Natural

System Commands

Version 9.3.4

October 2026

ADABAS & NATURAL

This document applies to Natural Version 9.3.4 and all subsequent releases.

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

Copyright © 1992-2026 Software GmbH, Darmstadt, Germany and/or its subsidiaries and/or its affiliates and/or their licensors.

The name Software AG and all Software GmbH product names are either trademarks or registered trademarks of Software GmbH and/or its subsidiaries and/or its affiliates and/or their licensors. Other company and product names mentioned herein may be trademarks of their respective owners.

Detailed information on trademarks and patents owned by Software GmbH and/or its subsidiaries is located at https://softwareag.com/licenses.

Use of this software is subject to adherence to Software GmbH's licensing conditions and terms. These terms are part of the product documentation, located at https://softwareag.com/licenses and/or in the root installation directory of the licensed product(s).

This software may include portions of third-party products. For third-party copyright notices, license terms, additional rights or restrictions, please refer to "License Texts, Copyright Notices and Disclaimers of Third-Party Products". For certain specific third-party license restrictions, please refer to section E of the Legal Notices available under "License Terms and Conditions for Use of Software GmbH Products / Copyright and Trademark Notices of Software GmbH Products". These documents are part of the product documentation, located at https://softwareag.com/licenses and/or in the root installation directory of the licensed product(s).

Use, reproduction, transfer, publication or disclosure is prohibited except as specifically provided for in your License Agreement with Software GmbH.

Document ID: NATWIN-NNATSYSCOM-934-20261013

Table of Contents

Preface vii
1 About this Documentation 1
Document Conventions 2
Online Information and Support 2
Data Protection 3
2 Issuing System Commands 5
Command Input 6
Command Line 6
NEXT Prompt 6
MORE Prompt 7
3 System Command Syntax 9
Syntax Elements 10
Example of Command Syntax 11
4 System Commands Grouped by Category 13
Navigating in Natural 14
Natural Development Environment 14
Managing Applications with Natural Objects 15
Monitoring, Debugging and Tracing 16
Natural with Other Products 17
5 CATALL 19
CATALL in Interactive Mode 20
CATALL in Batch Mode 21
6 CATALOG 23
7 CHECK 25
8 CLEAR 27
9 COMPOPT 29
COMPOPT Syntax Explanation 30
Compiler Options 30
Specifying Compiler Parameters 30
COMPOPT in a Remote z/OS Environment 31
Specifying Compiler Keyword Parameters (Remote z/OS Environment) 32
General Compilation Options (Remote z/OS Environment) 32
10 DEBUG 51
11 EDIT 53
EDIT Syntax 1 54
EDIT Syntax 2 56
EDIT Syntax 3 56
12 EXECUTE 57
EXECUTE Syntax Explanation 58
Examples of EXECUTE Command 59
13 FIN 61
14 GLOBALS 63
GLOBALS Syntax Explanation 64

GLOBALS Parameters ... 64
Interaction with SET GLOBALS and Other Statements ... 66
15 HELP ... 67
16 INPL ... 69
17 LAST ... 71
18 LASTMSG ... 73
19 LIST ... 75
LIST Syntax Overview ... 76
Displaying an Individual Source ... 77
Displaying a List of Objects ... 78
Displaying Numbers and Sizes of Objects ... 79
Displaying Directory Information ... 79
Displaying Views ... 80
Displaying File Information of Resource Objects ... 80
Displaying File Information of Error Message Containers ... 80
20 LIST XREF ... 81
21 LOGOFF ... 83
22 LOGON ... 85
23 MAIL ... 87
24 MAP ... 89
Establish a Connection to a Natural Development Server Environment ... 90
Establish a Connection to a Natural Application ... 91
25 PROFILE ... 93
26 PROFILER ... 95
27 PURGE ... 97
28 READ ... 99
29 REGISTER ... 101
30 RENAME ... 103
31 RENUMBER ... 105
32 RETURN ... 107
33 RPCERR ... 109
34 RUN ... 111
35 SAVE ... 113
36 SCAN ... 115
37 SCRATCH ... 117
38 SETUP ... 119
SETUP Syntax Explanation ... 120
SETUP/RETURN Example ... 121
39 STOW ... 123
40 STRUCT ... 125
Indentation of Source Code Lines ... 126
41 SYSAPI ... 129
42 SYSCP ... 131
43 SYSERR ... 133
44 SYSEXT ... 135

45 SYSEXV 137
46 SYSFILE 139
47 SYSLVERS 141
48 SYSMAIN 143
49 SYSMN 145
50 SYSNCP 147
51 SYSOBJH 149
52 SYSPROD 151
About SYSPROD 152
53 SYSPROF 153
54 SYSRPC 155
55 SYSWIZDB 157
56 SYSWIZDW 159
57 TECH 161
58 UNCATALOG 163
59 UNLOCK 165
Unlocking Natural Objects 166
Unlocking Documentation Objects 167
UNLOCK Parameter Descriptions 167
Parameter Processing and Display of Objects Found 169
60 UNMAP 171
Unmapping the Currently Active Environment/Application 172
Unmapping a Natural Development Server Environment 172
Unmapping a Natural Application 172
61 UNREGISTER 173
62 UPDATE 175
63 WHICH 177
Direct Command Syntax 178
WHICH Parameter Description 178
Default Output of WHICH Command 178
Full List of WHICH Command 179
Batch Processing 180
WHICH Command under Natural Security 180
64 XREF 181

Preface

This documentation describes the Natural system commands. It 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 Category	Provides an overview of the Natural system commands grouped by category.
System Commands in Alphabetical Order	Descriptions of the system commands in alphabetical order.

1 About this Documentation

- Document Conventions 2
- Online Information and Support 2
- Data Protection 3

Document Conventions

Convention	Description
Bold	Identifies elements on a screen.
`Monospace font`	Identifies service names and locations in the format *`folder.subfolder.service`*, APIs, Java classes, methods, properties.
`Italic`	Identifies: Variables for which you must supply values specific to your own situation or environment. New terms the first time they occur in the text. References to other documentation sources.
`Monospace font`	Identifies: Text you must type in. Messages displayed by the system. Program code.
{ }	Indicates a set of choices from which you must choose one. Type only the information inside the curly braces. Do not type the { } symbols.
|	Separates two mutually exclusive choices in a syntax line. Type one of these choices. Do not type the | symbol.
[ ]	Indicates one or more options. Type only the information inside the square brackets. Do not type the [ ] symbols.
...	Indicates that you can type multiple options of the same type. Type only the information. Do not type the ellipsis (...).

Online Information and Support

Product Documentation

You can find the product documentation on our documentation website at **https://documentation.softwareag.com**.

Product Training

You can find helpful product training material on our Learning Portal at **https://learn.softwareag.com**.

Tech Community

You can collaborate with Software GmbH experts on our Tech Community website at **https://tech-community.softwareag.com**. From here you can, for example:

- Browse through our vast knowledge base.
- Ask questions and find answers in our discussion forums.
- Get the latest Software GmbH news and announcements.
- Explore our communities.
- Go to our public GitHub and Docker repositories at **https://github.com/softwareag** and **https://hub.docker.com/publishers/softwareag** and discover additional Software GmbH resources.

Product Support

Support for Software GmbH products is provided to licensed customers via our Empower Portal at **https://empower.softwareag.com**. Many services on this portal require that you have an account. If you do not yet have one, you can request it at **https://empower.softwareag.com/register**. Once you have an account, you can, for example:

- Download products, updates and fixes.
- Search the Knowledge Center for technical information and tips.
- Subscribe to early warnings and critical alerts.
- Open and update support incidents.
- Add product feature requests.

Data Protection

Software AG products provide functionality with respect to processing of personal data according to the EU General Data Protection Regulation (GDPR). Where applicable, appropriate steps are documented in the respective administration documentation.

2 Issuing System Commands

- Command Input 6
- Command Line 6
- NEXT Prompt 6
- MORE Prompt 7

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

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*).

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.

3 System Command Syntax

- Syntax Elements 10
- Example of Command Syntax 11

Syntax Elements

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

Element	Explanation
ABCDEF	Upper-case non-italic 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 the 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 the 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 the 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.

4 System Commands Grouped by Category

- Navigating in Natural 14
- Natural Development Environment 14
- Managing Applications with Natural Objects 15
- Monitoring, Debugging and Tracing 16
- Natural with Other Products 17

This chapter is a summary of Natural system commands grouped by category.

Navigating in Natural

Command	Brief Description
FIN	Terminates a Natural session.
LAST	Displays the system commands that were last executed, and allows you to execute them again.
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.

Natural Development Environment

Command	Brief Description
COMPOPT	Sets compilation options that affect the way in which Natural objects are compiled.
GLOBALS	Changes the settings of Natural session parameters.
HELP	Invokes the Natural help system.
LAST	Displays the system commands that were last executed, and allows you to execute them again.
LIST XREF	Displays all active cross-reference data for the current library. (Only available if Predict is installed.)
MAP	Establishes a connection to a remote development server.
SYSAPI	Only applies to products that provide application programming interfaces (APIs). Invokes the SYSAPI utility to locate APIs if provided by the products installed at your site.
SYSCP	Invokes the SYSCP utility to view code page information.
SYSEXT	Invokes the SYSEXT utility with Natural application programming interfaces.
SYSEXV	Invokes the SYSEXV utility with examples of the new features of the current Natural versions and debugging hints.
SYSFILE	Displays work and print files information.
SYSPROD	Displays a list of the products installed at your site, and information on these products.
SYSPROF	Displays the current definitions of the Natural system files.

Command	Brief Description
UNMAP	Disconnects the currently active remote environment.
UPDATE	Prevents database updating being carried out by a program.
XREF	Controls the use of the Predict function "active cross-references". This function automatically creates documentation in Predict about the objects which a program/data area references. (Only available if Predict is installed.)

Managing Applications with Natural Objects

Command	Brief Description
CATALL	Catalogs (compiles) *all* objects or selected objects in the current library.
CATALOG	Catalogs (compiles) the source code currently in the editor work area, and if the syntax has been found to be correct, stores the resulting cataloged object in the Natural system file.
CHECK	Checks that the source code currently in the editor work area does not contain any syntax errors. 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	Opens an editor to create or modify source code.
EXECUTE	Executes a cataloged (compiled) program that has been stored as a cataloged object in the Natural system file.
INPL	Invokes the INPL utility. It is *only* used for the loading of installation data sets into the system files.
LIST	Lists one or more objects contained in the current library or the contents of the editor work area.
PURGE	Deletes one or more source objects from the current library.
READ	Transfers a source object from the Natural system file to the editor work area.
RENAME	Changes the name of an object or the name and the type of an object.
RENUMBER	Renumbers the source code currently in the editor work area.
RUN	Compiles and executes the source program currently in the work area of the editor.
SAVE	Stores the source code currently in the editor work area as a source object in the Natural system file.
SCAN	Searches for a character string within a source with the option to replace the string.
SCRATCH	Deletes a source object and/or the corresponding cataloged object from the current library.
STOW	Catalogs (compiles) and stores source code as both a source object and a cataloged object in the current Natural system file.

Command	Brief Description
STRUCT	Performs structural indentation of a program source and helps detecting structural inconsistencies.
SYSERR	Creates and maintains the messages you wish your Natural applications to display to the users.
SYSLVERS	List objects which have been cataloged within a selected Natural version range.
SYSMAIN	Transfers Natural objects within the Natural system from one library to another.
SYSMN	Invokes **Mainframe Navigation** to access and manipulate objects stored on z/OS from a Windows application. For detailed information, refer to the *Mainframe Navigation* documentation.
SYSNCP	Creates and maintains the command processors to be used in your Natural applications.
SYSOBJH	Processes Natural and non-Natural objects for distribution in Natural environments.
SYSRPC	Invokes the SYSRPC utility to create and maintain remote procedure calls, that is, provides the settings necessary to execute a subprogram located on a remote server.
SYSWIZDB	Invokes the Natural Data Browser, a development tool wizard within Natural Studio. It enables you to display and print or store file structures.
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.
UNCATALOG	Deletes one or more cataloged objects in the current library.
UNLOCK	Displays locked Natural objects with the option to unlock them.
WHICH	Lists objects in the libraries used at run time in the Library Search Order.

Monitoring, Debugging and Tracing

Command	Brief Description
DEBUG	This command is used to invoke the Natural debugger.
HELP	Invokes the Natural help system.
LASTMSG	Displays additional information on the error situation which occurred last.
RPCERR	Displays the last Natural error number and message if related to Natural RPC (Remote Procedure Call), and the last EntireX Broker reason code and associated message.
TECH	Displays technical and other information on your Natural session.

Natural with Other Products

The following system commands are only available in connection with other products installed at your site:

Product	Command	Brief Description
Natural Security	MAIL	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.
	PROFILE	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.
NaturalX	REGISTER	Registers Natural classes. They are registered for the server ID under which Natural was started.
	UNREGISTER	Unregisters Natural classes.
Predict	LIST XREF	Displays all active cross-reference data for the current library.
	XREF	Controls the use of the Predict function “active cross-references”. This function automatically creates documentation in Predict about the objects which a program/data area references.
Various products	SYSAPI	Invokes the SYSAPI utility to locate application programming interfaces (APIs) if provided by the products installed at your site.

5 CATALL

- CATALL in Interactive Mode 20
- CATALL in Batch Mode 21

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 processed in the order in which the object types are listed in the dialog box (see also the information for TYPES in the section *CATALL in Batch Mode*). 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 cataloged objects exist in the current library will be cataloged again; source objects for which no cataloged objects exist 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 not renumbered. If you wish automatic renumbering of lines, activate 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 z/OS 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 as source objects *and* as cataloged objects. When you are working in a remote z/OS development environment using SPoD, this option cannot be used and will be dimmed.

CATALL in Batch Mode

```
CATALL object-name [ RECAT ] [TYPES types] [ CATALOG ] [options ...]
                   [ ALL   ]               [ CHECK   ]
                                           [ SAVE    ]
                                           [ STOW    ]
```

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 (objects are processed in the order in which the object types are listed below): D DDMs G Global data areas

	L Local data areas A Parameter data areas C Copycodes T Texts 7 Functions N Subprograms S Subroutines H Helproutines M Maps 8 Adapters P Programs 4 Classes 3 Dialogs * All types (this is the default) The *types* have to be specified as *one* character string, for example, GLA for global, local and parameter 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).

6 CATALOG

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

Related commands: SAVE | STOW.

This command is used to catalog (compile) the source code currently in the work area of a Natural editor and (if the syntax has been found to be correct) store the resulting cataloged object in the current 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.

7 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*.

8 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*.

9 COMPOPT

- COMPOPT Syntax Explanation 30
- Compiler Options 30
- Specifying Compiler Parameters 30
- COMPOPT in a Remote z/OS Environment 31
- Specifying Compiler Keyword Parameters (Remote z/OS Environment) 32
- General Compilation Options (Remote z/OS Environment) 32

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

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

COMPOPT 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 | ECHECK | GFID | KCHECK | MASKCME | MAXPREC | 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 Object**

   In a Natural 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 object, but do not update settings set with the command `COMPOPT`.

COMPOPT in a Remote z/OS Environment

The topics provided below apply when using the `COMPOPT` command in a remote z/OS environment.

Specifying Compiler Keyword Parameters (Remote z/OS 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.
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 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 object, but do not update settings set with the command `COMPOPT`.

General Compilation Options (Remote z/OS Environment)

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

- 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

- DB2PKYU – Place Primary Key Fields into the Natural DML UPDATE Statement
- 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
- MASKCME - MASK Compatible with MOVE EDITED
- MAXPREC – Maximum Number of Digits after Decimal Point
- MEMOPT - Memory Optimization for Locally Declared Variables
- NMOVE22 - Assignment of Numeric Variables of Same Length and Precision
- 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
- V82COMP - Disallow Syntax Not Available on Natural for Mainframe 8.2

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 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.

DB2PKYU – Place Primary Key Fields into the Natural DML UPDATE Statement

Only applies if supported by the Natural for Db2 version installed at your site.

The DB2PKYU option can be used to update Db2 primary key fields with a Natural DML UPDATE statement. Db2 primary key fields are fields whose short names begin with the character O in the DDM.

Note: The setting of DB2PKYU at the end of the compilation is used for the complete Natural object. It cannot be changed for parts of a Natural object.

ON	Db2 primary key fields are updated. Db2 primary key fields which are updated within the Natural program are placed into the resulting Db2 positioned UPDATE statement of a Natural DML UPDATE statememt. The SQLCODE +535 Db2 returned for this positioned UPDATE is treated as 0 (zero) by the Natural for Db2 runtime system.
OFF	Db2 primary key fields are not updated. Db2 primary key fields which are updated within the Natural program are not placed into the resulting Db2 positioned UPDATE statement. 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 an 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, and 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 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 the structural information of a GDA (global data area). This GDA information is called a "GDA signature". A GDA signature is a hash value calculated from the GDA data definitions including the variable name, format, length, and initial values.

- Structural Information of the GDA
- How to Find out the Current Settings of the GDA Signatures
- Benefit of Using GDA Signatures

- Enabling GDA Signatures

Structural Information of the GDA

`ON`	When a GDA is cataloged, the GDA signature is stored in the cataloged GDA.
`OFF`	When a GDA is cataloged, the GDA is stored without a GDA signature. This is the default value.

When a Natural object that references a GDA is cataloged and the GDA contains a GDA signature, that GDA signature is copied into the cataloged object, regardless of the current GDASC setting.

At execution time, the GDA signature of the executed Natural object is compared to the GDA signature of the referenced GDA. If both signatures exist and are identical, the GDA is accepted, even if the timestamps are not identical.

The following table describes in detail what is compared at execution time and the expected result.

GDA Contains Signature	Natural Object Contains Signature	Compare	Result
yes	yes	Signatures	Error if not equal
yes	no	Timestamps	Error
no	yes	Timestamps	Error
no	no	Timestamps	Error if not equal

How to Find out the Current Settings of the GDA Signatures

The LIST DIRECTORY command shows the field `GDA signature` when a GDA or a Natural object contains a GDA signature.

For a GDA, the GDA signature will be checked for its existence. For a Natural object, the GDA signature will be checked against the signature of the used GDA.

See the following table with the settings of the field `GDA signature`:

GDA Contains Signature	Natural Object Contains Signature	GDA Signature in LIST DIRECTORY of a GDA	GDA Signature in LIST DIRECTORY of a Natural Object
yes	yes	Generated	Identical / Not identical
yes	no	Generated	Only in GDA
no	yes	-*1	Only in object
no	no	-*1	-*2

Notes:

1. `*1`: Show GDA timestamp when GDA was cataloged.
2. `*2`: Show GDA timestamps when the GDA and the Natural object were cataloged.

Benefit of Using GDA Signatures

The GDA signature of a GDA persists if it is merely recataloged but not modified, or if only the comments are modified within the GDA. In these cases, Natural objects referencing the GDA still retain the same GDA signature as the GDA and therefore do not need to be recataloged.

Enabling GDA Signatures

To enable GDA signatures, set `GDASC` to `ON`.

When you catalog a GDA for the first time with `GDASC` set to `ON`, you must also catalog all referencing Natural objects. GDA signatures will then be used at execution time.

Other libraries that do not (yet) have the GDA cataloged with `GDASC` set to `ON` will continue to use GDA timestamps for GDA validation checks. The Natural objects remain without GDA signature.

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=abbbbbb
```

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 an 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 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 z/OS platforms. It facilitates the transfer of programs written in mixed/lower-case characters from other platforms to a z/OS 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
```

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.

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.

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.

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, and 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. 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 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.
OFF	No parameter check is performed. This is the default setting.

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.

V82COMP - Disallow Syntax Not Available on Natural for Mainframe 8.2

The V82COMP option checks whether the syntax of the features provided since Natural for z/OS Version 9.2 is also supported by Natural for Mainframes Version 8.2. If any syntax incompatibilities between these two versions are detected, the compilation fails with an appropriate Natural error message and reason code.

The following values are possible:

ON	When a program is compiled, every attempt to use a syntax construction that is supported by Natural for z/OS 9.2, but not in version 8.2, is rejected with a NAT0597 syntax error and an appropriate **reason code** (see the following section).
OFF	No syntax check is performed. This is the default value.

Reason Codes for Syntax Errors

The following reason codes indicate which syntax parts are not supported by Natural for Mainframe 8.2:

Reason Code	Invalid Syntax on Natural for Mainframe 8.2
019	The following new system variable was used: `*TIMESTMPX`

020	The `COMPRESS` statement allows to use a field of type `LOGICAL`.
	
	Example:
	
	`DEFINE DATA LOCAL`
	`1 #L (L)  INIT <TRUE>`
	`1 #RESULT (A50)`
	`END-DEFINE`
	`COMPRESS 'The value of #L is' #L INTO #RESULT`

021	The `COMPRESS` statement allows to provide an edit mask for each field.
	
	Example:
	
	`DEFINE DATA LOCAL`
	`1 #A (A10)  INIT <'ABCDEF'>`
	`1 #P (P5.3)  INIT <-12.345>`
	`1 #L (L)  INIT <TRUE>`
	`1 #RESULT (A50)`
	`END-DEFINE`
	`COMPRESS #A (EM=X_X)  #P(EM=-ZZZZ9.9)  #L(EM=FALSE/TRUE) INTO #RESULT`

022	The work file number can be provided with a `CONST` variable in a `DEFINE WORK`, `WRITE WORK`, `READ WORK`, and `CLOSE WORK` statement.
	
	Example:
	
	`DEFINE DATA LOCAL`
	`1 #WNBR (I2)  CONST <12>`
	`END-DEFINE`
	`WRITE WORK FILE #WNBR 'ABC'`
	`CLOSE WORK FILE #WNBR`

023	The `READ .. BY ISN` statement was enhanced to allow backward reading in DESCENDING order. However, the usage of this feature requires to run against Adabas version 8.6 (or higher). Example:

Reason Code	Invalid Syntax on Natural for Mainframe 8.2

	`DEFINE DATA LOCAL`
	`1 VIEW1 VIEW OF EMPLOYEES`
	`2 NAME`
	`END-DEFINE`
	`READ (10) VIEW1 DESCENDING BY ISN = 1000  /* Reads 10 records starting ↩`
	`with ISN=1000`
	`DISPLAY *ISN NAME  /* (or the next lower) in ↩`
	`descending order`
	`END-READ`

10 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.

11 EDIT

- EDIT Syntax 1 54
- EDIT Syntax 2 56
- EDIT Syntax 3 56

This command is used to invoke a Natural editor for the purpose of creating and editing a Natural source.

Note: This command is not executable in batch mode. This command is not available if the Natural program, data area, and map editor are disabled in your environment. Use the source editor of NaturalONE to edit a source 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*.

EDIT 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.

The object types are described in *Objects for Natural Application Management* 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.

EDIT 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`.

EDIT 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
```

12 EXECUTE

- EXECUTE Syntax Explanation 58
- Examples of EXECUTE Command 59

```
{ 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*.

EXECUTE 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
```

13 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 data set.

14 GLOBALS

- GLOBALS Syntax Explanation 64
- GLOBALS Parameters 64
- Interaction with SET GLOBALS and Other Statements 66

```
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.

GLOBALS 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.

GLOBALS 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
RQTOUT	REQUEST DOCUMENT Timeout
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` 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.

15 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 NAT*nnnn*.
`HELP USER`*nnnn*	Displays the long text of the library-specific error message number *nnnn* in the current library.

16 INPL

INPL [R]

This command is used to invoke the Natural INPL utility. This utility is *only* used for the loading of installation data sets 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.

17 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.

18 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, and the date and time behind the error message itself.

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.

Function Keys

The output screen of the LASTMSG command provides the following function keys:

PF Key	Name	Function
PF1	Help	Invoke help. Provides information about the use of PF2 and PF5.
PF2	Clear	Clears all the error messages from the error message collector.
PF5	Info	Provides detailed information about an error, when the cursor is placed on the error message and you press PF5. The same thing happens when the cursor is placed on an error message and you press ENTER.

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

Natural RPC (Remote Procedure Call):

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

19 LIST

- LIST Syntax Overview 76
- Displaying an Individual Source 77
- Displaying a List of Objects 78
- Displaying Numbers and Sizes of Objects 79
- Displaying Directory Information 79
- Displaying Views 80
- Displaying File Information of Resource Objects 80
- Displaying File Information of Error Message Containers 80

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 description of the command `LIST XREF`.

LIST Syntax Overview

```
LIST [ [object-type] object-name
       COUNT [ * ]
             [ object-name-range ]
       DIRECTORY [object-name]
       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*	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.
LIST *object-type object-name*	

Displaying a List of Objects

`LIST 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 (?).
`LIST object-type object-name`	

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`, `NATvrAL` (where *vr* represents the relevant product version):

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

Displaying Numbers and Sizes of Objects

`LIST COUNT`	Displays a summary report that contains the numbers and sizes (in bytes or KB if greater than 1 MB) of all objects stored in the current library. The numbers and sizes listed refer to all objects that have been saved as source (**Saved**) objects only or as cataloged (**Cataloged**) objects only, and all objects for which both saved and cataloged objects (**Stowed**) exist.	
`LIST COUNT *`	Displays a summary report of all objects where the numbers and sizes of saved/cataloged objects are listed per object type(s) found.	
`LIST COUNT` *`object-name-range`*	Displays a summary report (same as above) for the specified range of objects. Valid range notations are:	
	`value>`	All objects with names greater than or equal to *`value`*
	`value<`	All objects with names less than or equal to *`value`*
	`value*`	All objects with names that start with *`value`*

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 (*).

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

- Establish a Connection to a Natural Development Server Environment 90
- Establish a Connection to a Natural Application 91

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 z/OS or Linux 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 PROFILER

```
PROFILER
```

This command is used to invoke the Profiler utility. The Profiler utility monitors the internal process flow of a Natural application and analyzes the performance of the application.

The Profiler utility can be executed in batch mode only.

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

27 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*.

28 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.

29 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.

30 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 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*.

31 RENUMBER

RENUMBER [(n)]

This command is used to renumber the lines in the source code currently in the work area of a Natural editor.

Note: If you want to renumber alphanumeric or Unicode constants, make sure that the RNCONST profile parameter is set to ON.

RENUMBER	If you enter the command without parameter, depending on the total number of source lines in the work area, the following default increment values are used for renumbering:	
	0001 to 0999 lines:	increments of 10
	1000 to 1999 lines:	increments of 5
	2000 to 4999 lines:	increments of 2
	5000 lines and more:	increments of 1
	Starting with line 9999, the line number assigned is always 9999. Be aware that you cannot compile sources that contain more than 9999 lines.	
RENUMBER (*n*)	*n* can be used to specify a value between 1 and 9999 as the increment for renumbering. If the given increment value would cause the renumbering to exceed the 9999 line number limit, the default increment value is used instead.	

See also *Renumbering of Source-Code Line Number References* in the *Programming Guide*.

32 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.

33 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 EntireX Broker reason code and associated message. Additionally, the node and server name from the last EntireX 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 RPC (Remote Procedure Call)* documentation.

34 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).

35 SAVE

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

Related commands: STOW | CATALOG.

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

36 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.

37 SCRATCH

```
SCRATCH [ { *              } ]
          { object-name... }
```

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`	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 *`	
`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.

38 SETUP

- SETUP Syntax Explanation 120
- SETUP/RETURN Example 121

```
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:

SETUP 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.

39 STOW

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

Related commands: SAVE | CATALOG.

This command is used to catalog (compile) and store a Natural object (in both source and object form) in the current 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.

40 STRUCT

- Indentation of Source Code Lines 126

STRUCT [(n)]

This command is used to perform structural indentation of the source code of the Natural 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
```

41 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.

42 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*.

43 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.

44 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.

45 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.

46 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 z/OS 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.

47 SYSLVERS

This command is used to invoke the `SYSLVERS` utility. The `SYSLVERS` utility lists objects which have been cataloged within a selected Natural version range.

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

48 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.

49 SYSMN

SYSMN

This command is used to invoke **Mainframe Navigation**.

For further information, refer to the *Mainframe Navigation* documentation.

50 SYSNCP

SYSNCP

This command is used to invoke the SYSNCP utility.

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

51 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.

52 SYSPROD

- About SYSPROD 152

SYSPROD

About SYSPROD

This command is used to ascertain which products are installed at your Natural site. You are given information about your current Natural version 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*.

53 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*.

54 SYSRPC

```
        [ CSMASS                                  ]
        [ PING        {                           }  ]
SYSRPC  [ SGMASS      { individual-syntax         }  ]
        [ SM REPLACE  {                           }  ]
        [ SRVLIST                                 ]
```

This system command invokes the `SYSRPC` utility which is used to maintain remote procedure calls.

You can specify direct commands with the `SYSRPC` system command in order to perform RPC-specific tasks. These tasks and the *`individual-syntax`* that applies to the direct commands are described in detail in *SYSRPC Utility* in the *Tools and Utilities* documentation:

Direct Command	Purpose and Related Topics
`CSMASS`	Calculates buffer sizes required for RPC calls. See *Calculating Size Requirements* and *Using the SYSRPC CSMASS Command*.
`PING`	Pings a single or all defined servers. See *Pinging an RPC Server* and *Using the SYSRPC PING Direct Command*.
`SGMASS`	Generates multiple interface objects. See *Generating Multiple Interface Objects* and *Using the SYSRPC SGMASS Command*.
`SM REPLACE`	Replaces single or multiple items in a service directory See *Replacing Items in the Service Directory*.
`SRVLIST`	Provides information on Natural RPC servers registered on EntireX Broker: see *Listing Servers Registered on EntireX Broker*.

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 RPC (Remote Procedure Call)* documentation.

55 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.

56 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.

57 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
- Natural editors: Indicates whether the Natural program, data area, and map editors are disabled or enabled (see `*EDITOR` system variable)
- operating system name and version
- machine class
- hardware
- IBM architecture level supported on the current IBM processor (z/OS)

  `0` (zero) denotes that architecture levels are not supported.
- TP monitor (z/OS and Windows (`*TPSYS`) in remote configuration only)
- device type
- terminal ID (z/OS 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 Natural object and all objects on higher levels, as well as the line numbers of the statements invoking the subordinate objects (z/OS and Linux 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 Natural 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.

58 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`	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 *`	
`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.

59 UNLOCK

- Unlocking Natural Objects 166
- Unlocking Documentation Objects 167
- UNLOCK Parameter Descriptions 167
- Parameter Processing and Display of Objects Found 169

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 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 required. 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]]
```

UNLOCK 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 (*). ■ P - Program ■ 4 - Class ■ N- Subprogram ■ S - Subroutine ■ 7 - Function ■ 8 - Adapter ■ C - Copycode ■ H - Helproutine ■ T - Text ■ 3 - Dialog ■ M - Map ■ L - Local Data Area ■ G - Global Data Area ■ A - Parameter Data Area

Parameter	Format/Length	Default Value	Description
			■ V - DDM (View) ■ X - Application
	A2	*	**Documentation object types:** User-defined short descriptions for documentation object types or an asterisk (*).
library-name	A8	*	The name of the library where the locked object is in. Asterisk notation (*) can be used.
dbid	A5	current database ID	The database ID of the defined library. Specify asterisk (*) or in format N5. On z/OS 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	The file number of the defined library. Specify asterisk (*) or in format N5. On z/OS 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 z/OS remote development environment and when profile parameter SLOCK=PRE has been set in the z/OS 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 z/OS remote development environment and when profile parameter SLOCK=PRE has been set in the z/OS 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	The 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	A10	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
locked-on2	A8		

Note: Locking can also be enabled locally on a Natural for z/OS server. 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 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 z/OS 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.

60 UNMAP

- Unmapping the Currently Active Environment/Application 172
- Unmapping a Natural Development Server Environment 172
- Unmapping a Natural Application 172

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.

61 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 (*).

62 UPDATE

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

This command is used to control database updates performed by a Natural program.

`UPDATE ON`	Database updates are allowed.
`UPDATE OFF`	Database updates are not allowed. A database update is not performed when a program with an `UPDATE`, `STORE`, or `DELETE` statement executes. Instead, a NAT1010 warning message is issued during the next screen I/O operation. In addition, a database loop that contains an `UPDATE` or `DELETE` statement does not place the records in hold status (no `read with hold`).

63 WHICH

- Direct Command Syntax 178
- WHICH Parameter Description 178
- Default Output of WHICH Command 178
- Full List of WHICH Command 179
- Batch Processing 180
- WHICH Command under Natural Security 180

```
WHICH [ object-name [FULL] ]
      [          ?         ]
```

This command lets you find objects in the libraries used at run time in the Library Search Order (LSO).

Direct Command Syntax

`WHICH`	Displays a window where you can enter an object name with the additional option to select the full output list.
`WHICH` *`object-name`*	Displays the object type, the name of the first library in which the object is found, and the DBID and FNR of this library. See ***Default Output of WHICH Command***.
`WHICH` *`object-name`* `FULL`	Lists all libraries of the library search order. For each listed library, it indicates whether the cataloged or source object is found. If you press PF5 in the output screen, you can start a new search. See ***Full List of WHICH Command***.
`WHICH ?`	Displays the help text of the `WHICH` command.

WHICH Parameter Description

Parameter	Format/Length	Default Value	Description
`object-name`	A8	none	The name of the object you want to search.

Default Output of WHICH Command

If you enter the `WHICH` command without the `FULL` option, the output looks like the following example:

```
Cataloged subprogram USR1002N found in library SYSEXT(12,124).
```

Full List of WHICH Command

If you enter the WHICH command with the FULL option, all libraries of the library search order are listed. The output looks like the following example:

```
12:00:49                ***** NATURAL WHICH COMMAND *****             2023-05-25
User XXXX                  - Search object USR1002N -           Library MYLIB

Library   DBID   FNR Type Save/Catalog Date   Message
-------- ----- ----- ---- ------------------- --------------------------------
MYLIB       12   123
MYLIB01     12   123
MYLIB02     12   123
PROTECT                                       Library access denied
SYSEXT      12   124  N   2022-04-20 07:59:36 Cataloged object found
                      N   2011-05-06 16:37:46 Saved object found
SYSTEM      12   123
SYSTEM      12   124









Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      Help        Exit        New   --    -     +     ++                Canc
```

Explanation of Column Headings

Column	Explanation
Library	The names of libraries in the library search order.
DBID FNR	The database ID and file number of the Natural system file containing the library.
Type	The Natural object type code of the object. See Object Type Code in the *Glossary*.
Save/Catalog Date	The save or catalog date of the object.
Message	The result of the search. An empty line indicates that the object was not found in that library.

Function Keys

The output screen of the WHICH command with FULL option provides the following function keys:

PF Key	Name	Function
PF1	Help	Invoke help.
PF3	Exit	Exit from `WHICH` command.
PF5	New	New search.
PF6	--	Top of list.
PF7	-	Page backward.
PF8	+	Page forward.
PF9	++	End of list.
PF12	Canc	Cancel the `WHICH` command.

Batch Processing

The output of `WHICH` without `FULL` option is the same as the online output.

If you call the `WHICH` command with the `FULL` option, all resulting libraries are printed in one list.

WHICH Command under Natural Security

For information on `WHICH` under Natural Security, see *Command Restrictions* in the *Natural Security* documentation.

64 XREF

```
XREF [ ON    ]
     [ OFF   ]
     [ 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.