted. If no previous application exists, control is returned to the initial application.
	If `RETURN` is issued and no return point is set, the `RETURN` command will be ignored.
	Under Natural Security:
	A `LOGOFF` command will be executed if `RETURN` is issued and no return point has been set.
`RETURN I`	This command causes control to be returned directly to the initial application. This option also causes Natural to delete all definitions of previous applications (except that of the initial application).
`RETURN nn`	This command causes control to be returned to the *nn*th previous application. When this option is used, all information for applications subsequent to the *nn*th application is deleted.
`RETURN *`	This command will display a list of all return points which are currently set up. On the list you may then select the return point to which you wish to return.

See the `SETUP` command for further information and examples.

32   RPCERR

```
RPCERR
```

This command is used to display the last Natural error number and message if it was RPC related, and it also displays the last Broker reason code and associated message. Additionally, the node and server name from the last Broker call can be retrieved.

For further information, see *Monitoring the Status of an RPC Session* in the *Operating a Natural RPC Environment* section of the *Natural Remote Procedure Call (RPC)* documentation.

33 RUN

```
RUN [REPEAT] [program-name [library-id]]
```

This command is used to compile and execute a source program or dialog. The program or dialog may be in the source work area or in the Natural system file.

See also:

- *Running Objects* in *Using Natural Studio*
- *Object Naming Conventions* in *Using Natural Studio*

RUN	If `program name` is not specified, Natural will compile and execute the program or dialog currently residing in the work area.
REPEAT	`REPEAT` defines that if the program or dialog being executed produces multiple screen output, the screens are to be output one after another without intervening prompting messages. When the program or dialog terminates, Natural will enter command mode.
program-name	The name of the program or dialog to be run. If `program-name` is specified without a library ID, Natural will read the source program or dialog into the source work area, compile, and execute the specified program or dialog only if it is stored under the current library ID. If it is not stored under the current library ID, an error message will be issued.
library-id	The library in which the program or dialog to be run is contained. If both `program-name` and `library-id` are specified, Natural will retrieve, compile, and execute the specified program or dialog only if it is stored under the library ID specified. If it is not stored under the current library ID, an error message will be issued. The setting for `library-id` must not begin with `SYS` (except `SYSTEM`).

34 SAVE

```
SAVE [object-name [library-id]]
```

Related commands: STOW | CATALOG.

This command is used to save the source code of the programming object currently in the work area of the editor and store it as a source object in the Natural system file.

See also:

- *Saving Objects* in *Using Natural Studio*
- *Object Naming Conventions* in *Using Natural Studio*

⚠ **Caution:** The SAVE command cannot be used if the profile parameter RECAT has been set to ON; in this case, use the STOW command to compile and store the object.

SAVE	If you use the command without *object-name*, the current source object in the source work area will be saved in the library from which the object was read into the source work area (for example, with EDIT or READ). An existing source code will be replaced.
SAVE *object-name*	A new source object is created. As *object-name*, you specify the name under which the source object is to be saved. The new source object is stored in the current library. If the source object exists, the command is rejected.
SAVE *object-name* *library-id*	When you save a source object under a different name or save a newly created object, the source object will, by default, be stored in the current library. If you wish to store it in another library, you have to specify the desired *library-id* after the *object-name*. A new source object is created, if the source object exists, the command is rejected.

35 SCAN

```
SCAN
```

This command invokes a dialog box which is used to find Natural objects and character strings within these objects. For detailed information on this dialog box, see *Finding Objects in a Library* in *Using Natural Studio*.

> **Note:** This command is not executable in batch mode.

36 SCRATCH

$$\underline{SCRA}TCH \quad \left[\left\{ \begin{array}{l} * \\ \textit{object-name}... \end{array} \right\}\right]$$

This command is used to delete one or more objects - in both source and object form. The contents of the source work area is not affected.

SCRATCH SCRATCH *	If you enter the SCRATCH command without an *object-name* or without an *object-name* but with an asterisk (*), a list of all objects or all selected objects in the current library will be displayed. On the list you may then mark the objects to be deleted.
SCRATCH *object-name*	As *object-name*, you specify the name(s) of the object(s) to be deleted. You can only delete objects which are stored in your current library. If you wish to delete all objects whose names begin with a specific string of characters, use asterisk notation (*) for the *object-name*.

> **Note:** If an FDIC system file is specified in the parameter file which is not valid, Natural will display an appropriate error message when the SCRATCH command is issued.

37 **SETUP**

```
SETUP [application-name] [command-name] [I]
```

This command is used to define applications to which control is to be returned using the `RETURN` command. This allows you to easily transfer from one application to another during a Natural session.

This chapter covers the following topics:

Syntax Explanation

The command syntax and the parameters that can be issued with the `SETUP` system command are explained below. If a parameter is to be omitted, you may use the input delimiter character to mark the beginning of the following parameter(s).

`SETUP`	If `SETUP` is issued without parameters, a menu will be displayed for the purpose of entering the command information.
`application-name`	The name of the application to which control is to be returned. A maximum of 8 characters may be used (A8).
	If `application-name` is blank, a `LOGON` command will not be issued. This permits multiple return points within the same application.
	If `application-name` is "*", the current setting of the system variable `*LIBRARY-ID` (that is, at the time `SETUP` is issued) is used to create the `LOGON` command when `RETURN` is issued.
`command-name`	The name of the command which is to be executed when control is returned to the application. A maximum of 60 characters may be used (A60).
	If `command-name` is blank, no command will be issued after the `LOGON`. This is useful for applications under Natural Security for which a startup program has already been defined.
	If `command-name` is "*", the current setting of the system variable `*STARTUP` (that is, at the time `SETUP` is issued) is used as the startup command when `RETURN` is issued.
`I`	If the `I` option is specified, all return points defined with previous `SETUP` commands will be deleted and the application specified with `SETUP I` will be defined as the new initial application.
	In a non-Security environment, if you log on from library `SYSTEM` to another library and no return point has been set, this other library will automatically be set as initial return point.

SETUP/RETURN Example

1. User starts Natural session (default application is APPL1).

    Return point APPL1 is defined on Level 1.

2. User issues command LOGON APPL2.

3. User executes a program which stacks two commands (establish return point and go to another application):

    ```
    SETUP *,MENU
    LOGON APPL3
    ```

    Return point APPL2, STARTUP MENU is defined on Level 2.

4. User issues command LOGON APPL4 (user selects another application).

5. User presses a PF key which has the setting RETURN. Natural will issue for the user:

    ```
    LOGON APPL2
    MENU
    ```

    Return to APPL2, delete Level 2.

6. User executes a program which stacks:

    ```
    SETUP *,MENU
    LOGON APPL5
    ```

    Return point APPL2, STARTUP MENU is defined on Level 2.

7. User executes a program which stacks:

    ```
    SETUP *,MENU
    LOGON APPL6
    ```

    Return point APPL5, STARTUP MENU is defined on Level 3.

8. User executes a program which stacks:

    ```
    SETUP *,MENU
    LOGON APPL7
    ```

    Return point APPL6, STARTUP MENU is defined on Level 4.

9. User executes a program which stacks:

```
SETUP *,MENU
LOGON APPL8
```

Return point `APPL7, STARTUP MENU` is defined on Level 5.

10. User executes a program which stacks:

```
SETUP *,MENU
LOGON APPL9
```

Return point `APPL8, STARTUP MENU` is defined on Level 6.

11. User issues command `RETURN 2` (return two levels back).

Natural will return user to `APPL7`, since that was the second previous session (all information for `APPL8` is now lost). Level 6 (`APPL8`) is deleted, Level 5 (`APPL7`) is activated and level deleted.

12. User issues command `RETURN`.

Level 4 (`APPL6`) is activated, level deleted. Natural will return user to `APPL6`, since that was the session previous to `APPL7`.

13. User issues command `RETURN`.

Level 3 (`APPL5`) is activated, level deleted. Natural will return user to `APPL5`, since that was the session previous to `APPL6`.

14. User issues command `RETURN I`.

Level 2 (`APPL2`) is deleted, Level 1 (`APPL1`) is activated.

38 STOW

```
STOW [object-name [library-id]]
```

Related commands: SAVE | CATALOG.

This command is used to compile and store a Natural programming object (in both source and object form) in the Natural system file. You can regard this command as a CATALOG followed by a SAVE.

See also:

- *Stowing Objects* in *Using Natural Studio*
- *Object Naming Conventions* in *Using Natural Studio*

STOW	If you use the command without *object-name*, the current source object in the source work area and the generated code are stored in the library under the name of the object last read into the source work area (for example, with EDIT or READ).
STOW *object-name*	Use this command syntax to store a new object (source and generated code) named *object-name* in the current library. If the object exists in either source or cataloged form, the command is rejected.
STOW *object-name library-id*	If both *object-name* and *library-id* are specified, a new object will be created and stored under that name in the specified library ID. If the object exists in either source or cataloged form, the command is rejected.

> **Note:** If an FDIC system file is specified in the parameter file which is not valid, Natural will display an appropriate error message when the STOW command is issued.

39 STRUCT

```
STRUCT [(n)]
```

This command is used to perform structural indentation of the source code of the programming object currently in the work area of the editor.

STRUCT	By default (that is, if (n) is not specified), indentation is by 2 positions.
STRUCT (n)	The parameter (n) may be supplied to specify the number of spaces used for indentation. Possible values: 1 - 9. Example: `STRUCT (5)`

The following types of statements are affected by the STRUCT command:

- processing loops (READ, FIND, FOR, etc.),
- conditional statement blocks (AT BREAK, IF, DECIDE FOR, etc.),
- DO/DOEND statement blocks,
- DEFINE DATA blocks,
- inline subroutines.

This chapter covers the following topics:

Indentation of Source Code Lines

You can have a source program indented so that the indentation of source-code lines reflects the structure of the program.

> **Note:** Indentation is performed differently for a reporting-mode program than for a structured-mode program.

Partial Indentation

You can exclude sections of your program source from structural indentation by using the special statements /*STRUCT OFF and /*STRUCT ON. These must be entered at the beginning of a source-code line. The source-code lines between these two statements will remain as they are when you issue the STRUCT command.

Example of Structural Indentation

Program before being structurally indented:

```
DEFINE DATA LOCAL
1 EMPL VIEW OF EMPLOYEES
2 PERSONNEL-ID
2 FULL-NAME
3 FIRST-NAME
3 NAME
1 VEHI VIEW OF VEHICLES
2 PERSONNEL-ID
2 MAKE
END-DEFINE
FIND EMPL WITH NAME = 'ADKINSON'
IF NO RECORDS FOUND
WRITE 'NO RECORD FOUND'
END-NOREC
FIND (1) VEHI WITH PERSONNEL-ID = EMPL.PERSONNEL-ID
DISPLAY EMPL.PERSONNEL-ID FULL-NAME MAKE
END-FIND
END-FIND
END
```

The same program after being structurally indented:

```
DEFINE DATA LOCAL
1 EMPL VIEW OF EMPLOYEES
  2 PERSONNEL-ID
  2 FULL-NAME
    3 FIRST-NAME
    3 NAME
1 VEHI VIEW OF VEHICLES
  2 PERSONNEL-ID
  2 MAKE
END-DEFINE
FIND EMPL WITH NAME = 'ADKINSON'
  IF NO RECORDS FOUND
    WRITE 'NO RECORD FOUND'
  END-NOREC
  FIND (1) VEHI WITH PERSONNEL-ID = EMPL.PERSONNEL-ID
    DISPLAY EMPL.PERSONNEL-ID FULL-NAME MAKE
  END-FIND
END-FIND
END
```

40 SYSAPI

```
SYSAPI
```

This command is used to invoke the `SYSAPI` utility.

This utility is used to locate Application Programming Interfaces (APIs) provided by Natural add-on products such as Entire Output Management (NOM).

For each API, the utility `SYSAPI` provides one or more example programs that contain a functional description of the API and that can be used to test the effect of the API.

For further information, see *SYSAPI - APIs of Natural Add-on Products* in the *Tools and Utilities* documentation.

41   SYSCP

```
SYSCP
```

This command is used to invoke the `SYSCP` utility.

The `SYSCP` utility can be used to obtain code page information.

For further information, see *SYSCP Utility - Code Page Information* in the *Tools and Utilities* documentation and *Unicode and Code Page Support*.

42 SYSERR

```
SYSERR
```

This command is used to invoke the `SYSERR` utility.

With the `SYSERR` utility, you can write your own application-specific messages.

- You can use the `SYSERR` utility to separate error or information messages from your Natural code and manage them separately.

- As well as unifying messages and defining message ranges for different kinds of messages, you can translate messages into another language and attach a long text to a message.

- You can also use the `SYSERR` utility to modify the texts of existing Natural system messages, although this is not recommended as modifications will be lost with new Natural releases.

For further information, see *SYSERR Utility* in the *Tools and Utilities* documentation.

43 SYSEXT

SYSEXT

This command is used to invoke the SYSEXT utility.

This utility is used to display various Natural application programming interfaces contained in the library SYSEXT.

For further information, see *SYSEXT - Natural Application Programming Interfaces* in the *Tools and Utilities* documentation.

44   SYSEXV

SYSEXV

This command is used to invoke the SYSEXV utility.

The SYSEXV utility gives you access to examples of new features available in the current and in some earlier versions of Natural.

For further information, see *SYSEXV Utility* in the *Tools and Utilities* documentation.

45 SYSFILE

SYSFILE

This command is used to display work and print files information. You can obtain information about the following:

- reports,
- logical devices,
- defined physical devices,
- defined printer profiles, and
- defined workfiles.

See also *Work and Print Files* in *Using Natural Studio*.

For further information on work and print files, see

- *Printer Profiles* in the *Configuration Utility* documentation, and
- *Device/Report Assignments* in the *Configuration Utility* documentation,
- *Work Files* in the *Operations* documentation,

SYSFILE in a Remote Mainframe Environment

You can obtain the following work and print files information:

- type of assignment,
- record format,
- logical record length,
- block size,
- status,
- dynamic parameter specification.

46 SYSINST

```
SYSINST
```

This command is used to invoke the Natural Installer. This utility helps you to install Natural add-on products, for example, Natural Security.

For further information, see *Installer* in the *Tools and Utilities* documentation.

47 **SYSMAIN**

```
SYSMAIN
```

This command is used to invoke the `SYSMAIN` utility. You use this utility to perform operations such as copy, move and delete on Natural objects. The `SYSMAIN` utility is also used to transfer objects within the Natural system from one environment to another using the import function.

For further information, see *SYSMAIN Utility* in the *Tools and Utilities* documentation.

> **Note:** This command is not executable in batch mode.

48 SYSMN

`SYSMN`

This command is used to invoke Mainframe Navigation.

For further information, refer to the Mainframe Navigation documentation.

49 SYSNCP

```
SYSNCP
```

This command is used to invoke the SYSNCP utility.

For further information, see *SYSNCP Utility* in the *Tools and Utilities* documentation.

50   SYSOBJH

```
SYSOBJH
```

This command is used to invoke the Object Handler. You use the Object Handler to process Natural and non-Natural objects for distribution in Natural environments.

For further information, see *Object Handler* in the *Tools and Utilities* documentation.

51 SYSPROD

```
SYSPROD
```

This command is used to ascertain which products are installed at your Natural site. You are given information on your current Natural version, Natural selectable units and products running with or under Natural.

When you enter the command, a dialog displays information such as the following for each product installed:

- the product name,
- the product version (see also *Version* in the *Glossary*),
- the installation date,
- the product identification code (ID).

See also *Product Information* in *Using Natural Studio*.

52 SYSPROF

`SYSPROF`

This command is used to display the current definitions of the Natural system files.

For each system file, the following information is displayed (on the **System Files** page):

- the file name
- the database ID
- the file number
- the database type

In addition, the following information can be displayed for each defined combination of database ID and file number:

- the path in the file system (on the **Files in File System** page)
- the logical file number, if assigned (on the **All Files** page)

See also *System Files* in *Using Natural Studio*.

53 SYSRPC

`SYSRPC`

This command is used to invoke the `SYSRPC` utility.

The `SYSRPC` utility provides functions for maintaining remote procedure calls.

For further information, see *SYSRPC Utility* in the *Tools and Utilities* documentation.

For information on how to apply the `SYSRPC` utility functions to establish a framework for communication between server and client systems, refer to the *Natural Remote Procedure Call (RPC)* documentation.

54 **SYSWIZDB**

```
SYSWIZDB
```

This command is used to invoke the Data Browser, a development tool wizard within Natural Studio. It enables you to display and print or store file structures.

For further information, see *Data Browser* in the *Tools and Utilities* documentation.

55   SYSWIZDW

```
SYSWIZDW
```

This command is used to invoke the Dialog Wizard, a tool for creating dialogs for specific purposes. The defined dialogs can have several layouts that adapt to desired requirements.

For further information, see *Dialog Wizard* in the *Dialog Editor* section of the *Editors* documentation.

56 TECH

```
TECH
```

This command is used to display the following technical and other information about your Natural session:

- user ID
- library ID
- Natural version (see also *Version* in the *Glossary*)
- startup transaction
- Natural Security indicator
- operating system name and version
- machine class
- hardware
- TP monitor (Mainframes and Windows (`*TPSYS`) in remote configuration only)
- device type
- terminal ID (Mainframes and Windows in remote configuration only)
- code page
- locale
- last command issued
- information on the last error that occurred
- names, database IDs and file numbers of all currently active steplibs
- names, types and levels of the currently active programming object and all objects on higher levels, as well as the line numbers of the statements invoking the subordinate programming objects (Mainframes, UNIX and OpenVMS only).

See also *Technical Information* in *Using Natural Studio*.

📄 **Notes:**

1. For character-user-interface applications only: To display this information from any point in an application, you can use the terminal command `%<TECH`. In addition, the following information is still available: Names, types and levels of the currently active programming object and all objects on higher levels.

2. This command is also available in a remote session. All information can be read in batch mode.

57 **UNCATALOG**

```
UNCATALOG [object-name ...]
```

This command is used to delete one or more object modules.

To prevent inconsistencies, you are recommended to use the menu command **Delete** and to delete both source code and object module of an object. See *Deleting Objects* in *Using Natural Studio*.

You can only delete objects which are stored in the library to which you are currently logged on. The contents of the source work area is not affected by the UNCATALOG command.

`UNCAT` `UNCAT *`	If you enter the UNCATALOG command without an *object-name* or with an asterisk (*), a list of all cataloged objects in the current library will be displayed; on the list, you can then mark the object(s) to be deleted.
`UNCAT object name`	As *object-name*, you specify the name of the object to be deleted. If more than one object is to be deleted, the *object-names* must be separated by one or more blanks (or the currently defined delimiter character). If you wish to delete all objects whose names begin with a specific string of characters, use asterisk notation (*) for the *object-name*. A list containing all objects selected will be displayed. On the list, you can then mark the object(s) to be deleted.

> **Note:** If an FDIC system file is specified in the parameter file which is not valid, Natural will display an appropriate error message when the UNCATALOG command is issued.

58 UNLOCK

This command is used for unlocking

- Natural source objects in a remote development environment, and

- documentation objects in the local development environment, provided that Predict Version 4.4 or above is installed on Windows.

- documentation objects in a remote development environment, provided that the Object Description plug-in is installed (see separate Object Description documentation).

It is used to view source objects or documentation objects that are locked and to unlock them if need be. This command is recommended for use by the Natural administrator only. However, the administrator can enable the use of this command for each user profile in Natural Security.

This chapter covers the following topics:

For further information, see *Unlocking Objects Manually* in the *Remote Development Using SPoD* documentation.

See also *Object Naming Conventions* in the *Using Natural Studio* documentation.

Unlocking Natural Objects

If the system command `UNLOCK` is used without parameters, a dialog appears where you can enter the parameters.

```
UNLOCK
```

The following shows the direct command syntax for unlocking Natural objects.

```
UNLOCK [NATURAL] [OBJECT] object-name
        [TYPE object-type]
        [LIBRARY library-name]
        [DBID dbid] [FNR fnr]
        [PASSWORD password] [CIPHER cipher]
        [APPLICATION application-name]
        [USER locked-by]
        [DATE locked-on [locked-on2]]
```

Unlocking Documentation Objects

The following shows the direct command syntax for unlocking documentation objects.

```
UNLOCK DOCUMENT [OBJECT] object-name
       [TYPE object-type]
       [USER locked-by]
       [DATE locked-on [locked-on2]]
```

Parameter Descriptions

The object name must be defined in each case. If any of the other parameters is not specified, the corresponding default value will be used.

Parameter	Format/Length	Default Value	Description
object-name	A33	*	The name of the object to be unlocked. Asterisk notation (*) or ">" can be used.
object-type	A1	*	**Natural object types:** In place of object-type, you may specify one of the object type codes shown below or an asterisk (*).<table><tr><td>P</td><td>Program</td></tr><tr><td>4</td><td>Class</td></tr><tr><td>N</td><td>Subprogram</td></tr><tr><td>S</td><td>Subroutine</td></tr><tr><td>7</td><td>Function</td></tr><tr><td>8</td><td>Adapter</td></tr><tr><td>C</td><td>Copycode</td></tr><tr><td>H</td><td>Helproutine</td></tr><tr><td>T</td><td>Text</td></tr><tr><td>3</td><td>Dialog</td></tr><tr><td>M</td><td>Map</td></tr><tr><td>L</td><td>Local Data Area</td></tr><tr><td>G</td><td>Global Data Area</td></tr><tr><td>A</td><td>Parameter Data Area</td></tr><tr><td>V</td><td>DDM (View)</td></tr></table>

Parameter	Format/Length	Default Value	Description
			X	Application
	A2	*	**Documentation object types:** User-defined short descriptions for documentation object types or an asterisk (*).
library-name	A8	*	Name of the library where the locked object is in. Asterisk notation (*) can be used.
dbid	A5	current database ID	Database ID of the defined library. Specify asterisk (*) or in format N5. On mainframe servers with parameter SLOCK=PRE, the following applies: When asterisk notation (*) is used, only the current FNAT, FUSER and FDIC system files are scanned.
fnr	A5	current file number	File number of the defined library. Specify asterisk (*) or in format N5. On mainframe servers with parameter SLOCK=PRE, the following applies: When asterisk notation (*) is used, only the current FNAT, FUSER and FDIC system files are scanned.
password	A8	blank	If used, the password for the specified system file (*dbid* and *fnr*). Needs not to be specified, when the *dbid* and *fnr* of the current FNAT or FUSER is used. This parameter is available only in a mainframe remote development environment and when profile parameter SLOCK=PRE has been set in the mainframe environment.
cipher	A8	blank	If used, the cipher key for the specified system file (*dbid* and *fnr*). Needs not to be specified, when the *dbid* and *fnr* of the current FNAT or FUSER is used. This parameter is available only in a mainframe remote development environment and when profile parameter SLOCK=PRE has been set in the mainframe environment.
application-name	A32	blank	If used, the name of the application to which the locked object belongs. If you specify a blank, all locked objects, irrespective of whether they are linked to an application or not, are listed in a results window where they can be unlocked manually.

Parameter	Format/Length	Default Value	Description
locked-by	A8	current user ID	ID of the user who caused the object to be locked. Asterisk notation (*) can be used. If Natural Security is used, it can be changed only if the security unlock flag is set to "F" (forced unlock) in the Natural Security user profile.
locked-on *locked-on2*	A10 A8	blank	The two date parameters are available to provide for the different date formats: 2005-09-28 (date format according to the DTFORM profile parameter) 2005-09-28 11:27:20 Today Today + *nnnn* Today - *nnnn* Yesterday

> **Note:** Locking can also be enabled locally on a mainframe server based on Natural for Mainframes Version 4.2 or above. In this case, the following limitations apply: The *application-name* cannot be used as a selection criterion. For *dbid* and *fnr*, the current FNAT and FUSER system files are searched if asterisk notation (*) is used.

Parameter Processing and Display of Objects Found

If the parameter(s) specified is (are) valid and a complete object name is specified and if the corresponding object is found and it was locked by the current user, this object is unlocked immediately and a corresponding message is displayed. This applies under the condition that the object name is specified directly without using asterisk notation (*) and the current user tries to unlock his own locked records.

If any of the parameters specified is invalid or if no objects are found, the unlock dialog with an error message box will appear.

In the following cases, the locked objects found are listed in a results window where they can be unlocked manually:

- if you used asterisk notation (*) or ">" (where applicable),
- if you did not specify a specific object name,
- if you did not specify an application name.

> **Note:** On mainframe servers with parameter `SLOCK=PRE`, the following applies: When asterisk notation (*) is used for object type and library, the locked DDMs have also to be listed by scanning the current `FDIC` system file.

If the object type of a documentation object is not unique, look into the hidden column next to the object type for the internal object types.

For further information on the results window, see *Unlocking Objects Manually* in the *Remote Development Using SPoD* documentation.

59 UNMAP

The `UNMAP` command enables you to perform the following functions, using the Natural command line:

To map a Natural Development Server or a Natural application, you can use the system command `MAP` or the dialog described in *Mapping/Unmapping an Application* in the *Remote Development Using SPoD* documentation.

Unmapping the Currently Active Environment/Application

The following command syntax applies if you want to unmap the currently active Natural Development Server environment or Natural application:

```
UNMAP
```

Unmapping a Natural Development Server Environment

The following command syntax applies if you want to unmap a Natural Development Server environment:

```
UNMAP ENVIRONMENT=environment-name
```

Where `environment-name` is the alias name of the connection. If the environment name contains blanks, it must be enclosed in single quotes ('...').

Unmapping a Natural Application

The following command syntax applies if you want to unmap a Natural application:

```
UNMAP APPLICATION=application-name
```

Where `application-name` is the name of the application to be unmapped.

60 **UNREGISTER**

```
UNREGISTER { class-module-name }  [ { library-name } [server-id] ]
           { *                }    [ { *            }             ]
```

Related command: REGISTER.

This command is used to unregister Natural classes.

For further information, see *The UNREGISTER Command* in the *Administrating NaturalX Applications* part of the *Operations* documentation.

> **Note:** Under Natural Security, this command can only be called for a single library. This means the library name has either to be omitted or a specific library has to be used. It is not possible to use an asterisk (*).

61 UPDATE

```
UPDATE  { ON  }
        { OFF }
```

This command is used to prevent (or allow) database updating being carried out by a program.

UPDATE ON	This allows updating. This command will be ignored if the Natural administrator has made updating impossible during Natural installation.
UPDATE OFF	This prevents updating which would normally be performed as a result of an UPDATE, STORE, or DELETE statement. Programs containing these statements will execute normally but no modification of the database will occur. When an update operation is encountered, a message will be displayed instead of a database update being performed.

When the system command CHECK is used with UPDATE OFF, an error message is displayed. The UPDATE command has no effect on other Natural system commands.

62   XREF

```
        ┌ ON     ┐
        │ OFF    │
XREF    │ FORCE  │
        │ DOC    │
        └        ┘
```
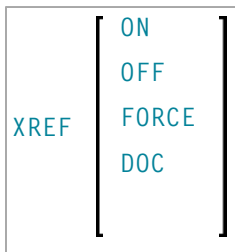
This command is only available if Predict has been installed. It controls the usage of the Predict function "active cross-references".

The active cross-reference facility automatically creates documentation in the Data Dictionary about the objects with a program/data area reference. These objects include programs, subprograms, subroutines, helproutines, maps, data areas, database views, database fields, user-defined variables, processing rules, error numbers, work files, printers, classes and retained ISN sets.

The active cross-reference is created when a program/data area is cataloged.

To look at cross-reference data, you use the XREF option of the system command LIST.

For further information on active cross-references, see the Predict documentation.

The following command options are available:

XREF	If you enter the XREF command without parameters, a menu/dialog is displayed where you specify the desired option.
XREF ON	This command activates the active cross-reference function. Cross-reference data will be stored in the respective Predict entries each time a Natural program/data area is cataloged.
XREF OFF	This command deactivates the active cross-reference facility. No cross-reference data will be stored. Existing cross-reference data for the object being cataloged will be deleted.

XREF FORCE	The object can only be cataloged if a Predict entry exists for it. When the object is cataloged, its cross-reference data will be stored in Predict. If no Predict entry exists, the object cannot be cataloged.
XREF DOC	The object can only be cataloged if a Predict entry exists for it. However, when the object is cataloged, no cross-reference data will be stored in Predict, and existing cross-reference data for the object will be deleted. If no Predict entry exists, the object cannot be cataloged.

Natural Security Considerations

If Natural Security is installed, the setting for XREF may be set for each library in the library security profile. Depending on the security profile, some options of the XREF command may not be available to you.