Natural

Debugger

Version 9.2.5

October 2026

ADABAS & NATURAL

This document applies to Natural Version 9.2.5 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 © 1979-2026 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: NATMF-DEBUG-925-20261009

Table of Contents

Preface vii
1 About this Documentation 1
Document Conventions 2
Online Information and Support 2
Data Protection 3
2 Debugger Tutorial 5
Prerequisites 6
Fundamentals of Debugging 6
Session 1 - Analyzing a Natural Error 7
Session 2 - Using a Breakpoint 12
Session 3 - Using a Watchpoint 18
Session 4 - Tracing the Logical Flow of Programs 24
Session 5 - Using Statistics about the Program Execution 28
Additional Hints for Using the Debugger 31
Example Sources 35
3 Concepts of the Debugger 39
Session Control and Control Functions 40
Debug Entries/Spies 41
Debug Break Window 43
4 Start the Debugger 45
Debugger under Natural Security 46
Operational Requirements 46
Invoke the Debugger 47
Default Object 48
5 Switch Test Mode On and Off 49
6 Debug Environment Maintenance 51
Set Test Mode ON/OFF 52
Load Debug Environment 53
Save Debug Environment 53
Reset Debug Environment 54
Delete Debug Environment 54
Maintain Debug Environments in Different Libraries 55
7 Spy Maintenance 57
Set Test Mode ON/OFF 58
Activate Spy 58
Deactivate Spy 59
Delete Spy 59
Display Spy 59
Modify Spy 60
8 Breakpoint Maintenance 61
Conditions of Use 62
Set Test Mode ON/OFF 63
Activate Breakpoint 63

Deactivate Breakpoint 64
Delete Breakpoint 64
Display Breakpoint 64
Modify Breakpoint 66
Set Breakpoint 67
Fields and Columns on Breakpoint Screens 68
9 Watchpoint Maintenance 71
Set Test Mode ON/OFF 72
Activate Watchpoint 73
Deactivate Watchpoint 73
Delete Watchpoint 73
Display Watchpoint 74
Modify Watchpoint 76
Set Watchpoint 77
Fields and Columns on Watchpoint Screens 79
10 Call Statistics Maintenance 83
Set Test Mode ON/OFF 84
Set Call Statistics On/Off 84
Display All Objects 85
Display Called Objects 85
Display Non-Called Objects 86
Print Objects 87
11 Statement Execution Statistics Maintenance 89
Set Test Mode ON/OFF 90
Set Statement Execution Statistics ON/OFF/COUNT 90
Delete Statement Execution Statistics 92
Display Statement Execution Statistics 93
Print Statements 96
12 Variable Maintenance 99
Display User-Defined, Global and DB-Related System Variables 100
Display System Variables 103
Modify Variable 104
13 List Object Source 105
Maintain Breakpoints 107
14 Error Handling 109
Errors during Application Execution 110
Errors during Debugger Execution 110
15 Execution Control Commands 113
ESCAPE BOTTOM 114
ESCAPE ROUTINE 114
EXIT 114
GO 115
NEXT 115
RUN 115
STEP 115

STEP SKIPSUBLEVEL 115
STEP SKIPSUBLEVEL n 116
STOP 116
16 Navigation and Information Commands 117
BREAK 118
FLIP 118
LAST 118
OBJCHAIN 118
ON/OFF 119
PROFILE 119
SCAN 120
SCREEN 120
SET OBJECT 120
STACK 120
SYSVARS 121
TEST ON/OFF 121
17 Command Summary and Syntax 123
All Debug Commands 124
Syntax Diagrams 129
18 Preparing Natural for Attached Debugging 133
Introduction 134
Prerequisites for Attached Debugging 134
Example 135

Preface

The debugger is used to detect, locate and correct program errors, test or optimize program execution, or analyze a Natural error that interrupts program execution.

Tutorial	First steps with the debugger.
Concepts of the Debugger	Basic concepts of the debugger.
Start the Debugger	Operational requirements and instructions for invoking the debugger.
Switch Test Mode On and Off	Setting the test mode to activate and deactivate debugging.
Debug Environment Maintenance	Saving and using a predefined debug environment.
Spy Maintenance	Setting, modifying, deleting and activating both breakpoints and watchpoints.
Breakpoint Maintenance	Setting, modifying, deleting and activating breakpoints. Explanations of breakpoint screen contents.
Watchpoint Maintenance	Setting, modifying, deleting and activating watchpoints. Explanations of watchpoint screen contents.
Call Statistics Maintenance	Obtaining statistics about invoked objects.
Statement Execution Statistics Maintenance	Obtaining statistics about executed statement lines.
Variable Maintenance	Displaying and modifying variables.
List Object Source	Displaying an object source.
Error Handling	Handling errors that can occur during application or debugger execution.
Execution Control Commands	Debugger commands for program flow control.
Navigation and Information Commands	Debugger commands for screen navigation, object information and debugger profile settings.
Command Summary and Syntax	All debugger commands and appropriate command syntax.
Preparing Natural for Attached Debugging	Using a debug attach server running under NaturalONE.

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

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

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 Debugger Tutorial

- Prerequisites 6
- Fundamentals of Debugging 6
- Session 1 - Analyzing a Natural Error 7
- Session 2 - Using a Breakpoint 12
- Session 3 - Using a Watchpoint 18
- Session 4 - Tracing the Logical Flow of Programs 24
- Session 5 - Using Statistics about the Program Execution 28
- Additional Hints for Using the Debugger 31
- Example Sources 35

This tutorial introduces the basic features of the debugger and discusses different debugging methods. It takes you through a simple scenario that demonstrates how the debugger can be used to analyze runtime errors and control program execution.

It is important that you work through Sessions 1 to 5 in sequence.

Notes:

1. For ease of use, the tutorial primarily quotes direct commands to demonstrate the debugger features and not the alternative menu functions.
2. For a full description of all debugger features mentioned in this tutorial, refer to the relevant sections in the remainder of the *Debugger* documentation.

Prerequisites

- You should be familiar with programming in Natural.
- Before you start with Session 1, you need to create all example programs (DEBUG1P and DEBUG2P) and subprograms (`DEBUG1N`, `DEBUG2N`, `DEBUG3N` and `DEBUG4N`) provided in the section ***Example Sources*** later in this tutorial. Save and catalog these objects with the system command `STOW`.

Fundamentals of Debugging

The debugger can be used to interrupt the execution flow of a Natural object at a particular debug event and obtain information on the current status of the interrupted object such as the next statement to be executed, the value of a variable and the hierarchy (program levels) of objects called.

You basically need to take the following two major steps to pass control to the debugger for program interruption:

1. Activate the debugger with the system command `TEST ON`.

   This allows the debugger to receive control for each statement to be executed by the Natural runtime system.
2. Set one or more debug entries (breakpoints and watchpoints) for the Natural objects to be executed.

   This allows the debugger to decide when to take over control from the Natural runtime system and interrupt the program execution.

A Natural error always interrupts the program execution. No debug entry is required then, the debugger steps in automatically.

The following is an overview of all possible program interruptions:

Program Interruption	Explanation
Breakpoint	Causes a program interruption for a statement line in a Natural object. The debugger interrupts the program execution whenever the statement line for which a breakpoint is set is to be executed, that is, *before* the statement contained in this line is processed.
Watchpoint	Causes a program interruption for a variable in a Natural object. The debugger interrupts the program execution whenever the contents of the variable for which a watchpoint is set have changed, that is, *after* the statement that references this variable is processed.
Step mode	Steps through the object during the program execution. Step mode is initiated by a debugger command and requires that the debugger previously received control because of a breakpoint or a watchpoint. In step mode, the debugger interrupts the program execution *before* each executable statement contained in this object is processed.
Natural error	Causes an automatic program interruption.

Session 1 - Analyzing a Natural Error

This session describes investigation methods for a Natural error that occurs during program execution.

To simulate a Natural error

- From the `NEXT` prompt, execute `DEBUG1P`.

  The following Natural error message appears: `DEBUG1N 0180 NAT0954 Abnormal termination S0C7 during program execution.`

  The message points to line 180 in the subprogram `DEBUG1N`: `BONUS := SALARY * PERCENT / 100`. This indicates that incorrect values are returned for one or more of the variables referenced. However, at this point, this is no clear evidence of what actually causes the problem; and it could be difficult to determine the cause if the variable values were retrieved from a database (as is typical for employee records).

To activate the debugger for further problem investigation

1 At the `NEXT` prompt, enter the following:

```
TEST ON
```

The message `Test mode started.` indicates that the debugger is activated.

Note: `TEST ON` remains active for the duration of the current session or until you enter `TEST OFF` to deactivate the debugger.

2 Again, execute `DEBUG1P` from the `NEXT` prompt.

A **Debug Break** window similar to the example below appears:

```
+------------------- Debug Break -------------------+
| Break by ABEND SOC7 at NATARI2+2A4-4 (NAT0954)    |
| at line  180 in subprogram DEBUG1N (level 2)      |
| in library DEBUG    in system file (10,32).       |
|                                                   |
|         G   Go                                    |
|         L   List break                            |
|         M   Debug Main Menu                       |
|         N   Next break command                    |
|         R   Run (set test mode OFF)               |
|         S   Step mode                             |
|         V   Variable maintenance                  |
|                                                   |
| Code .. G                                         |
|                                                   |
| Abnormal termination SOC7 during program execution|
| PF2=Step,PF13=Next,PF14=Go,PF15=Menu,PF17=SkipS   |
+---------------------------------------------------+
```

Since a Natural error occurs, the debugger steps in automatically and displays the **Debug Break** window.

Additional information on where the error occurs is displayed at the top of the window: the module (`NATARI2`) in the Natural nucleus (helpful for support), the type of object (`subprogram`) the library (`DEBUG`) and the database ID and file number (`10,32`) of the system file.

The **Debug Break** window also provides debugger functions that can be used, for example, to continue the program execution (**Go** or **Run**), invoke the debugger maintenance menu (**Debug Main Menu**) or activate step mode. You execute a function by using either the appropriate function code or PF key.

To inspect the erroneous statement line

- In the **Code** field, replace the default entry `G` by `L` to execute the **List break** function.

  The source of `DEBUG1N` is displayed:

```
13:48:54              ***** NATURAL TEST UTILITIES *****            2007-09-06
Test Mode ON                - List Object Source -             Object DEBUG1N
                                                               Bottom of data
Co Line Source                                                   Message
__ 0070   2 NUMCHILD  (N2)                                     |
__ 0080   2 ENTRYDATE (D)                                      |
__ 0090   2 SALARY    (P7.2)                                   |
__ 0100   2 BONUS     (P7.2)                                   |
__ 0110 LOCAL                                                  |
__ 0120 1 TARGETDATE  (D)    INIT <D'2009-01-01'>              |
__ 0130 1 DIFFERENCE  (P3.2)                                   |
__ 0140 1 PERCENT     (P2.2) INIT <3.5>                        |
__ 0150 END-DEFINE                                             |
__ 0160 DIFFERENCE := (TARGETDATE - ENTRYDATE) / 365           |
__ 0170 IF DIFFERENCE GE 10        /* BONUS FOR YEARS IN COMPAN | last line
__ 0180   BONUS := SALARY * PERCENT / 100                      | * NAT0954 *
__ 0190 END-IF                                                 |
__ 0200 SALARY := SALARY + 1800    /* SALARY PLUS ANNUAL INCREA |
__ 0210 END                                                    |

Command ===>



Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      Help  Step  Exit  Last  Scan  Flip  -     +     Li Br <     >     Canc
```

  `last line` indicates that the statement contained in line 170 is the last statement that executed successfully.

  The statement in line 180 which causes the problem is highlighted and annotated with `* NAT0954 *`.

  This indicates that the error is caused by either the contents of the variable `SALARY` or `PERCENT`. Most likely, this is `SALARY` since `PERCENT` is properly initialized.

To check the contents of `SALARY`

1 In the Command line, enter the following:

```
DIS VAR SALARY
```

  A **Display Variable** screen similar to the example below appears for the variable `SALARY`:

```
18:59:51              ***** NATURAL TEST UTILITIES *****            2007-09-06
Test Mode ON           - Display Variable (Alphanumeric) -       Object DEBUG1N


Name ...... EMPLOYEE.SALARY
Fmt/Len ... P 7.2
Type ...... parameter
Index .....
Range .....

Position ..
Contents ..


Command ===>


Variable contains invalid data.

Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      Help  Step  Exit  Last  Mod   Flip              Li Br Alpha Hex   Canc
```

The message `Variable contains invalid data.` indicates that the contents of the variable, which seems to be blank, does not match the format of the variable. This becomes clear when you view the hexadecimal representation of the variable contents as described in the next step.

2 Press PF11 (Hex) to display the hexadecimal contents of the variable.

The screen now looks similar to the example below:

```
11:13:33              ***** NATURAL TEST UTILITIES *****            2007-09-06
Test Mode ON           - Display Variable (Hexadecimal) -        Object DEBUG1N


Name ...... EMPLOYEE.SALARY
Fmt/Len ... P 7.2
Type ...... parameter
Index .....
Range .....

Position ..
Contents .. 4040404040


Command ===>


Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      Help  Step  Exit  Last  Mod   Flip              Li Br Alpha Hex   Canc
```

The hexadecimal value shows that the variable is not in packed numeric format, thus leading to a calculation error during the program execution. DEBUG1P obviously provides DEBUG1N with an incorrect value for SALARY.

Tip: You can press PF10 (Alpha) to switch back to the alphanumeric representation.

3 In the Command line, enter the following:

```
GO
```

The command GO returns control from the debugger to the Natural runtime system, which continues the program execution until the end of the program or the next debug event. In this case, there is no additional debug event and the NEXT prompt appears with the known Natural error message.

To correct SALARY in the object source

1 Open DEBUG1P with the program editor and remove the comment sign (*) entered for SALARY := 99000.

2 Save and catalog the program with the system command STOW.

3 Execute DEBUG1P.

The debugger does not interrupt the program though TEST ON is still set. The program executes successfully and outputs a report:

```
Page      1                                                 07-09-06  15:28:06

EMPLOYEE RECEIVES:   100800.00
    PLUS BONUS OF:     3465.00




NEXT                                                            LIB=DEBUG
```

Session 2 - Using a Breakpoint

You can interrupt the program execution at a specific statement line by setting a breakpoint for this line.

To set a breakpoint for a statement line in DEBUG1N

1 At the NEXT prompt, enter the following:

```
TEST SET BP DEBUG1N 170
```

The message Breakpoint DEBUG1N0170 set at line 170 of object DEBUG1N. confirms that a breakpoint with the name DEBUG1N0170 is set for statement line 170 in the DEBUG1N subprogram.

Notes:

1. A breakpoint can only be set for an executable statement. If you try to set a statement for a non-executable statement, an appropriate error message appears.
2. A breakpoint is usually only valid during the current Natural session. If required, you can save a breakpoint for future sessions: see ***Saving Breakpoints and Watchpoints*** in *Additional Hints for Using the Debugger*.

2 Execute DEBUG1P.

The debugger now interrupts the program execution at the statement line, where the new breakpoint is set. The **Debug Break** window appears:

```
+------------------- Debug Break -------------------+
| Break by breakpoint DEBUG1N0170                   |
| at line  170 in subprogram DEBUG1N (level 2)      |
| in library DEBUG    in system file (10,32).       |
|                                                   |
|         G   Go                                    |
|         L   List break                            |
|         M   Debug Main Menu                       |
|         N   Next break command                    |
|         R   Run (set test mode OFF)               |
|         S   Step mode                             |
|         V   Variable maintenance                  |
|                                                   |
| Code .. G                                         |
|                                                   |
|                                                   |
| PF2=Step,PF13=Next,PF14=Go,PF15=Menu,PF17=SkipS   |
+---------------------------------------------------+
```

The window indicates the name of the breakpoint, the corresponding statement line and object and the library that contains the object. It also indicates the operational level of subprogram `DEBUG1N`.

To view the statement indicated in the Debug Break window

- Execute the **List break** function.

  The source of DEBUG1N is displayed on the **List Object Source** screen:

```
11:36:45              ***** NATURAL TEST UTILITIES *****            2007-09-06
Test Mode ON                 - List Object Source -              Object DEBUG1N
                                                                  Bottom of data
Co Line Source                                                      Message
__ 0070   2 NUMCHILD  (N2)                                        |
__ 0080   2 ENTRYDATE (D)                                         |
__ 0090   2 SALARY    (P7.2)                                      |
__ 0100   2 BONUS     (P7.2)                                      |
__ 0110 LOCAL                                                     |
__ 0120 1 TARGETDATE  (D)    INIT <D'2009-01-01'>                 |
__ 0130 1 DIFFERENCE  (P3.2)                                      |
__ 0140 1 PERCENT     (P2.2) INIT <3.5>                           |
__ 0150 END-DEFINE                                                |
__ 0160 DIFFERENCE := (TARGETDATE - ENTRYDATE) / 365              | last line
__ 0170 IF DIFFERENCE GE 10        /* BONUS FOR YEARS IN COMPAN  | DEBUG1N0170
__ 0180   BONUS := SALARY * PERCENT / 100                         |
__ 0190 END-IF                                                    |
__ 0200 SALARY := SALARY + 1800    /* SALARY PLUS ANNUAL INCREA  |
__ 0210 END                                                       |

Command ===>


Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      Help  Step  Exit  Last  Scan  Flip  -     +     Li Br <     >     Canc
```

Statement line 170 indicated in the **Debug Break** window is highlighted. The **Message** column indicates the name of the breakpoint (`DEBUG1N0170`) set for this statement line and the last statement line executed (line 160 as indicated by `last line`).

Remember: A breakpoint interrupts the program execution *before* the statement for which the breakpoint is set is processed.

There are several direct commands you can enter on the **List Object Source** screen to obtain more information on the current object. As an example, you can view all variables as described in the following step.

To display a list of variables contained in `DEBUG1N`

- In the Command line, enter the following:

```
DIS VAR
```

A **Display Variables** screen similar to the example below appears:

```
11:06:13              ***** NATURAL TEST UTILITIES *****            2007-09-06
Test Mode ON          - Display Variables (Alphanumeric) -       Object DEBUG1N
                                                                          All
Co Le Variable Name                    F       Leng Contents              Msg.
    1 EMPLOYEE
__  2 NAME                             A         20 MEIER
__  2 ENTRYDATE                        D            1989-01-01
__  2 SALARY                           P        7.2 99000.00
__  2 BONUS                            P        7.2 *** invalid data ***
__  1 TARGETDATE                       D            2009-01-01
__  1 DIFFERENCE                       P        3.2 20.00
__  1 PERCENT                          P        2.2 3.50



Command ===>


Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      Help  Step  Exit  Last  Zoom  Flip  -     +     Li Br Alpha Hex   Canc
```

The screen lists all variables defined in `DEBUG1N`. You can neglect the remark `invalid data` for `BONUS`. In this case, it is not essential whether `BONUS` is properly initialized since it is used as a target operand only. However, to exercise another debugger command, change the contents of `BONUS` in the following step.

To check and modify the contents of `BONUS`

1 In the **Co** column, next to `BONUS`, enter the following:

```
MO
```

Or:

In the Command line, enter the following:

```
MOD VAR BONUS
```

A **Modify Variable** screen similar to the example below appears:

```
11:29:50              ***** NATURAL TEST UTILITIES *****             2007-09-06
Test Mode ON           - Modify Variable (Alphanumeric) -        Object DEBUG1N


Name ...... EMPLOYEE.BONUS
Fmt/Len ... P 7.2
Type ...... parameter
Index .....
Range .....

Position .. 1
Contents .. ___________



Command ===>


Variable contains invalid data.

Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      Help  Step  Exit  Last  Save  Flip              Li Br Alpha Hex   Canc
```

2 You can use the hexadecimal display to verify that the variable is not in packed numeric format. Press PF10 (Alpha) to switch back to the alphanumeric representation.

3 In the **Contents** field, enter a value in packed numeric format, for example, `12345.00` and press PF5 (Save).

The screen now looks similar to the example below:

```
11:50:00              ***** NATURAL TEST UTILITIES *****            2007-09-06
Test Mode ON           - Display Variable (Alphanumeric) -       Object DEBUG1N


Name ...... EMPLOYEE.BONUS
Fmt/Len ... P 7.2
Type ...... parameter
Index .....
Range .....

Position ..
Contents .. 12345.00



Command ===>


Variable BONUS modified.

Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      Help  Step  Exit  Last  Mod   Flip              Li Br Alpha Hex   Canc
```

A message confirms the modification of **Contents**.

4 Press PF9 (Li Br) or PF3 (Exit).

The **List Object Source** screen appears.

5 In the Command line, enter the following:

```
GO
```

The debugger returns control to the Natural runtime system, which finishes executing DEBUG1P since no further debug event occurs. The report produced by the program is output:

```
Page      1                                                   07-09-06  10:02:51

EMPLOYEE RECEIVES:   100800.00
    PLUS BONUS OF:     3465.00




NEXT                                                                LIB=DEBUG
```

6 Before you continue with the next session, delete all current breakpoints by entering the following at the NEXT prompt:

```
TEST DEL BP * *
```

A message appears confirming that all breakpoint (in this case, only one breakpoint) are deleted.

Session 3 - Using a Watchpoint

`DEBUG1P` and `DEBUG1N` perform a calculation for a single employee's bonus and salary payment. If multiple employee records were processed, you would probably test whether the variable `BONUS` is now updated correctly. This is done by setting a watchpoint for this variable. A watchpoint allows the debugger to interrupt the program execution when the contents of the specified variable change.

To set a watchpoint for the variable `BONUS`

1 At the NEXT prompt, enter the following:

```
TEST SET WP DEBUG1N BONUS
```

The message `Watchpoint BONUS set for variable EMPLOYEE.BONUS.` confirms that a watchpoint is set for the variable `BONUS` in the `DEBUG1N` example subprogram.

Notes:

1. If you enter a debugger direct command in the Command line of a debugger screen, you must omit the keyword `TEST`. For example, instead of `TEST SET WP DEBUG1N BONUS`, you would then enter `SET WP DEBUG1N BONUS` only.
2. A watchpoint is usually only valid during the current Natural session. If required, you can save a watchpoint for future sessions: see ***Saving Breakpoints and Watchpoints*** in *Additional Hints for Using the Debugger*.

2 Execute `DEBUG1P` from the `NEXT` prompt.

The debugger interrupts the program execution at the new watchpoint and invokes the **Debug Break** window:

```
+------------------- Debug Break -------------------+
| Break by watchpoint BONUS                          |
| at line  180 in subprogram DEBUG1N (level 2)       |
| in library DEBUG    in system file (10,32).        |
|                                                    |
|         G   Go                                     |
|         L   List break                             |
|         M   Debug Main Menu                        |
|         N   Next break command                     |
|         R   Run (set test mode OFF)                |
|         S   Step mode                              |
|         V   Variable maintenance                   |
|                                                    |
| Code .. G                                          |
|                                                    |
|                                                    |
| PF2=Step,PF13=Next,PF14=Go,PF15=Menu,PF17=SkipS    |
+----------------------------------------------------+
```

The window indicates that a watchpoint was detected in line 180. This line contains the statement that processes the variable `BONUS`.

The debugger interrupted the program execution *after* the statement for `BONUS` was processed. Only then could the debugger recognize that the contents of the variable had changed.

3 Execute the **List break** function.

The **List Object Source** now looks similar to the example below:

```
16:24:46              ***** NATURAL TEST UTILITIES *****            2007-09-06
Test Mode ON                 - List Object Source -              Object DEBUG1N
                                                                  Bottom of data
Co Line Source                                                      Message
__ 0070   2 NUMCHILD  (N2)                                        |
__ 0080   2 ENTRYDATE (D)                                         |
__ 0090   2 SALARY    (P7.2)                                      |
__ 0100   2 BONUS     (P7.2)                                      |
__ 0110 LOCAL                                                     |
__ 0120 1 TARGETDATE  (D)    INIT <D'2009-01-01'>                 |
__ 0130 1 DIFFERENCE  (P3.2)                                      |
__ 0140 1 PERCENT     (P2.2) INIT <3.5>                           |
__ 0150 END-DEFINE                                                |
__ 0160 DIFFERENCE := (TARGETDATE - ENTRYDATE) / 365              |
__ 0170 IF DIFFERENCE GE 10        /* BONUS FOR YEARS IN COMPAN | DEBUG1N0170
__ 0180   BONUS := SALARY * PERCENT / 100                         | BONUS
__ 0190 END-IF                                                    |
__ 0200 SALARY := SALARY + 1800    /* SALARY PLUS ANNUAL INCREA |
__ 0210 END                                                       |

Command ===>


Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      Help  Step  Exit  Last  Scan  Flip  -     +     Li Br <     >     Canc
```

The statement which references the variable `BONUS` is highlighted and the **Message** column indicates the name of the watchpoint set for the variable.

To check for changes in `BONUS`

1 In the Command line, enter the following:

```
DIS VAR BONUS
```

The **Display Variable** screen appears and displays a value of `3465.00` in the **Contents** field. This shows that the contents of the variable `BONUS` have changed.

2 Press PF3 (Exit) to return to the **List Object Source** screen.

To check for changes in `SALARY`

1 To test the contents of the variable `SALARY` in a later step, set a breakpoint for `SALARY` by entering the following in the **Co** column of line 200:

```
SE
```

From the **List Object Source** screen, a line command such as `SE` is a convenient alternative to using the `SET BP` direct command.

The **Message** column indicates that a breakpoint (BP) is set for line 200:

```
17:55:58              ***** NATURAL TEST UTILITIES *****            2007-09-06
Test Mode ON                - List Object Source -              Object DEBUG1N
                                                                 Bottom of data
Co Line Source                                                     Message
__ 0070   2 NUMCHILD  (N2)                                       |
__ 0080   2 ENTRYDATE (D)                                        |
__ 0090   2 SALARY    (P7.2)                                     |
__ 0100   2 BONUS     (P7.2)                                     |
__ 0110 LOCAL                                                    |
__ 0120 1 TARGETDATE  (D)    INIT <D'2009-01-01'>                |
__ 0130 1 DIFFERENCE  (P3.2)                                     |
__ 0140 1 PERCENT     (P2.2) INIT <3.5>                          |
__ 0150 END-DEFINE                                               |
__ 0160 DIFFERENCE := (TARGETDATE - ENTRYDATE) / 365             |
__ 0170 IF DIFFERENCE GE 10        /* BONUS FOR YEARS IN COMPAN | DEBUG1N0170
__ 0180   BONUS := SALARY * PERCENT / 100                        | BONUS
__ 0190 END-IF                                                   |
__ 0200 SALARY := SALARY + 1800    /* SALARY PLUS ANNUAL INCREA | BP set
__ 0210 END                                                      |

Command ===>


Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      Help  Step  Exit  Last  Scan  Flip  -     +     Li Br <     >     Canc
```

2 In the Command line, enter the following:

```
GO
```

The **Debug Break** window appears:

```
+------------------- Debug Break -------------------+
| Break by breakpoint DEBUG1N0200                   |
| at line  200 in subprogram DEBUG1N (level 2)      |
| in library DEBUG    in system file (10,32).       |
|                                                   |
|         G   Go                                    |
|         L   List break                            |
|         M   Debug Main Menu                       |
|         N   Next break command                    |
|         R   Run (set test mode OFF)               |
|         S   Step mode                             |
|         V   Variable maintenance                  |
|                                                   |
| Code .. G                                         |
|                                                   |
|                                                   |
| PF2=Step,PF13=Next,PF14=Go,PF15=Menu,PF17=SkipS   |
+---------------------------------------------------+
```

3 Execute the **List break** function.

The **List Object Source** screen now looks similar to the example below:

```
10:49:31             ***** NATURAL TEST UTILITIES *****            2007-09-06
Test Mode ON                - List Object Source -              Object DEBUG1N
                                                                 Bottom of data
Co Line Source                                                     Message
__ 0070   2 NUMCHILD  (N2)                                       |
__ 0080   2 ENTRYDATE (D)                                        |
__ 0090   2 SALARY    (P7.2)                                     |
__ 0100   2 BONUS     (P7.2)                                     |
__ 0110 LOCAL                                                    |
__ 0120 1 TARGETDATE  (D)    INIT <D'2009-01-01'>                |
__ 0130 1 DIFFERENCE  (P3.2)                                     |
__ 0140 1 PERCENT     (P2.2) INIT <3.5>                          |
__ 0150 END-DEFINE                                               |
__ 0160 DIFFERENCE := (TARGETDATE - ENTRYDATE) / 365             |
__ 0170 IF DIFFERENCE GE 10        /* BONUS FOR YEARS IN COMPAN | DEBUG1N0170
__ 0180   BONUS := SALARY * PERCENT / 100                        | last line
__ 0190 END-IF                                                   |
__ 0200 SALARY := SALARY + 1800    /* SALARY PLUS ANNUAL INCREA | DEBUG1N0200
__ 0210 END                                                      |

Command ===>


Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      Help  Step  Exit  Last  Scan  Flip  -     +     Li Br <     >     Canc
```

Since this is a breakpoint, the statement that references (and updates) `SALARY` has not yet been executed. As a result, the contents of the variable have not changed.

4 In the Command line, enter `DIS VAR SALARY` to verify that the contents of `SALARY` are unchanged.

The variable screen proves that `SALARY` still contains `99000`, the initial value assigned in `DEBUG1P`.

5 To view the update of the variable contents, step to the next statement by choosing either of the following methods:

In the Command line, enter the following:

```
STEP
```

Or:

Press PF2 (Step).

The screen now looks similar to the example below:

```
13:38:24              ***** NATURAL TEST UTILITIES *****            2007-09-06
Test Mode ON                - List Object Source -             Object DEBUG1N
                                                               Bottom of data
Co Line Source                                                   Message
__ 0070   2 NUMCHILD  (N2)                                     |
__ 0080   2 ENTRYDATE (D)                                      |
__ 0090   2 SALARY    (P7.2)                                   |
__ 0100   2 BONUS     (P7.2)                                   |
__ 0110 LOCAL                                                  |
__ 0120 1 TARGETDATE  (D)    INIT <D'2009-01-01'>              |
__ 0130 1 DIFFERENCE  (P3.2)                                   |
__ 0140 1 PERCENT     (P2.2) INIT <3.5>                        |
__ 0150 END-DEFINE                                             |
__ 0160 DIFFERENCE := (TARGETDATE - ENTRYDATE) / 365           |
__ 0170 IF DIFFERENCE GE 10        /* BONUS FOR YEARS IN COMPAN | DEBUG1N0170
__ 0180   BONUS := SALARY * PERCENT / 100                      |
__ 0190 END-IF                                                 |
__ 0200 SALARY := SALARY + 1800    /* SALARY PLUS ANNUAL INCREA | last line
__ 0210 END                                                    | step mode

Command ===>


Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      Help  Step  Exit  Last  Scan  Flip  -     +     Li Br <     >     Canc
```

You skipped one line and processed the next executable statement in line 200, which updates `SALARY`. The **Message** column indicates that step mode is set. In step mode, the debugger continues the program execution at the next executable statement.

6 In the Command line, enter `DIS VAR SALARY` to check the variable contents.

The **Display Variable** screen appears and displays a value of `100800.00` in the **Contents** field. This proves that the contents of the variable `SALARY` have changed.

7 In the Command line, enter the following:

```
GO
```

The debugger returns control to the Natural runtime system, which finishes executing `DEBUG1P` since no further debug event occurs. The report produced by the program is output.

Session 4 - Tracing the Logical Flow of Programs

This session describes debugging methods you can use to better understand, overview and control a complex Natural application with numerous objects.

The session starts out with instructions for analyzing the logical flow of an application on the statement level. It then demonstrates how breakpoints can be used to find out the sequence in which programs are executed.

The instructions in this session are based on a simple (but sufficient for demonstration) example application that consists of one program (`DEBUG2P`) and three subprograms (`DEBUG2N`, `DEBUG3N` and `DEBUG4N`).

To set a breakpoint at program begin or end

1 Set a breakpoint for `DEBUG2P` by entering the following at the `NEXT` prompt:

```
TEST SET BP DEBUG2P BEG
```

The message `Breakpoint DEBUG2P-BEG set at line BEG of object DEBUG2P.` confirms that a breakpoint is set in `DEBUG2N`.

Using the keyword `BEG` instead of a specific line number has the effect that the breakpoint is set at the beginning of the program, that is, for the first statement to be executed. This can even be the `DEFINE DATA` statement, for example, if an `INIT` clause is used, which generates an executable statement when the program is cataloged.

Tip:

You can also specify the keyword `END` to set a breakpoint for the last statement to be executed. This can be the `END` statement but also the `FETCH` or `CALLNAT` statement.

2 Execute `DEBUG2P`.

The **Debug Break** window appears:

```
+------------------- Debug Break -------------------+
| Break by breakpoint DEBUG2P-BEG                   |
| at line  130 in program DEBUG2P (level 1)         |
| in library DEBUG    in system file (10,32).       |
|                                                   |
|         G   Go                                    |
|         L   List break                            |
|         M   Debug Main Menu                       |
|         N   Next break command                    |
|         R   Run (set test mode OFF)               |
|         S   Step mode                             |
|         V   Variable maintenance                  |
|                                                   |
| Code .. G                                         |
|                                                   |
|                                                   |
| PF2=Step,PF13=Next,PF14=Go,PF15=Menu,PF17=SkipS   |
+---------------------------------------------------+
```

The debugger now steps in at the first breakpoint set for the program.

3 Execute the **List break** function to check the source and see that the debugger now steps in at the first executable statement `NAME := 'MEIER'`.

To step through an application

1 On the **List Object Source** screen, set step mode by either pressing PF2 (Step) or entering `STEP` in the Command line.

The last statement executed is annotated with `last line`. The next statement to be executed is highlighted and annotated with `step mode`.

Tip:

If you do not want the debugger to pause at every single statement but step through an application more quickly, in the `STEP` command, specify the number of statements you want to skip, for example: `STEP 2` or `STEP 10`.

2 Press PF2 (Step) repeatedly until the `CALLNAT` statement is annotated with `step mode`.

3 Continue with PF2 (Step) and execute the `CALLNAT`.

The invoked subprogram `DEBUG2N` is displayed, where the next statement to be executed is highlighted:

```
11:59:19              ***** NATURAL TEST UTILITIES *****            2007-09-06
Test Mode ON                 - List Object Source -              Object DEBUG2N
                                                                     Top of data
Co Line Source                                                     Message
__ 0010 ** SUBPROGRAM DEBUG2N: CALLS 'DEBUG3N' AND 'DEBUG4N'FOR |
__ 0020 ******************************************************* |
__ 0030 DEFINE DATA                                             | step mode
__ 0040 PARAMETER                                               |
__ 0050 1 EMPLOYEE                                              |
__ 0060   2 NAME      (A20)                                     |
__ 0070   2 NUMCHILD  (N2)                                      |
__ 0080   2 ENTRYDATE (D)                                       |
__ 0090   2 SALARY    (P7.2)                                    |
__ 0100   2 BONUS     (P7.2)                                    |
__ 0110 LOCAL                                                   |
__ 0120 1 TARGETDATE  (D)    INIT <D'2009-01-01'>               |
__ 0130 1 DIFFERENCE  (P3.2)                                    |
__ 0140 1 PERCENT     (P2.2) INIT <3.5>                         |
__ 0150 END-DEFINE                                              |

Command ===>


Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      Help  Step  Exit  Last  Scan  Flip  -     +     Li Br <     >     Canc
```

As an alternative, you could skip the `CALLNAT` by entering `STEP SKIP` in the Command line.

You would then only step through the statements in the invoking program `DEBUG2` but not through the statements within an invoked subprogram.

To view the levels at which the objects are executed

1 In the **List Object Source** screen of `DEBUG2N`, enter the following in the Command line:

```
OBJCHAIN
```

A **Break Information** screen similar to the example below appears:

```
13:45:34               ***** NATURAL TEST UTILITIES *****            2007-09-06
                              - Break Information -

No GDA active for the current program.

Break by step mode
at line   30 in subprogram DEBUG2N (level 2)
in library DEBUG    in system file (10,32).
```

In addition to the object information already known, this screen indicates whether the program references a GDA (global data area).

2 Press ENTER to scroll down one page.

The screen now looks similar to the example below:

```
13:46:34              ***** NATURAL TEST UTILITIES *****           2007-09-06
                           - Current Object Chain -

Level Name     Type        Line Library   DBID   FNR
  2   DEBUG2N  Subprogram     0 DEBUG       10    32
  1   DEBUG2P  Program      170 DEBUG       10    32
```

This screen indicates the operational levels at which the objects are executed: subprogram `DEBUG2N` is executed at level 2 and program `DEBUG2P` (which invokes the subprogram) is executed at the superior level 1.

3 Press ENTER.

The **List Object Source** screen appears.

4 In the Command line, enter the following:

```
GO
```

The debugger returns control to the Natural runtime system, which finishes executing `DEBUG2P` since no further debug event occurs. The report produced by the program is output:

```
Page      1                                                  07-09-06  10:04:21

EMPLOYEE RECEIVES:    99300.00
    PLUS BONUS OF:     3565.00




NEXT                                                            LIB=DEBUG
```

5 Delete all breakpoints currently set by entering the following at the `NEXT` prompt:

```
TEST DEL BP * *
```

A message appears confirming that all breakpoints are deleted.

To set breakpoints to follow the program execution

1 At the NEXT prompt, enter the following:

```
TEST SET BP ALL BEG
```

The message `Breakpoint ALL-BEG set at line BEG of object ALL.` appears.

This indicates that you have set a breakpoint for the first executable statement of each object to be executed.

2 Execute `DEBUG2P`.

A **Debug Break** window appears for `DEBUG2P`.

3 Execute the **Go** function repeatedly.

Each time you execute **Go**, the next object invoked is indicated in the **Debug Break** window (`DEBUG2N` first and then `DEBUG3N` and `DEBUG4N`). Thus, you can easily determine which objects are invoked at what point during the program execution. Additionally, for each object, you can apply the menu functions of the **Debug Break** window.

4 When the `NEXT` prompt appears, delete all breakpoints currently set by entering the following:

```
TEST DEL BP * *
```

A message appears confirming that all breakpoints are deleted.

Session 5 - Using Statistics about the Program Execution

You can use the debugger to view statistical information on which objects are called and how often they are called. Additionally, you can find out which statements are executed, and how often.

To check what objects are called during program execution

1 At the NEXT prompt, enter the following:

```
TEST SET CALL ON
```

The message `Call statistics started.` confirms that the statistics function is activated.

2 Execute `DEBUG2P`.

The debugger logs all object calls executed, and the report produced by the program is output.

3 At the `NEXT` prompt, enter the following:

```
TEST DIS CALL
```

A **Display Called Objects** screen similar to the example below appears:

```
10:43:47              ***** NATURAL TEST UTILITIES *****            2007-09-06
Test Mode ON               - Display Called Objects -            Object
                                                                          All
Object   Library  Type         DBID   FNR S/C Ver Cat Date   Time       Calls
*_______ DEBUG___
DEBUG2P  DEBUG    Program        10    32 S/C 4.2 2007-08-30 13:48          1
DEBUG2N  DEBUG    Subprogram     10    32 S/C 4.2 2007-08-30 13:48          1
DEBUG3N  DEBUG    Subprogram     10    32 S/C 4.2 2007-08-30 13:48          1
DEBUG4N  DEBUG    Subprogram     10    32 S/C 4.2 2007-08-30 13:48          1



Command ===>


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

The screen lists all objects executed: the invoking program (DEBUG2P) and all other objects invoked (DEBUG2N, DEBUG3N and DEBUG4N). It also indicates how frequently each object is invoked (CALLS), the type of object called, where the object is stored and under which Natural version, whether source and cataloged objects exist, and when the object was cataloged.

4 Press PF3 (Exit) or PF12 (Canc) until the NEXT prompt appears.

To check which statements are executed during program execution

1 At the NEXT prompt, enter the following:

```
TEST SET XSTAT COUNT
```

The message Statement execution counting started for library/object */*. confirms that the statistics function is activated for all objects contained in the current library and all steplibs concatenated with this library.

2 Execute DEBUG2P.

The debugger logs all statements processed by the program before the report produced by the program is output.

3 At the NEXT prompt, enter the following:

```
TEST DIS XSTAT
```

A **List Statement Execution Statistics** screen similar to the example below appears:

```
11:39:10              ***** NATURAL TEST UTILITIES *****            2007-09-06
Test Mode ON         - List Statement Execution Statistics -     Object
                                                                         All
Co Object   Library  Type        DBID   FNR Obj.Called Exec Exec   %  Total No.
   *_______ *_______                            n Times able uted     Executions
__ DEBUG2P  DEBUG    Program       10    32          1    8    8 100          8
__ DEBUG2N  DEBUG    Subprogram    10    32          1    8    8 100          8
__ DEBUG3N  DEBUG    Subprogram    10    32          1    2    2 100          2
__ DEBUG4N  DEBUG    Subprogram    10    32          1   10    7  70          7



Command ===>


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

The screen lists the number of calls (`Obj. Called n Times`), the number of executable statements (`Exec able`), the number of executed statements (`Executed`), the percentage of executed statements as related to the total number of executable statements (`%`), and the total number of executed statements (`Total No. Executions`).

4 In the **Co** column, next to `DEBUG4N`, enter the following:

```
DS
```

A statistics screen similar to the example below appears:

```
12:11:19              ***** NATURAL TEST UTILITIES *****            2007-09-06
Test Mode ON              - Display Statement Lines -            Object DEBUG4N

Line Source                                                                Count
0010 ** SUBPROGRAM 'DEBUG4N': CALCULATES SPECIAL SALARY INCREASE
0020 ************************************************************
0030 DEFINE DATA
0040 PARAMETER
0050 1 SALARY (P7.2)
0060 END-DEFINE
0070 DECIDE FOR FIRST CONDITION                                              1
0080   WHEN SALARY < 50000                                                   1
0090     SALARY := SALARY + 1800                                   not executed
0100   WHEN SALARY < 70000                                                   1
0110     SALARY := SALARY + 1200                                   not executed
0120   WHEN SALARY < 90000                                                   1
0130     SALARY := SALARY + 600                                    not executed
0140   WHEN NONE                                                             1
0150     SALARY := SALARY + 300                                              1

Command ===>


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

The screen indicates how often a statement was executed and the executable statements that were not processed.

Additional Hints for Using the Debugger

This section provides additional hints for using the debugger.

- Time Stamps of Objects
- Saving Breakpoints and Watchpoints
- Debug Main Menu for Maintenance Functions
- Help for Commands on Maintenance Screens
- Major Functions Available during Program Interruption
- Next Option for Additional Commands During Program Interruption
- Displaying Large Variables and Arrays
- Printing Debugger Statistics

- Using the Debugger in Batch Mode

Time Stamps of Objects

A cataloged object that does not exactly correspond to the source object can cause debugging errors. If you want to guarantee that source and cataloged object correspond to each other, save and catalog them with the system command `STOW`.

For details, see the section ***Operational Requirements***.

Saving Breakpoints and Watchpoints

You can save the breakpoints and watchpoints set in the current session as a debug environment and load this environment for use in a future session. This is helpful if you want to repeatedly test an application with the same debug entries.

For details, see the section ***Debug Environment Maintenance***.

Debug Main Menu for Maintenance Functions

All debugger maintenance functions, such as setting a breakpoint or creating statistics, can be executed by using either a direct command or the maintenance functions provided in the **Debug Main Menu**. You open this menu by entering one of the following:

- `TEST` at a command prompt.
- `MENU` at the Command line of a debugger screen.
- `M` in the **Code** field of the **Debug Break** window.

Help for Commands on Maintenance Screens

For a list of direct commands available on a debugger maintenance screen, press `PF1` (Help) or enter a question mark (?) in the Command line.

A debugger maintenance screen that contains list items usually also provides line commands that can be used to further process an item. You enter a line command in the **Co** column, next to the required item. For a list of valid line commands, enter a question mark (?) in this column.

Major Functions Available during Program Interruption

The major functions available during the program interruption are listed in the following section. They can be executed from either the **Debug Break** window or the Command line of a debugger maintenance screen.

Code in Debug Window	Alternative Direct Command	Function
G	GO	Continues the program execution until the next debug event occurs.
L	LIST BREAK	Lists the object source at the statement line where the debug event occurs.
N	NEXT	Executes the next break command if specified for a breakpoint or watchpoint. See also *Next Option for Additional Commands During Program Interruption*.
R	RUN	Switches test mode off and continues the program execution.
S	STEP	Processes the executable statements line by line.
V	DIS VAR	Displays a list of variables defined for the interrupted object.

Next Option for Additional Commands During Program Interruption

When displaying or modifying a breakpoint or watchpoint, you will notice that the debugger command BREAK is attached to each of them. This command invokes the **Debug Break** window and must not be removed. However, you can specify additional debugger commands to be executed during the program interruption after the BREAK command. An additional command is executed when you enter either the command NEXT in the Command line or the function code N in the **Debug Break** window.

You enter the debugger commands in the **Commands** field of the appropriate breakpoint or watchpoint maintenance screen as shown in the following example:

```
11:38:55              ***** NATURAL TEST UTILITIES *****             2007-09-06
Test Mode ON                - Modify Breakpoint -               Object

Spy number ..............   1
Initial state ........... A (A = Active, I = Inactive)
Breakpoint name ......... DEBUG1P0170_   DBID/FNR ....... 10/32
Object name ............ DEBUG1P_       Library ........ DEBUG
Line number ............ 0170
Label ................... ________________________________
Skips before execution .. ____0
Max number executions ... ____0


Commands ... BREAK_____________________________________________________
             STACK_____________________________________________________
             DIS VAR BONUS_____________________________________________
             __________________________________________________________
             __________________________________________________________
             __________________________________________________________

Command ===>


Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      Help        Exit  Last  Save  Flip                                Canc
```

In the example above, the command `STACK` instructs the debugger to view the Natural stack. The command `DIS VAR BONUS` instructs the debugger to display the specified variable. This is helpful, for example, if you set a breakpoint in a loop and always want to view the value of one particular variable only. You then do not have to enter the `DIS VAR` command repeatedly.

For details, see the description of the field **Commands** in the sections *Fields and Columns on Breakpoint Screens* and *Fields and Columns on Watchpoint Screens*.

Displaying Large Variables and Arrays

The **Display Variable** screen shows all definitions of a variable and displays its contents in alphanumeric or hexadecimal format. For the display features available for large variables, whose contents extend beyond the current screen or variables with array definitions, see the section *Display Variable - Individual*.

Printing Debugger Statistics

You can print the statistical reports produced by the debugger or download them to a PC.

For details, see ***Print Objects*** in the section *Call Statistics Maintenance* and ***Print Statements*** in the section *Statement Execution Statistics Maintenance*.

Using the Debugger in Batch Mode

The debugger is mainly designed for interactive operations in online mode. Although you can, in principle, execute all debugger features in batch mode, processing online operations in batch (for example, the use of PF keys) can require complex batch programming. However, there are also debugger features for which batch processing is a convenient alternative. One example is collecting and printing statistical data about an application as described in ***Example of Generating and Printing Statistics in Batch*** in the section *Batch Processing*.

Example Sources

This section contains the source code of the example programs and subprograms required in Sessions 1 to 5.

Program `DEBUG1P`

```
** PROGRAM 'DEBUG1P: CALLS 'DEBUG1N' FOR SALARY AND BONUS CALCULATION
************************************************************************
DEFINE DATA
LOCAL
1 EMPLOYEE    (A42)
1 REDEFINE EMPLOYEE
  2 NAME      (A20)
  2 NUMCHILD  (N2)
  2 ENTRYDATE (D)
  2 SALARY    (P7.2)
  2 BONUS     (P7.2)
END-DEFINE
NAME      := 'MEIER'
NUMCHILD  := 2
ENTRYDATE := D'1989-01-01'
* SALARY  := 99000
CALLNAT 'DEBUG1N' NAME NUMCHILD ENTRYDATE SALARY BONUS
WRITE 'EMPLOYEE RECEIVES:'   SALARY
WRITE '    PLUS BONUS OF:'   BONUS
END
```

Subprogram DEBUG1N

```
** SUBPROGRAM 'DEBUG1N': CALCULATES BONUS AND SALARY INCREASE
************************************************************************
DEFINE DATA
PARAMETER
1 EMPLOYEE
  2 NAME      (A20)
  2 NUMCHILD  (N2)
  2 ENTRYDATE (D)
  2 SALARY    (P7.2)
  2 BONUS     (P7.2)
LOCAL
1 TARGETDATE  (D)    INIT <D'2009-01-01'>
1 DIFFERENCE  (P3.2)
1 PERCENT     (P2.2) INIT <3.5>
END-DEFINE
DIFFERENCE := (TARGETDATE - ENTRYDATE) / 365
IF DIFFERENCE GE 10        /* BONUS FOR YEARS IN COMPANY
  BONUS := SALARY * PERCENT / 100
END-IF
SALARY := SALARY + 1800    /* SALARY PLUS ANNUAL INCREASE
END
```

Program DEBUG2P

```
** PROGRAM 'DEBUG2P': CALLS 'DEBUG2N'FOR SALARY AND BONUS CALCULATION
************************************************************************
DEFINE DATA
LOCAL
1 EMPLOYEE    (A42)
1 REDEFINE EMPLOYEE
  2 NAME      (A20)
  2 NUMCHILD  (N2)
  2 ENTRYDATE (D)
  2 SALARY    (P7.2)
  2 BONUS     (P7.2)
END-DEFINE
NAME      := 'MEIER'
NUMCHILD  := 2
ENTRYDATE := D'1989-01-01'
SALARY    := 99000
CALLNAT 'DEBUG2N' NAME NUMCHILD ENTRYDATE SALARY BONUS
WRITE 'EMPLOYEE RECEIVES:'   SALARY
WRITE '    PLUS BONUS OF:'   BONUS
END
```

Subprogram DEBUG2N

```
** SUBPROGRAM DEBUG2N: CALLS 'DEBUG3N' AND 'DEBUG4N'FOR SPECIAL RATES
************************************************************************
DEFINE DATA
PARAMETER
1 EMPLOYEE
  2 NAME      (A20)
  2 NUMCHILD  (N2)
  2 ENTRYDATE (D)
  2 SALARY    (P7.2)
  2 BONUS     (P7.2)
LOCAL
1 TARGETDATE  (D)    INIT <D'2009-01-01'>
1 DIFFERENCE  (P3.2)
1 PERCENT     (P2.2) INIT <3.5>
END-DEFINE
DIFFERENCE := (TARGETDATE - ENTRYDATE) / 365
IF DIFFERENCE GE 10        /* BONUS FOR YEARS IN COMPANY
  BONUS := SALARY * PERCENT / 100
END-IF
IF NUMCHILD > 0
  CALLNAT 'DEBUG3N' NUMCHILD BONUS     /* SPECIAL BONUS
END-IF
CALLNAT 'DEBUG4N' SALARY               /* SPECIAL SALARY INCREASE
END                                                                  ↩
```

Subprogram DEBUG3N

```
** SUBPROGRAM 'DEBUG3N': CALCULATES SPECIAL BONUS
************************************************************************
DEFINE DATA
PARAMETER
1 NUMCHILD (N2)
1 BONUS    (P7.2)
END-DEFINE
BONUS := BONUS + NUMCHILD * 50
END
```

Subprogram DEBUG4N

```
** SUBPROGRAM 'DEBUG4N': CALCULATES SPECIAL SALARY INCREASE
************************************************************************
DEFINE DATA
PARAMETER
1 SALARY (P7.2)
END-DEFINE
DECIDE FOR FIRST CONDITION
  WHEN SALARY < 50000
    SALARY := SALARY + 1800
  WHEN SALARY < 70000
```

```
    SALARY := SALARY + 1200
  WHEN SALARY < 90000
    SALARY := SALARY + 600
  WHEN NONE
    SALARY := SALARY + 300
END-DECIDE
END
```

3 Concepts of the Debugger

- Session Control and Control Functions 40
- Debug Entries/Spies 41
- Debug Break Window 43

The debugger takes over control of a Natural session for debugging purposes while a Natural object is executing. This allows you to follow the process flow of a program and perform various program investigations.

You can specify the places in a program where you want the debugger to pause by setting debug entries (breakpoints or watchpoints) for that program.

When program execution pauses, you can review the contents of the variables or parameters used in the program to analyze the program logic, or you can determine the reason for a Natural error.

This section provides general information on the functionality of the debugger.

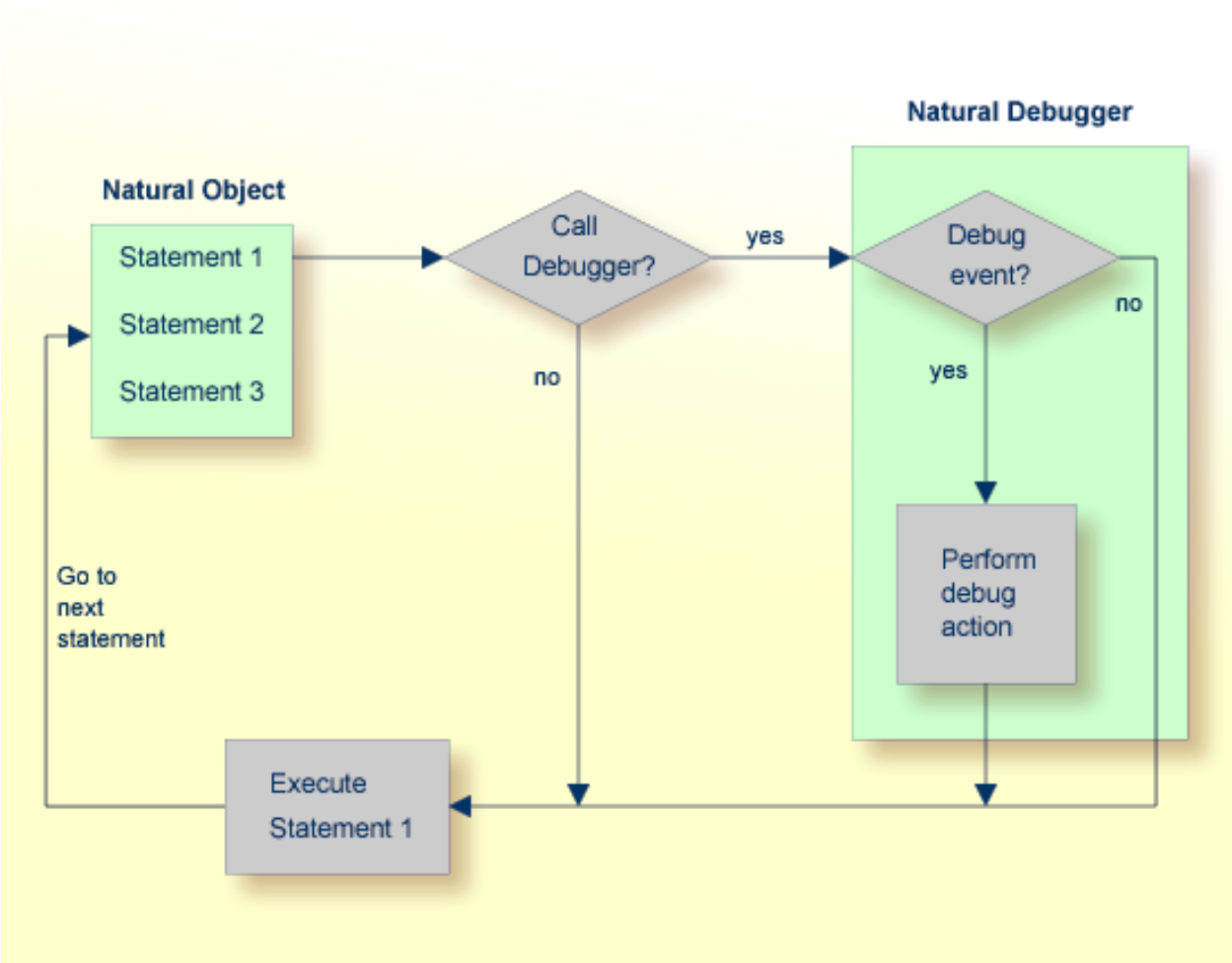
Session Control and Control Functions

The debugger obtains control over a Natural session when test mode is set to `ON` (see ***Switch Test Mode On and Off***). If the profile parameter `DBGERR` is set to `ON` (see the *Parameter Reference* documentation) the debugger is invoked when a Natural error occurs, irrespective of any debug entries and the test mode setting (`ON` or `OFF`).

When the debugger controls a session, the debugger performs one or more of the following functions:

- Checks debug entries.
- Interrupts a Natural object at a statement line for which a breakpoint was set.
- Interrupts a Natural object when the value of a variable for which a watchpoint was set has changed.
- Displays information on the debug entries (watchpoint and/or breakpoint) found.
- Provides statistics on the Natural objects called.
- Provides statistics on the statements executed in a Natural object.
- Interrupts a Natural object when a Natural error occurs. See also the section ***Error Handling***.

The following graphic illustrates an example of the process flow when a Natural object is executed with the debugger:



Debug Entries/Spies

Debug entries are also referred to as spies in the debugger environment. Two types of debug entries (spies) are available: breakpoints and watchpoints.

The following topics are covered below:

- Maintenance and Validation
- Names of Debug Entries
- Initial or Current State
- Counter for Debug Events

- Commands for Debug Entries

Maintenance and Validation

Debug entries for the current debug session can be set, modified, listed, displayed, activated, deactivated and deleted by using the appropriate debugger maintenance functions described in the relevant sections of the debugger documentation. Debug entries can also be saved for future use as described in *Debug Environment Maintenance*.

The validity check of debug entries is either performed immediately when a breakpoint or watchpoint is defined on the appropriate maintenance screen or during program execution.

If a validity check fails during program execution, the note `Check for invalid spy definition` appears in the **Debug Break** window (see ***Debug Break Window***). In addition, the invalid breakpoint or watchpoint is marked on the relevant breakpoint or watchpoint maintenance screens.

When a debug entry is set or modified, Natural internally stores the library, database ID and file number where the object is located. The object may be located in the current library or in one of its steplibs. If an object of the same name is later executed from another library, the corresponding debug entry is not executed.

Names of Debug Entries

The debugger assigns a name and a unique number (spy number) to each debug entry. The name assigned to a debug entry (also referred to as spy name) can be either a name specified by the user or a default name created by the debugger. A debug entry can be selected by its number with the corresponding debugger commands. If more than one debug entry has to be executed at a specific statement line, they are executed in ascending order of their numbers.

Initial or Current State

Each debug entry has an initial state and a current state. Possible values are `A` (active) and `I` (inactive). The initial value is specified when setting or modifying the breakpoint or watchpoint and determines the state of the debug entry at environment start or after reset. During the debug session, the state can be changed with the debug commands `ACTIVATE` and `DEACTIVATE` (see also the syntax diagrams in ***Command Summary and Syntax***).

Counter for Debug Events

Each debug entry has an event count, which is increased every time the debug entry is executed. A debug entry is not executed if the current state is inactive. The event count of the breakpoint or watchpoint is not increased either.

The number of executions of a debug entry can be restricted in two ways:

- A number of skips can be specified before the debug entry is executed. The debug entry is then ignored until the event count is higher than the number of skips specified.
- A maximum number of executions can be specified, so that the debug entry is ignored, as soon as the event count exceeds the specified number of executions.

Commands for Debug Entries

For each debug entry (breakpoint or watchpoint), up to six debug commands can be specified. These commands are executed at execution time of the breakpoint or watchpoint. You can use all debugger commands that can be applied during a debug interrupt. The default command is the `BREAK` command, which displays the **Debug Break** window as shown in the following section.

Caution: If you delete the `BREAK` command when setting a debug entry and you do not enter any command that issues a dialog, there is no way to assume control during program interruption.

Debug Break Window

When the debugger obtains control of the session, a **Debug Break** window similar to the example below appears:

```
+------------------- Debug Break -------------------+
! Break by breakpoint DEBPGM-ALL                     !
! at line  180 in program DEBPGM (level 1)           !
! in library SAG      in system file (10,32).        !
!                                                    !
!         G   Go                                     !
!         L   List break                             !
!         M   Debug Main Menu                        !
!         N   Next break command                     !
!         R   Run (set test mode OFF)                !
!         S   Step mode                              !
!         V   Variable maintenance                   !
!                                                    !
! Code .. G                                          !
! Note: Check for invalid spy definition.            !
!                                                    !
! PF2=Step,PF13=Next,PF14=Go,PF15=Menu,PF17=SkipS    !
+----------------------------------------------------+
```

The **Debug Break** window shows the type and name of the debug entry that has caused the break (that is, the name of the corresponding breakpoint or watchpoint), its source-code line number, and the name of the interrupted Natural object.

In addition, at the bottom of the **Debug Break** window, messages may appear that either indicate a Natural error (see also ***Errors during Application Execution*** in *Error Message Handling*) and/or the possibility of an invalid debug entry.

The functions provided in the **Debug Break** window are described in the following table. For further details, see ***Execution Control Commands***.

Function	Code	Description
Go	**G**	Continues the execution of the Natural object up to the next debug entry specified.
List break	**L**	Lists the code of the Natural object currently active. The last statement executed is highlighted.
Debug Main Menu	**M**	Invokes the **Debug Main Menu** which provides all functions needed to maintain debug entries at which control is to be assumed.
Next break command	**N**	Executes the next command specified for the current breakpoint or watchpoint.
Run (set test mode OFF)	**R**	Continues the execution of the Natural object with test mode set off.
Step mode	**S**	Continues the execution of the Natural object in step mode.
Variable maintenance	**V**	Displays the variables in the Natural object currently active and modifies the contents of these variables.

4 Start the Debugger

- Debugger under Natural Security 46
- Operational Requirements 46
- Invoke the Debugger 47
- Default Object 48

This section describes basic operational requirements and provides a rough guideline on how to proceed when planning to apply the debugger.

Debugger under Natural Security

The use of the debugger can be controlled by Natural Security:

- You can protect the debugger against unauthorized use by disallowing the `TEST` system command, which invokes the debugger; see *Command Restrictions* in the section *Library Maintenance* in the *Natural Security* documentation.
- You can disallow or restrict the use of the debugger as described in *Components of an Environment Profile* in the *Natural Security* documentation.

Operational Requirements

The debugger is only invoked when you execute a cataloged object stored in the current library in the current Natural system file. The debugger is *not* invoked when you execute source code contained in the work area by using the `RUN` command.

Efficient and correct debugging requires that the source code in the source object corresponds to the compiled source code in the cataloged object which can be guaranteed with the system command `STOW`. If you change a source object *after* you cataloged it, it is possible that a debug entry (breakpoint or watchpoint) does not function properly because the referenced statement or variable has changed or no longer exists. When the debugger detects that a source object has an earlier time stamp than the corresponding cataloged object, the following warning appears `Time stamps of source and cataloged object do not match`.

The debugger investigates all Natural objects contained in the current library or in one of its steplibs. The debugger does not investigate Natural objects stored in the Natural system library `SYSLIB` or `SYSLIBS`.

The following restriction applies to the use of the debugger:

- The debugger can only be applied to objects of Natural Version 2.3 and above, but not to Natural objects cataloged with any previous version.
- The debugger supports only debug environments which were created with Natural Version 2.3 and above; debug environments created with any previous version will be ignored. For detailed information on debug environments, see ***Debug Environment Maintenance***.

Batch Processing

Although the debugger is mainly designed for interactive usage in online mode, the debugger commands can also be used for batch execution such as for setting breakpoints or watchpoints.

Note: There are restrictions for batch processing which can cause a debugger command to be rejected. For example, the debugger does not support the commands `++` and `+4`.

Example of Generating and Printing Statistics in Batch

The following is an example of using debugger direct commands in batch mode to generate and print a report about call statistics:

```
//NATBATCH EXEC PGM=NATBAT42,
//   PARM=('INTENS=1,IM=D,CF=$,PRINT=((1-2),AM=STD)')
//STEPLIB  DD DISP=SHR,DSN=NATURAL.V2.TEST.NUCLEUS
//CMPRINT  DD SYSOUT=X
//SYSOUT   DD SYSOUT=X
//CMPRT01  DD SYSOUT=X
//CMSYNIN  DD *
LOGON DEBUGLIB
TEST PROFILE
,,,,CMPRT01
,,,,,,$K3
,,$K3
TEST ON
TEST SET XSTAT COUNT
DEBUG2P
TEST PRINT XSTAT
FIN
/*
```

Invoke the Debugger

To invoke the debugger

1 Establish a debug environment for a Natural object or application:

- Invoke the **Debug Main Menu** by entering the Natural system command `TEST`.

  Or: From within a running application, enter the terminal command `%<TEST`.
- Use the functions of the **Debug Main Menu** to specify debug entries for a Natural object or application:

  Debug environment maintenance
  Spy maintenance

Breakpoint maintenance
Watchpoint maintenance
Call statistics maintenance
Statement execution statistics maintenance
Variable maintenance
List object source

2 Activate the debugger:

- At a command prompt, enter the command `TEST ON`.
- Or: In the **Debug Main Menu**, enter function code `T`.

3 Execute the Natural object or application.

The debugger pauses program execution at the specified debug entries and invokes the **Debug Break** window.

To invoke the debugger for error handling

- At session start, set the profile parameter `DBGERR` to `ON`.

  See also *DBGERR - Automatic Start of Debugger at Runtime Error* in the *Parameter Reference* documentation.

  Or:

  During the session, enter the command `TEST ON` at a command prompt or enter function code `T` in a main debug maintenance menu.

  The debugger invokes the **Debug Break** window when a Natural error occurs.

See also the section ***Error Handling***.

Default Object

The maintenance functions of the debugger as described in the relevant sections refer to objects you specify either in the corresponding name fields of menus or with direct commands. If you do not specify an object name, by default, the debugger assumes the name of the current object as it is displayed in the **Object** field, in the upper right corner of the **Debug Main Menu**. With a default object specified, no object name is required in direct commands and menu options used to specify breakpoints or watchpoints. To change the default object, see the syntax of the command `SET` in the section *Command Summary and Syntax*.

5 Switch Test Mode On and Off

To activate a previously established debug environment, test mode must be set to `ON`.

To set test mode on or off

- In a main debug maintenance menu, enter function code `T` to switch test mode on or off.

  Or:

  Enter one of the following direct commands:

  ```
  TEST ON
  ```

  or

  ```
  TEST OFF
  ```

When executing a Natural object with test mode set to `ON`, the debugger continuously checks all debug entries for any required action.

When executing a Natural object with test mode set to `OFF`, all debug entries are ignored.

The command `TEST`, and with it the whole application, can be protected by Natural Security as described in *Command Restrictions* in the section *Library Maintenance* in the *Natural Security* documentation.

6 Debug Environment Maintenance

- Set Test Mode ON/OFF 52
- Load Debug Environment 53
- Save Debug Environment 53
- Reset Debug Environment 54
- Delete Debug Environment 54
- Maintain Debug Environments in Different Libraries 55

Since a debug environment mainly consists of debug entries, it is established by setting breakpoints and watchpoints as described in the relevant maintenance sections.

Once established, a debug environment can be stored for subsequent usage. The file where debug environments are stored can be specified with the debugger command `PROFILE` (see *Navigation and Information Commands*). You can also delete a debug environment or reset its counters to their initial values.

Note: See also the usage restrictions described in ***Operational Requirements***.

The following items are also part of a debug environment and are therefore saved or loaded every time you save or load a debug environment:

- the test mode setting (`ON` or `OFF`);
- all options that can be set with the debugger command `PROFILE` (except the file for loading or saving debug environments);
- the settings of the **Statement execution statistics maintenance** function (`ON`, `OFF` or `COUNT`).

To invoke the debug environment maintenance function

- In the **Debug Main Menu**, enter function code `E`.

  Or:

  Enter the following direct command:

  ```
  EM
  ```

  The **Debug Environment Maintenance** menu appears.

This section describes the functions provided in the **Debug Environment Maintenance** menu and provides instructions for performing maintenance functions in different libraries.

With each function selected, you must enter the name of the debug environment to be maintained.

Set Test Mode ON/OFF

See the section ***Switch Test Mode On and Off***.

Load Debug Environment

To load a debug environment from your user system file (FUSER)

- In the **Debug Environment Maintenance** menu, enter function code L and the name of an environment.

  Or:

  Enter the following direct command:

  ```
  LOAD ENVIRONMENT name
  ```

  The specified debug environment is loaded.

If you do not specify a name, the default environment with the name Noname is loaded.

Enter an asterisk (*) to obtain a list all available debug environments. On the list, you can mark the desired environment with the line command LO to load it into the debug buffer, or with the line command DE to delete it.

Save Debug Environment

To save a debug environment

- In the **Debug Environment Maintenance** menu, enter function code S and the name of an environment.

  Or:

  Enter the following direct command:

  ```
  SAVE ENVIRONMENT name
  ```

  The specified environment is **reset** (see below) and saved to the file location specified with the debugger command PROFILE (see the section *Navigation and Information Commands*).

If you do not specify a name, the environment is saved with the name Noname.

If a debug environment with the specified name already exists, you are prompted for confirmation to overwrite the old environment.

Reset Debug Environment

The debug environment should be reset before each test run. Resetting the environment leads to the following results:

- The current states of all debug entries are set to their initial states;
- All event counts are set to zero;
- The call statistics in the debug buffer are cleared as described in the section *Call Statistics Maintenance*.

To reset a debug environment

- In the **Debug Environment Maintenance** menu, enter function code `R` and the name of an environment.

  Or:

  Enter the following direct command:

  ```
  RESET ENVIRONMENT name
  ```

  The specified debug environment is reset.

If you do not specify an environment name, the current debug environment is reset.

Delete Debug Environment

To delete a debug environment

1. In the **Debug Environment Maintenance** menu, enter function code `D` and the name of the environment.

   Or:

   Enter the following direct command:

   ```
   DELETE ENVIRONMENT name
   ```

   The confirmation window appears.
2. In the confirmation window, enter `Y` (Yes) to confirm the deletion.

   The debug specified environment is deleted.

If you do not specify an environment name, the current debug environment is deleted.

Maintain Debug Environments in Different Libraries

The `SYSMAIN` utility provides the functions to copy or move debug environments between different libraries and/or system files and to delete, list or rename a debug environment.

When a debug environment has been moved or copied from one library to another, the breakpoints and watchpoints still refer to the old (source) library. You adapt the debug environment to the new (target) library by modifying the corresponding breakpoints (see also ***Modify Breakpoint*** in *Breakpoint Maintenance*) or watchpoints (see also ***Modify Watchpoint*** in *Watchpoint Maintenance*). When you perform the modify function, you do not have to change any of the existing definitions; upon executing the save command (PF5), the library reference automatically changes to the new library as can be seen in the **Library** field entry on the **Modify Breakpoint** or **Modify Watchpoint** screen.

Related Topic:

Processing Debug Environments - SYSMAIN Utility, *Utilities* documentation

7 Spy Maintenance

- Set Test Mode ON/OFF 58
- Activate Spy 58
- Deactivate Spy 59
- Delete Spy 59
- Display Spy 59
- Modify Spy 60

This function is used to activate, deactivate, list or delete all debug entries (spies) that is, breakpoints *and* watchpoints. Besides, **Spy maintenance** is an alternative method of accessing the breakpoint or watchpoint maintenance screens. These screens are explained in the sections *Breakpoint Maintenance* and *Watchpoint Maintenance*.

To invoke the spy maintenance function

- In the **Debug Main Menu**, enter function code `S`.

  Or:

  Enter the following direct command:

  ```
  SM
  ```

  The **Spy Maintenance** menu appears.

The functions provided in the **Spy Maintenance** menu are described in the following section.

Set Test Mode ON/OFF

See the section ***Switch Test Mode On and Off***.

Activate Spy

To set the current state of specified spies to active

- In the **Spy Maintenance** menu, enter function code `A` and a spy number *or* a spy name.

  Or:

  Use the direct command `ACTIVATE`, the syntax of which is described in the section *Command Summary and Syntax*.

If you do not specify a spy number or a spy name, *all* spies (breakpoints and watchpoints) are activated.

Deactivate Spy

To set the current state of specified spies to inactive

- In the **Spy Maintenance** menu, enter function code `B` and a spy number *or* a spy name.

  Or:

  Use the direct command `DEACTIVATE`, the syntax of which is described in the section *Command Summary and Syntax*.

If you do not specify a spy number or a spy name, *all* spies (breakpoints and watchpoints) are deactivated.

Delete Spy

To delete specified spies

- In the **Spy Maintenance** menu, enter function code `C` and a spy number *or* a spy name.

  Or:

  Use the direct command `DELETE`, the syntax of which is described in the section *Command Summary and Syntax*.

If you do not specify a spy number or a spy name, *all* spies (breakpoints and watchpoints) are deleted.

Display Spy

To display specified spies

- In the **Spy Maintenance** menu, enter function code `D` and a spy number *or* a spy name.

  Or:

  Use the direct command `DISPLAY`, the syntax of which is described in the section *Command Summary and Syntax*.

If the specified spy is unique, the **Display Breakpoint** or **Display Watchpoint** screen appears respectively and all specifications of this breakpoint or watchpoint are displayed.

If the specified spy is not unique, a list of the spies concerned is displayed. On the list, you can activate, deactivate, display, modify or delete a spy by marking it with the line command `AC`, `DA`, `DI`, `MO` or `DE` respectively.

If you do not specify a spy number or a spy name, *all* spies (breakpoints and watchpoints) are displayed.

Modify Spy

To modify specified spies

- In the **Spy Maintenance** menu, enter function code `M` and a spy number *or* a spy name.

  Or:

  Use the direct command `MODIFY`, the syntax of which is described in the section *Command Summary and Syntax*.

  If the specified spy is unique, the **Modify Breakpoint** or **Modify Watchpoint** screen appears respectively and the breakpoint or watchpoint specifications can be modified.

  If the specified spy is not unique, a list of the spies concerned is displayed. On the list, you can activate, deactivate, display, modify or delete a spy by marking it with the line command `AC`, `DA`, `DI`, `MO` or `DE` respectively.

If you do not specify a spy number or a spy name, *all* spies (breakpoints and watchpoints) are displayed for selection and modification.

8 Breakpoint Maintenance

- Conditions of Use 62
- Set Test Mode ON/OFF 63
- Activate Breakpoint 63
- Deactivate Breakpoint 64
- Delete Breakpoint 64
- Display Breakpoint 64
- Modify Breakpoint 66
- Set Breakpoint 67
- Fields and Columns on Breakpoint Screens 68

A breakpoint causes the execution of a Natural object to be interrupted at a specific statement line. This section describes how and when to set breakpoints. Note that the maintenance functions described here may also be invoked from an object source by using the **List object source** function.

To invoke Breakpoint Maintenance

- In the **Debug Main Menu**, enter function code `B`.

  Or:

  Enter the following direct command:

  ```
  BM
  ```

  The **Breakpoint Maintenance** menu appears.

This section describes conditions for using breakpoint maintenance, the functions provided in the **Breakpoint Maintenance** menu and the fields and columns contained in a breakpoint screen.

Conditions of Use

A breakpoint is set by specifying the name of the Natural object to be processed and the line number in the object's source code where the breakpoint is to be executed.

Once a breakpoint has been specified, it remains set for the entire Natural session, unless you delete it.

A breakpoint refers to a specific line number in source code. A subsequent change of the source code itself may therefore lead to the breakpoint no longer applying to the desired statement, and thus the Natural object not being interrupted at the desired position. To circumvent this problem with program loops, labels can be set within these loops. Breakpoints set for these labels are adjusted to the correct line number if statement lines are inserted or deleted.

The unique identifier for a breakpoint is the spy number as assigned by the debugger.

Breakpoints cannot be set on comment lines, on any statement line other than the first one (if a single statement occupies more than one program line), and on lines that contain one of the following statements only:

- `AT BREAK OF`
- `AT END OF DATA`
- `AT END OF PAGE`
- `AT START OF DATA`
- `AT TOP OF PAGE`

- BEFORE BREAK
- DECIDE
  See also the usage restrictions described in ***Operational Requirements***.
- DEFINE SUBROUTINE
- DEFINE WINDOW
- FORMAT
- IF NO RECORDS FOUND
- ON ERROR
- OPTIONS

Whether it is possible or not to set breakpoints for lines compiled with the Natural Optimizer Compiler depends on the NODBG option of the OPTIONS statement described in *Switching on the Optimizer Compiler* in the *Natural Optimizer Compiler* documentation.

Set Test Mode ON/OFF

See the section ***Switch Test Mode On and Off***.

Activate Breakpoint

To set the current state of specified breakpoints to active

- In the **Breakpoint Maintenance** menu, enter function code A, an object name and/or a line number.

  Or:

  Use the direct command ACTIVATE, the syntax of which is described in the section *Command Summary and Syntax*.

  If you do not specify an object name or a line number, *all* breakpoints are activated.

Deactivate Breakpoint

To set the current state of specified breakpoints to inactive

- In the **Breakpoint Maintenance** menu, enter function code `B`, an object name and/or a line number.

  Or:

  Use the direct command `DEACTIVATE`, the syntax of which is described in the section *Command Summary and Syntax*.

  If you do not specify an object name or a line number, *all* breakpoints are deactivated.

Delete Breakpoint

To delete specified breakpoints

- In the **Breakpoint Maintenance** menu, enter function code `C`, an object name and/or a line number.

  Or:

  Use the direct command `DELETE`, the syntax of which is described in the section *Command Summary and Syntax*.

  If you do not specify an object name or a line number, *all* breakpoints are deleted.

Display Breakpoint

To display a breakpoint

- In the **Breakpoint Maintenance** menu, enter function code `D`, an object name and a line number.

  If you do not enter an object name, the **default object** (if specified) is used.

  Or:

Use the direct command DISPLAY, the syntax of which is described in the section *Command Summary and Syntax*.

If a breakpoint has been set for the specified object and line number, a **Display Breakpoint** screen with all breakpoint definitions appears similar to the example below:

```
11:16:12              ***** NATURAL TEST UTILITIES *****             2006-02-07
Test Mode ON                - Display Breakpoint -              Object

Spy number ..............   1
Initial state ........... active         Current state .. active
Breakpoint name ......... BRK0130        DBID/FNR ....... 10/32
Object name ............ DEBPGM1        Library ........ SAG
Line number ............ 0130
Label ..................
Skips before execution ..     0
Max number executions ...     0
Number of activations ...     0
Error in definition ..... - none -

Commas ... BREAK





Command ===>
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      Help        Exit  Last  Mod   Flip                                Canc
```

If no unique breakpoint is found, the **List Breakpoints** screen described below appears.

The fields on the **Display Breakpoint** screen are described in ***Fields and Columns on Breakpoint Screens***.

To list breakpoints

- In the **Breakpoint Maintenance** menu, enter function code D, an object name or a line number. You can use asterisk (*) notation to specify a range of object names, for example, ABC*. If you enter an asterisk (*) only, all object names are selected. If you do not enter an object name, the **default object** (if specified) is used.

  Or:

  Use the direct command DISPLAY, the syntax of which is described in the section *Command Summary and Syntax*.

A **List Breakpoints** screen similar to the example below appears which lists all breakpoints set for the specified object(s) or line number:

```
11:41:56              ***** NATURAL TEST UTILITIES *****            2006-01-30
Test Mode ON                  - List Breakpoints -              Object
                                                                          All
Co No. BP Name      Library  Object   Line  DBID   FNR Stat Skips Execs Count E
       *___________ *_______ *_______ 0000             I  C
__   1 BRK0130      SAG      DEBPGM1  0130    10    32 A  A     0     0     0
__   2 BRKPGM3-END  SAG      DEBPGM3  END     10    32 A  A     0     0     0
__   3 BRKPGM3-300  SAG      DEBPGM3  0300    10    32 A  A     0     0     0
__   4 BRKPGM2-400  SAG      DEBPGM2  0400    10    32 A  A     0     0     0
__   5 BRKPGM2-430  SAG      DEBPGM2  0430    10    32 A  A     0     0     0
__   6 BRKPGM1-END  SAG      DEBPGM1  END     10    32 A  A     0     0     0
__   7 BRKPGM1-ALL  SAG      DEBPGM1  ALL     10    32 A  A     0     0     0









Command ===>
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      Help        Exit  Last        Flip  -     +                       Canc
```

The list is sorted in ascending order by the **spy numbers** contained in the **No.** column.

For details on the columns contained in the **List Breakpoints** screen and the line commands that can be executed on any list item, refer to ***Fields and Columns on Breakpoint Screens***.

Modify Breakpoint

To modify a breakpoint

1 In the **Breakpoint Maintenance** menu, enter function code `M`, an object name and a line number. If you do not enter an object name, the **default object** (if specified) is used.

Or:

Use the direct command `MODIFY`, the syntax of which is described in the section *Command Summary and Syntax*.

If a unique breakpoint has been specified, the **Modify Breakpoint** screen appears where you can change the field entries. The fields on the **Modify Breakpoint** screen are described in ***Fields and Columns on Breakpoint Screens***.

If no unique breakpoint is found, the **List Breakpoints** screen (see *Display Breakpoint*) appears.

2 When you have finished editing the breakpoint definitions, choose PF3 (Exit) or PF5 (Save) to save any modification. See also ***Maintenance and Validation*** for information on validity checks of debug entries. If you choose PF12 (Canc), the breakpoint remains unchanged.

Set Breakpoint

To add a breakpoint for a session

- In the **Breakpoint Maintenance** menu, enter function code S, an object name and/or a line number.

  Or:

  Use the direct command SET, the syntax of which is described in the section *Command Summary and Syntax*.

  If you specify not an object name but a valid line number, the name of the **default object** (see the section *Start the Debugger*) is assumed. If no default object is specified, a selection window appears that displays all objects available in the current library.

  If object name and line number are specified correctly, the breakpoint is usually set and confirmed immediately.

  However, a breakpoint set for copycode can only be validated when a program that contains the copycode is executed. See also ***Maintenance and Validation*** for information on validity checks of debug entries.

  The breakpoint receives the default command (BREAK), its initial and current states are set to active and no execution restrictions are specified. Note that if you delete the command BREAK when setting a breakpoint and you do not enter any command that issues a dialog, there is no way for the debugger to receive control during program interruption.

Fields and Columns on Breakpoint Screens

The fields contained in a **Display Breakpoint** or a **Modify Breakpoint** screen and the columns of a **List Breakpoints** screen are described in the following table:

Field	Column	Explanation
Test Mode		Indicates whether test mode is set to `ON` or `OFF`.
Object		Displays the name of the **default object** (see *Start the Debugger*) if specified.
	Co	Input field for any of the following line commands: `AC` Activate breakpoint `DA` Deactivate breakpoint `DI` Display breakpoint `MO` Modify breakpoint `DE` Delete breakpoint `?` List valid line commands `.` Exit breakpoint screen
Spy number	**No.**	A unique number assigned by the debugger when setting the breakpoint.
Initial state	**Stat I**	Specifies the initial state and the current state of the breakpoint: active (`A`) or inactive (`I`).
Current state	**Stat C**	
Breakpoint name	**BP Name**	The name of the breakpoint. Valid values: 1 to 12 characters. The default name for a breakpoint consists of the object name and the line number.
DBID/FNR	**DBID**	The database ID (DBID) and file number (FNR) of the system file where the Natural object is stored.
	FNR	
Library	**Library**	The name of the library that contains the object.
Object name	**Object**	The name of the object available in the current library or one of its steplibs.
Line number	**Line**	The line number of a statement in the object source code. See also ***Conditions of Use*** above. You can also specify `BEG`, `END` or `ALL` as line numbers: `BEG` Specifies the breakpoint that is to interrupt program execution at the first statement executed in an object. `BEG` breakpoints cannot be specified for copycode. `END` Specifies the breakpoint that is to interrupt program execution at the last statement executed in an object, for example, an `END` or a `FETCH` statement.

Field	Column	Explanation
		END breakpoints cannot be specified for copycode. ALL Specifies that a breakpoint is to interrupt program execution at each program line that contains an executable statement.
Label		Refers to a label set earlier in the source code of an object for statements that define processing loops: see also ***Conditions of Use*** above. Valid values: 1 to 32 characters.
Skips before execution	**Skips**	Determines that the breakpoint is not to be executed until the corresponding statement line has been executed a certain number of times. Valid values: 0 (default) to 32767.
Max number executions	**Execs**	Any value greater than zero (0) determines the maximum number of breakpoint executions. Valid values: 0 (default) to 32767.
Number of activations	**Count**	Indicates how many times a breakpoint was activated for the relevant statement line. The counter is reset when a program is started at Level 1.
Error in definition	**E**	Indicates that the statement line in the breakpoint definition cannot be found in the cataloged object during program execution. This error can be caused if the source of an object is changed and recataloged during debugging.
Commands		Up to six debug commands. Enter one command per line. For a summary of all available commands, see ***Command Summary and Syntax***. **Caution:** If you delete the command BREAK when modifying a breakpoint and you do not enter any command that issues a dialog, there is no way for the debugger to receive control during program interruption.

9 Watchpoint Maintenance

- Set Test Mode ON/OFF 72
- Activate Watchpoint 73
- Deactivate Watchpoint 73
- Delete Watchpoint 73
- Display Watchpoint 74
- Modify Watchpoint 76
- Set Watchpoint 77
- Fields and Columns on Watchpoint Screens 79

A watchpoint causes the execution of a Natural object to be interrupted whenever the value of a variable changes. In addition, you can make the interruption dependent on a condition related to a specific variable value as described under **Watchpoint Operators** (see also *Set Watchpoint*) below.

The use of watchpoints allows you to detect unintended alterations of variables caused by objects that contain errors.

A variable is considered to have changed either when its current value differs from the value recorded when the watchpoint was last triggered or when it differs from the initial value. Comparative validation of watchpoint values is restricted to a field length of 253 bytes. For large variables that exceed the maximum length, only the first 253 bytes are used in the comparison.

A watchpoint is defined by specifying the name of the Natural object and the name of the appropriate variable.

The unique identifier for a watchpoint is the spy number assigned by the debugger.

Once a watchpoint has been specified, it remains set for the entire Natural session, unless you delete it.

To invoke the watchpoint maintenance function

- In the **Debug Main Menu**, enter function code `W`.

  Or:

  Enter the following direct command:

  ```
  WM
  ```

  The **Watchpoint Maintenance** menu appears.

This section describes the functions provided in the **Watchpoint Maintenance** menu and the fields and columns contained in a watchpoint screen.

Set Test Mode ON/OFF

See the section ***Switch Test Mode On and Off***.

Activate Watchpoint

To set the current state of specified watchpoints to active

- In the **Watchpoint Maintenance** menu, enter function code `A`, an object name and/or a variable name.

  Or:

  Use the direct command `ACTIVATE`, the syntax of which is described in the section *Command Summary and Syntax*.

If you do not specify an object or a variable (or leave the default asterisk in the **Variable** field), *all* watchpoints are activated.

Deactivate Watchpoint

To set the current state of specified watchpoints to inactive

- In the **Watchpoint Maintenance** menu, enter function code `B`, an object name and/or a variable name.

  Or:

  Use the direct command `DEACTIVATE`, the syntax of which is described in the section *Command Summary and Syntax*.

If you do not specify an object name or a variable (or leave the default asterisk in the **Variable** field), *all* watchpoints are deactivated.

Delete Watchpoint

To delete specified watchpoints

- In the **Watchpoint Maintenance** menu, enter function code `C`, an object name and/or a variable name.

  Or:

Use the direct command DELETE, the syntax of which is described in the section *Command Summary and Syntax*.

If you do not specify an object name or a variable (or leave the default asterisk in the **Variable** field), *all* watchpoints are deleted.

Display Watchpoint

To display a watchpoint

1 In the **Watchpoint Maintenance** menu, enter function code D, an object name and/or a variable name. If you do not enter an object name, the **default object** (if specified) is used.

Or:

Use the direct command DISPLAY, the syntax of which is described in the section *Command Summary and Syntax*.

If a watchpoint has been set for the specified object and variable name, a **Display Watchpoint** screen with all watchpoint definitions appears similar to the example below:

```
10:25:32              ***** NATURAL TEST UTILITIES *****            2006-02-14
Test Mode ON                - Display Watchpoint -             Object

Spy number ..............  12
Initial state ........... active         Current state .. active
Watchpoint name ......... WATCHTEST1     DBID/FNR ....... 10/32
Object name ............ WATCHPGM       Library ........ SAG
Variable name ........... WATCHVARIABLE
Skips before execution ..     0          Format/length .. A 10
Max number executions ...     0          Persistent ..... N   Act.level ...  0
Number of activations ...     0
Error in definition ..... - none -

Commas ... BREAK









Command ===>
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      Help        Exit  Last  Mod   Flip                    Alpha Hex   Canc
```

The fields on the **Display Watchpoint** screen are described in *Fields and Columns on Watchpoint Screens*.

If no unique watchpoint is found, the **List Watchpoints** screen (see below) appears.

2 On the **Display Watchpoint** screen, you can view the condition for watchpoint activation as specified with the watchpoint operator (see also *Watchpoint Operators*):

Choose PF10 (Alpha) to display the operator and/or operand value in alphanumeric format. .

Or:

Choose PF11 (Hex) to display the operator and/or operand value in hexadecimal format.

Choose PF22 (Cmds) to switch back to the default view of the **Display Watchpoint** screen, which contains the **Commands** field.

To list watchpoints

- In the **Watchpoint Maintenance** menu, enter function code `D`, an object name or a variable name. You can use asterisk (*) notation to specify a range of object names and/or variable names, for example, `ABC*`. If you enter an asterisk (*) only, all names are selected. If you do not enter an object name, the **default object** (if specified) is used.

  Or:

  Use the direct command `DISPLAY`, the syntax of which is described in the section *Command Summary and Syntax*.

  A **List Watchpoints** screen similar to the example below appears which lists all watchpoints set for the specified object(s) or variable name:

```
10:14:05              ***** NATURAL TEST UTILITIES *****            2006-02-14
Test Mode ON                  - List Watchpoints -                Object
                                                                     Top of data
Co No. WP Name      Library  Object    DBID   FNR Stat Skips Execs Count P E
       *___________ *_______ *_______             I  C
      *_____________________________________________________________________
__   1 NAME         SAG      DEBPGM      10    32 A  A     0     0     0 N
       EMPLOYEES-VIEW.NAME
__   5 #MAKE        SAG      DEBPGM      10    32 A  A     0     0     0 N
       #MAKE
__  10 LEAVE-DUE    SAG      DEBPGM      10    32 A  A     0     0     0 N
       EMPLOYEES-VIEW.LEAVE-DUE
__  11 WATCHTEST2   SAG      DEBPGM      10    32 A  A     0     0     0 N
       TESTWP
__  12 WATCHTEST1   SAG      WATCHPGM    10    32 A  A     0     0     0 N
       WATCHVARIABLE
__  13 WATCHTEST3   SAG      DEBPGM      10    32 A  A     0     0     0 N
       WPTEST



Command ===>
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      Help        Exit  Last        Flip  -     +                       Canc
```

The list is sorted in ascending order by the **spy numbers** contained in the **No.** column.

For details on the columns contained in the **List Watchpoints** screen and the line commands that can be executed on any list item, refer to ***Fields and Columns on Watchpoints Screens***.

Modify Watchpoint

To modify a watchpoint

1 In the **Watchpoint Maintenance** menu, enter function code `M`, an object name and a variable name. If you do not enter an object name, the **default object** (if specified) is used.

Or:

Use the direct command `MODIFY`, the syntax of which is described in the section *Command Summary and Syntax*.

If a unique watchpoint has been specified, the **Modify Watchpoint** screen appears where you can change field entries. The fields on the **Modify Watchpoint** screen are described in ***Fields and Columns on Watchpoint Screens***.

If no unique watchpoint is found, the **List Watchpoints** screen (see *Display Watchpoint*) appears.

2 On the **Modify Watchpoint** screen, you can change the condition for watchpoint activation as specified with the watchpoint operator (see also *Watchpoint Operators*):

Choose PF10 (Alpha) to modify the operator and/or operand value in alphanumeric format. .

Or:

Choose PF11 (Hex) to modify the operator and/or operand value in hexadecimal format.

Choose PF22 (Cmds) to switch back to the default view of the **Modify Watchpoint** screen, which contains the **Commands** field.

3 When you have finished editing the watchpoint definitions, choose PF3 (Exit) or PF5 (Save) to save any modification. If you choose PF12 (Canc), the watchpoint remains unchanged.

Set Watchpoint

To add a watchpoint for a session

- In the **Watchpoint Maintenance** menu, enter function code S, an object name and a variable name.

  Or:

  Use the direct command SET, the syntax of which is described in the section *Command Summary and Syntax*.

  Or:

  Before executing a Natural object:

  - Invoke the **List Object Source** screen (see *List Object Source*).
  - In the **Source** column, position the cursor at a variable name and choose PF18 (Se Wp).

  If you specify not an object name but a valid variable name, the name of the **default object** (see the section *Start the Debugger*) is assumed. If no default object is specified, a selection window appears that displays all objects available in the current library. If no default object is specified, a selection window appears that displays all objects available in the current library.

  If object name and variable names are specified correctly, the watchpoint is set immediately and a corresponding confirmation message is displayed on the screen. A watchpoint set for a dynamic variable or an X-array is only validated during program execution. See also *Maintenance and Validation* for information on validity checks of debug entries.

  The watchpoint receives the default command (BREAK), its initial and current state are set to active and no execution restrictions are specified. Note that if you delete the default command

BREAK when setting a watchpoint and you do not enter any command that issues a dialog, there is no way for the debugger to receive control during program interruption.

Watchpoint Operators

You can specify a condition for watchpoint activation by entering an operator and an appropriate operand (if relevant) on a watchpoint maintenance screen.

To specify watchpoint operators

1 On the **Set Watchpoint** or **Modify Watchpoint** screen of the selected watchpoint, choose PF10 (Alpha) if you want to specify an operator operand in alphanumeric format.

Or:

On the **Set Watchpoint** or **Modify Watchpoint** screen of the selected watchpoint, choose PF11 (Hex) if you want to specify an operator operand in hexadecimal format.

Two input fields appear in the lower half of the screen.

2 In the left input field, enter one of the watchpoint operators listed in the following table.

In the right input field, if relevant, enter the operand value to be compared with the variable. For watchpoints with operators specified for dynamic variables (alphanumeric or binary), the operand values will be compared from left to right. Since the field length of a dynamic variable varies, up to 253 bytes can be entered as comparative value. If the current length of the dynamic variable is shorter than the maximum comparative length of 253 bytes, the comparison is made only in the current length of the dynamic variable.

Operator	Explanation
MOD	Modification. Activates the watchpoint each time a modification of the variable occurs. This is the default setting.
EQ	Equal to. Activates the watchpoint when the variable has been modified and when the current value of the variable is equal to the specified operand value.
NE	Not equal to. Activates the watchpoint when the variable has been modified and when the current value of the variable is not equal to the specified operand value.
GT	Greater than. Activates the watchpoint when the variable has been modified and when the current value of the variable is greater than the specified operand value.
GE	Greater than or equal to. Activates the watchpoint when the variable has been modified and when the current value of the variable is greater than or equal to the specified operand value.

Operator	Explanation
LT	Less than. Activates the watchpoint when the variable has been modified and when the current value of the variable is less than the specified operand value.
LE	Less than or equal to. Activates the watchpoint when the variable has been modified and when the current value of the variable is less than or equal to the specified operand value.
INV	Invalid contents. Activates the watchpoint each time the value assigned to a variable of the Type N, P, D or T does not comply with the following conditions: N Numeric unpacked. P Packed numeric. D Date range from 1582-01-01 to 2700-12-31. T Time range from 1582-01-01 00:00:00.0 to 2700-12-31 23:59:59.9.

You can choose PF22 (Cmds) to switch back to the default view of the **Set Watchpoint** or **Modify Watchpoint** screen, which contains the **Commands** input field.

3 Choose PF5 (Save) to save the operator definitions.

Or:

Choose PF12 (Canc), to leave the operator definitions unchanged and exit the **Modify Watchpoint** screen.

Fields and Columns on Watchpoint Screens

The fields contained in a **Display Watchpoint** or a **Modify Watchpoint** screen and the columns of a **List Watchpoints** screen are described in the following table:

Field	Column	Explanation
Test Mode		Indicates whether test mode is set to ON or OFF.
Object		Displays the name of the **default object** (see *Start the Debugger*) if specified.
	Co	Input field for any of the following line commands: AC Activate watchpoint DA Deactivate watchpoint DI Display watchpoint MO Modify watchpoint DE Delete watchpoint

Field	Column	Explanation
		? List valid line commands . Exit watchpoint screen
Spy number	**No.**	A unique number assigned by the debugger when setting the watchpoint.
Initial state	**Stat I**	Specifies the initial state and the current state of the watchpoint: active (A) or inactive (I).
Current state	**Stat C**	
Watchpoint name	**WP Name**	The name of the watchpoint. The default name for a watchpoint is the name of the variable concerned. Valid values: 1 to 12 characters. Names that exceed the field size will be truncated after 12 characters. On the **List Watchpoints** screen, the watchpoint name is listed in the first line, above the variable name.
DBID/FNR	**DBID**	The database ID (DBID) and file number (FNR) of the system file where the Natural object is stored.
	FNR	
Library	**Library**	The name of the library that contains the object.
Object name	**Object**	The name of the object available in the current library or one of its steplibs. If you want to specify a system variable as a watchpoint, enter an asterisk (*) in the **Object name** field.
Variable name		The name of a user-defined, global or system variable. If the variable is part of a group, it may be prefixed by the group name. If you want to specify a system variable, enter an asterisk (*) in the **Object name** field. For an array, an index description has to be specified (watchpoints can be defined for single elements only). On the **List Watchpoints** screen, the variable name is listed in the second line, below the watchpoint name. See also ***Variable Maintenance*** for further details.
Skips before execution	**Skips**	Determines that the watchpoint is not to be executed until the condition set for the watchpoint has been fulfilled (see also **Watchpoint Operators**). Valid values: 0 (default) to 32767.
Max number executions	**Execs**	Any value greater than zero (0) determines the maximum number of watchpoint executions. Valid values: 0 (default) to 32767.

Field	Column	Explanation
Number of activations	**Count**	Indicates how many times the watchpoint condition for the variable was met as specified with the **watchpoint operator**. The counter is reset when a program is started at Level 1.
Format/length		The Natural data format and length of the variable, for example, A10.
Persistent	**P**	Marks a watchpoint as persistent. Persistent watchpoints are not restricted to the Natural object for which they are defined, but apply additionally to all subordinate program levels. Persistent watchpoints only make sense for variables that are passed to a subprogram by reference and not `BY VALUE RESULT`: see the relevant parameter description of the `CALLNAT` statement in *Parameters - operand2*, in the *Statements* documentation. **Restriction:** Persistent watchpoints are not allowed for variables defined in a parameter or context clause. Valid value: `Y` (Yes) or `N` (No). `N` is the default.
Act. level		Refers to **Persistent**. Indicates the program level at which a persistent watchpoint was activated automatically.
Error in definition	**E**	Indicates an invalid watchpoint definition. This error may occur if the executing program is recataloged during debugging after the respective variable definition was modified. A watchpoint set for a dynamic variable or an X-array (eXtensible array) is only validated during program execution.
Commands		Up to six debug commands. Enter one command per line. For a summary of all available commands, see ***Command Summary and Syntax***. **Caution:** If you delete the command `BREAK` and you do not enter any command that issues a dialog, there is no way for the debugger to receive control during program interruption.

10 Call Statistics Maintenance

- Set Test Mode ON/OFF 84
- Set Call Statistics On/Off 84
- Display All Objects 85
- Display Called Objects 85
- Display Non-Called Objects 86
- Print Objects 87

This function is used to obtain statistical information on which Natural objects were invoked during the execution of an application, and information on how often an object was invoked. Call statistics are deleted after resetting the debug environment.

To invoke the call statistics maintenance function

- In the **Debug Main Menu** enter function code `C`.

  Or:

  Enter the following direct command:

  ```
  CS
  ```

  The **Call Statistics Maintenance** menu is displayed.

The functions provided in the **Call Statistics Maintenance** menu are explained in the following section whereas all print functions are described in *Print Objects*.

Set Test Mode ON/OFF

See the section *Switch Test Mode On and Off*.

Set Call Statistics On/Off

When executing a Natural object with **call statistics** set to `ON`, all calls made to a specific object are counted and the resulting statistics can afterwards be displayed or printed.

To set call statistics to ON or OFF

- In the **Call Statistics Maintenance** menu, enter function code `C` to activate or deactivate call statistics.

  Or:

  Enter one of the following direct commands:

  ```
  SET CALL ON
  ```

  or

  ```
  SET CALL OFF
  ```

Note: If the **call statistics** function is switched off and no call statistics have been created or call statistics have been deleted by resetting the debug environment, the information stored for statement execution statistics (see *Statement Execution Statistics Maintenance*) is used for display. This allows you to detect the non-invoked Natural objects during the execution of an application.

Display All Objects

This function provides an overview of the call frequency of all objects contained in a library.

To display the call frequency of all objects in a library

- In the **Call Statistics Maintenance** menu, enter function code `1` and a library name.

  Or:

  Enter the following direct command:

  ```
  DISPLAY OBJECT library
  ```

  See also the syntax of `DISPLAY` in *Command Summary and Syntax*.

  If you do not specify a library name, the library where you are currently logged on is assumed by default.

  A **Display Call Statistics** screen similar to the **example screen** shown in **Display Called Objects** appears.

  The **Display Call Statistics** screen lists all objects in the specified library and indicates their call frequency in the **Calls** column on the right-hand side. For each call statement, such as `FETCH` or `CALLNAT`, an entry with the name of the object and a counter variable is written into the debug buffer. The counter is then increased for each call of the corresponding object.

Display Called Objects

The screen invoked by this function corresponds to the **Display Call Statistics** screen, but only the objects that have been invoked are displayed.

To display called objects of a library

- In the **Call Statistics Maintenance** menu, enter function code `2` and a library name.

  Or:

Enter the following direct command:

```
DISPLAY CALL library
```

See also the syntax of DISPLAY in *Command Summary and Syntax*.

The **Display Called Objects** screen appears:

```
16:06:53              ***** NATURAL TEST UTILITIES *****             2002-02-15
Test mode ON               - Display Called Objects -            Object
                                                                          All
Object   Library  Type         DBID   FNR S/C Ver Cat Date   Time       Calls
*_______ SAG_____
MAINPGM  SAG      Program        10    32 S/C 3.1 2002-02-15 11:51          1
SUBPGM   SAG      Subprogram     10    32 S/C 3.1 2002-02-15 11:50          3
EMP-PGM  SAG      Program        10    32 S/C 3.1 2002-01-22 11:49          2
EMPLIND  SAG      Program        10    32 S/C 3.1 2001-08-13 11:18          1
```

If you do not specify a library name, the library where you are currently logged on is assumed by default.

Display Non-Called Objects

The screen invoked by this function corresponds to the **Display Call Statistics** screen, but only the objects that have *not* been invoked are displayed.

To display non-called objects

- In the **Call Statistics Maintenance** menu, enter function code 3 and a library name.

  Or:

  Enter the following direct command:

  ```
  DISPLAY NOCALL library
  ```

  See also the syntax of DISPLAY in *Command Summary and Syntax*.

If you do not specify a library name, the library where you are currently logged on is assumed by default.

For an **example screen**, see *Display Called Objects* above.

Print Objects

With the print functions, you can directly route a generated list of call statistics to a printer or download the list to a PC. You specify a printer as the output device on the **User Profile** screen of the debugger. Use the debugger command `PROFILE` (see the section *Navigation and Information Commands*) to invoke this screen.

If you do not specify a library name, the library where you are currently logged on is assumed by default.

As indicated under *Print Options* below, to invoke one of the print functions, you can enter either a function code in the **Statement Execution Statistics Maintenance** menu, a line command on the **Display Statement Lines** screen, or a direct command at the command prompt.

Print Options

Print Function	Function Code	Direct Command
All Objects	**4**	`PRINT OBJECT library`
Called Objects	**5**	`PRINT CALL library`
Non-Called Objects	**6**	`PRINT NOCALL library`

See also the syntax of `PRINT` in *Command Summary and Syntax*.

Related Topic:

Example of Generating and Printing Statistics in Batch in the section *Batch Processing*

Example of a PC Download

If Entire Connection and Natural Connection are installed at your site, you can download a statistics list to a PC as described in the following instructions.

To download a list to a PC

1 At session start, specify the profile parameter `PRINT` as follows:

```
PRINT=((1),AM=PC)
```

2 After session start, activate the PC connection using the following terminal command:

```
%+
```

3 Invoke and activate the debugger.

4 Invoke the **User Profile** screen by entering the debugger command `PROFILE` (see *Navigation and Information Commands*).

5 On the **User Profile** screen, in the **Output device** field, replace the current entry by `PCPRNT01` and choose PF3 (Exit) to save the settings.

6 Activate the **call statistics** function and execute the application for which you want the debugger to collect statistics data.

7 From the statistics screen, choose a print function.

In the Entire Connection window that appears, you can specify the output file and the PC directory.

11 Statement Execution Statistics Maintenance

- Set Test Mode ON/OFF 90
- Set Statement Execution Statistics ON/OFF/COUNT 90
- Delete Statement Execution Statistics 92
- Display Statement Execution Statistics 93
- Print Statements 96

This function is used to obtain statistical information on which statement lines of invoked Natural objects were executed. The function also provides information on how often an object was invoked and how often a statement line was executed.

Statement execution statistics can be used for the following purposes:

- To detect dead (never gets executed) programming code in an application;
- To estimate the coverage of an application test (how many statement lines have not been executed at least once for testing);
- To locate frequently executed code segments that could have an impact on the application's performance.

To invoke the statement execution statistics maintenance function

- In the **Debug Main Menu**, enter function code `X`.

  Or:

  Enter the following direct command:

  ```
  XS
  ```

  The **Statement Execution Statistics Maintenance** menu is displayed.

The functions provided in the **Statement Execution Statistics Maintenance** menu are explained in the following section whereas all print functions are described in *Print Statements*.

Set Test Mode ON/OFF

See the section ***Switch Test Mode On and Off***.

Set Statement Execution Statistics ON/OFF/COUNT

This function is used to activate statistics about executed statement lines of Natural objects.

This section covers the following topics:

- Setup Options

- Activate and Deactivate Statistics

Setup Options

When executing a Natural object with **statement execution statistics** set to `ON` or `COUNT`, all statement lines executed within a specific object are listed in a statistical report.

With the option `ON`, the debugger only retains whether a specific statement line was executed or not; with the option `COUNT`, it counts how often a statement line was executed. You can specify a library and an object name to restrict statement execution statistics to the desired Natural objects. The default is to collect statistics for all objects of the current library. Asterisk (*) notation is possible.

If you switch statement execution statistics from `ON` to `COUNT` or vice versa, existing statistics are not affected, that is, their status of `ON` or `COUNT` remains.

The statistical data collected is stored in the debug buffer. The amount of storage that is required to store statistical information for a Natural object is approximately

(number of source lines) / 8 + 100 bytes with **statement execution statistics** set to `ON` and
(number of source lines) * 4 + 100 bytes with **statement execution statistics** set to `COUNT`.

If you modify a Natural object by inserting or deleting lines and you do not renumber the object lines before you `STOW` it, the amount of storage required for the object's statistics may increase. To avoid this, set **Auto Renumber** to `Y` (Yes) in your editor profile (see *Editor Profile* in the *Editors* documentation), or use the system command `CATALL` (see the *System Commands* documentation) with the **Renumber source-codes lines** option enabled (this is the default).

You can use the debugger command `PROFILE` (see *Navigation and Information Commands*) to limit the size of the debug buffer. With **statement execution statistics** set to `COUNT`, no statement execution statistics are collected for objects with more than 8000 statement lines.

Statement execution statistics are part of the debug environment; therefore, they are affected by the direct commands `SAVE ENVIRONMENT` and `LOAD ENVIRONMENT` (see also the section ***Debug Environment Maintenance***).

Activate and Deactivate Statistics

This section provides instructions for activating or deactivating statement execution statistics.

You can specify a library and/or an object name to restrict statement execution statistics to the desired Natural objects. The default is to collect statistics for all objects of the current library. Asterisk (*) notation is possible.

To activate statement execution statistics

- In the **Statement Execution Statistics Maintenance** menu, enter function code `S`, the name of a library and/or the name of an object. In the **State** field, change the value to `ON`.

Or:

Enter one of the following direct commands:

```
SET XSTATISTICS ON library (object)
```

or

```
SET XSTATISTICS COUNT library (object)
```

See also the syntax of `SET` in *Command Summary and Syntax*.

If you do not specify a library and/or an object, the statistics data about all objects in your current library are activated.

To deactivate statement execution statistics

- In the **Statement Execution Statistics Maintenance** menu, enter function code `S`, the name of a library and/or the name of an object. In the **State** field, change the value to `OFF`.

  Or:

  Enter the following direct command:

  ```
  SET XSTATISTICS OFF library (object)
  ```

  See also the syntax of `SET` in *Command Summary and Syntax*.

If you do not specify a library and/or an object, the statistics data about all objects in your current library are deactivated.

Delete Statement Execution Statistics

To delete statement execution statistics

- In the **Statement Execution Statistics Maintenance** menu, enter function code `C` and the name of a library and/or the name of an object.

  Or:

  Enter the following direct command:

  ```
  DELETE XSTATISTICS library (object)
  ```

  See also the syntax of `DELETE` in *Command Summary and Syntax*.

If you do not specify a library and/or an object, the statistics data about all objects in your current library are deleted.

Display Statement Execution Statistics

This function invokes a screen with a list of the specified statement execution statistics.

To invoke the List Statement Execution Statistics screen

1 In the **Statement Execution Statistics Maintenance** menu, enter function code `D`.

Or:

Enter the following direct command:

```
DISPLAY XSTATISTICS
```

The **List Statement Execution Statistics** screen is displayed:

```
16:02:01              ***** NATURAL TEST UTILITIES *****            2002-02-15
Test Mode ON        - List Statement Execution Statistics -     Object
                                                                          All
Co Object   Library  Type        DBID   FNR Obj.Called Exec Exec   %  Total No.
   *_______ *_______                            n Times able uted     Executions
__ TEST     SAG      Program       10    32          4   20   17  85         95
__ MAP01    SAG      Map           10    32          6    2    2 100         12
__ SPGM02   SAG      Subprogram    10    32          2    6    2  33          4
__ SAGTEST1 SAG      Program       10    32          2   20   10  50         17
__ DEBPGM   SAG      Program       10    32          1    6    6 100         34
```

For each object, the following information is displayed:

- the call frequency;
- the number of executable statements;
- the number of executed statements;
- the percentage of executed statements as related to the total number of executable statements;
- the total number of executed statements.

A list entry is highlighted if data is missing or possibly inconsistent.

2 On the statistics list, you can mark an item with a line command for further processing:

Line Command	Explanation
DE	Deletes statement execution statistics as described above.
DS	Displays all statement lines.
DX	Displays executed statement lines only.
DN	Displays non-executed statement lines only.
I	Displays information on the cataloged object and errors.
PS	Prints all statement lines.
PX	Prints executed statement lines only.
PN	Prints non-executed statement lines only.

For further information on print functions, see ***Print Statements***.

The following section describes the screens, which can be invoked with the display commands.

- Display All Statement Lines
- Display Executed Statement Lines
- Display Non-Executed Statement Lines

Display All Statement Lines

The **Display Statement Lines** screen shows the object source and indicates whether or not a statement line has been executed.

To invoke the Display Statement Lines screen

- On the **List Statement Execution Statistics** screen, mark an entry with the line command DS.

  Or:

  Enter the following direct command:

  ```
  DISPLAY STATEMENT library (object)
  ```

  See also the syntax of DISPLAY in *Command Summary and Syntax*.

  The **Display Statement Lines** screen appears. If **statement execution statistics** has been set to COUNT, the execution frequency of the statement line is displayed as shown in the example screen below:

```
16:04:01              ***** NATURAL TEST UTILITIES *****             2002-02-15
Test Mode ON              - Display Statement Lines -            Object SAGTEST

Line Source                                                              Count
0200   RD1. READ EMPLOYEES-VIEW BY NAME                                      2
0210       STARTING FROM #NAME-START THRU #NAME-END
0220 *
0230     IF LEAVE-DUE >= 20                                                  1
0240       PERFORM MARK-SPECIAL-EMPLOYEES                          not executed
0250     ELSE                                                      not executed
0260       RESET #MARK                                                       1
0270     END-IF
0280 *
0290     RESET #MAKE #MODEL                                                  1
0300     CALLNAT 'SPGM02' PERSONNEL-ID #MAKE #MODEL                          1
0310 *
0320     WRITE TITLE / '*** PERSONS WITH 20 OR MORE DAYS LEAVE DU            1
0330       / '***    ARE MARKED WITH AN ASTERISK        ***' //
0340     DISPLAY   '//N A M E' NAME                                          2
```

If no unique object has been specified, the **List Statement Execution Statistics screen** is displayed.

Display Executed Statement Lines

The **Display Executed Statement Lines** screen corresponds to the **Display Statement Lines screen**, but only the statement lines that have been executed are displayed.

To invoke the Display Executed Statement Lines screen

- On the **List Statement Execution Statistics** screen, mark an entry with the line command `DX`.

  Or:

  Enter the following direct command:

  ```
  DISPLAY EXEC library (object)
  ```

  See also the syntax of `DISPLAY` in *Command Summary and Syntax*.

  If no unique object has been specified, the **List Statement Execution Statistics screen** is displayed.

Display Non-Executed Statement Lines

The **Non-Executed Statement Lines** screen corresponds to the **Display Statement Lines screen**, but only the statement lines that have not been executed are displayed.

To invoke the Display Non-Executed Statement Lines screen

- On the **List Statement Execution Statistics** screen, mark an entry with the line command `DN`.

  Or:

  Enter the following direct command:

  ```
  DISPLAY NOEXEC library (object)
  ```

  See also the syntax of `DISPLAY` in *Command Summary and Syntax*.

  If no unique object has been specified, the **List Statement Execution Statistics screen** is displayed.

Print Statements

With the print functions, you can directly route a generated list of statement execution statistics to a printer or download the list to a PC. You define a printer as the output device on the **User Profile** screen of the debugger. Use the debugger command `PROFILE` (see the section *Navigation and Information Commands*) to invoke this screen.

If you do not specify a library name, the library where you are currently logged on is assumed by default.

As indicated under *Print Options* below, to invoke one of the print functions, you can either enter a function code in the **Statement Execution Statistics Maintenance** menu, enter a line command on the **Display Statement Lines** screen or enter a direct command.

Print Options

Print Function	Function Code	Line Command	Direct Command
Print statement execution statistics	**1**		`PRINT XSTATISTICS library (object)`
Print all statements	**2**	`PS`	`PRINT STATEMENT library (object)`
Print executed statements	**3**	`PX`	`PRINT EXEC library (object)`
Print non-executed statements	**4**	`PN`	`PRINT NOEXEC library (object)`

See also the syntax of `PRINT` in the section *Command Summary and Syntax*.

Related Topics:

- ***Example of a PC Download*** in *Print Objects* in the section *Call Statistics Maintenance*
- ***Example of Generating and Printing Statistics in Batch*** in the section *Batch Processing*

12 Variable Maintenance

- Display User-Defined, Global and DB-Related System Variables 100
- Display System Variables 103
- Modify Variable 104

This function is used to display and modify variables within the debugger when a Natural object has been interrupted.

For the interrupted Natural object, the **Variable maintenance** function displays user-defined variables, global variables and the database-related system variables `*COUNTER`, `*ISN` and `*NUMBER` together with Natural data formats, lengths and contents.

Display User-Defined, Global and DB-Related System Variables

This section provides instructions for invoking either the **Display Variables** (summary) screen with a list of all variables, or the **Display Variable** (individual) screen with all details on a particular variable.

- Display Variables - Summary
- Display Variable - Individual

Display Variables - Summary

To display a summary of user-defined, global and database-related system variables

- In the **Debug Main Menu** or in the **Debug Break** window, enter function code `V`.

  Or:

  Enter the following direct command:

  ```
  DISPLAY VARIABLE variable,variable,...
  ```

The **Display Variables** (summary) screen provides a list of the variables specified for the interrupted Natural object. Long values may be displayed truncated on the screen. For arrays, only the contents of the first occurrence are displayed.

To switch between alphanumeric and hexadecimal representation of the variable contents, choose PF10 (Alpha) and PF11 (Hex).

To toggle between the truncated display of a variable and the full name display with the group name, variable name and indices (if relevant), choose PF5 (Zoom).

For *`variable`*, a system variable can also be specified. See ***Display System Variables*** for more information.

Display Variable - Individual

To display an individual variable in its entirety

- From the **Display Variables** (summary) screen, select a variable by marking it with the line command `DI`.

  Or:

  Enter the following direct command:

  ```
  DISPLAY VARIABLE variable
  ```

  Or:

  On the **List Object Source** screen, in the **Source** column, position the cursor at a variable name and choose PF18 (Di Va).

  The following restrictions apply when using PF18 (Di Va):

  If a variable name (including the occurrences of an array) spans more than one line, only the contents of the first line are evaluated.
  If an array name is not followed by an index, the entire array is displayed.
  If the index of an array is constant, for example, array (3,2,6), only this occurrence is displayed.
  If the index of an array is variable, for example, array (i,j) or array (3:i), the variables are evaluated before the respective occurrences of the array are displayed.

  Or:

  On the **List Object Source** screen, in the **Source** column, position the cursor at a variable name and choose ENTER.

  When using ENTER, the same restrictions apply as for PF18, see above. However, the variable or array occurrence is displayed in a window instead of using the **Display Variable** (individual) screen, if the index for an array does not denote more than one occurrence. If the index for an array denotes more than one occurrence, data is displayed using the **Display Variable** (individual) screen.

  Or:

  Instead of positioning the cursor manually and choosing ENTER you can also use Entire Connection for ease of use. Here, a double click with the left mouse button positions the cursor and simulates the ENTER key.

The **Display Variable** (individual) screen, or a window appears with all relevant specifications for the particular variable.

If data is displayed using a window and the length of the contents of the variable exceeds 256 bytes, only the first 256 bytes are displayed. For the **Display Variable** (individual) screen there is no such limit and you can navigate through the entire contents of the variable as described in the instructions below.

To display the entire contents of the variable or navigate within the contents

- Choose PF22 to page backward or PF23 to page forward.

  Or:

  In the **Position** field, enter a numeric value to start the display at a particular position.

You can choose PF10 (Alpha) and PF11 (Hex) to switch between alphanumeric and hexadecimal representation of the variable contents.

To display all occurrences of an array using screen functions

- From the **Display Variables** screen, select a variable by marking it with the line command `DI`.

  Or:

  Choose PF7 (-) and PF8 (+) to page between the individual occurrences.

To display one or more occurrences of an array using direct commands

- Use the following direct command:

  ```
  DISPLAY VARIABLE variable-name(index-specification)
  ```

  where *`variable-name`* denotes the name of the variable, and *`index-specification`* denotes any of the following: an index notation, an index range, or asterisk (*) for all occurrences of a dimension. Variables that are part of the *`index-specification`* are evaluated before the respective occurrences are displayed.

  Examples:

`DISPLAY VARIABLE ARRAY1(*)`	One-dimensional array: Displays all occurrences of the one-dimensional array `ARRAY1`.

`DISPLAY VARIABLE ARRAY1(1)`
 	
or
 	
`DISPLAY VARIABLE ARRAY1`
	One-dimensional array:
	
	Displays the first occurrence of the one-dimensional array `ARRAY1`.

`DISPLAY VARIABLE ARRAY2(2,3:4)`	Two-dimensional array: Displays the second occurrence of the first dimension and the index notation of the second dimension of the two-dimensional array `ARRAY2`.
`DISPLAY VARIABLE ARRAY3(1,3:4,*)`	Three-dimensional array: Displays the first occurrence of the first dimension, the index notation of the second dimension, and all occurrences of the third dimension of the three-dimensional array `ARRAY3`.
`DISPLAY VARIABLE ARRAY4(I,J + 1)`	Two-dimensional array: Displays the occurrence of the two-dimensional array `ARRAY4` specified by the value of variable `I` and the value of the expression `J + 1`.

Display System Variables

To display system variables (except database-related system variables)

- Enter the following direct command:

  ```
  SYSVARS
  ```

  The **System Variables** screen appears with a limited set of system variables.

To display a single system variable

- Use the following direct command:

  ```
  DISPLAY VARIABLE system-variable-name
  ```

  where *`system-variable-name`* is the name of the system variable which can also be displayed using the SYSVARS direct command.

For variables of the type Handle, the name of the class of the instance that the Handle refers to is displayed in alphanumeric representation. If the class name is not available, the Globally Unique Identifier (GUID) is displayed instead. If the class was defined within Natural, the class name or GUID is suffixed with `(NAT)`.

The contents of properties of an instance of a class cannot be displayed within the debugger.

Modify Variable

This function does not apply to system variables.

This function is used to change the value of user-defined and global variables and the database-related system variables.

To modify the contents of a variable from the Modify Variable screen

1 Invoke the **Modify Variable** screen by marking the variable with the line command `MO`.

   Or:

   On the **Display Variable** screen, choose PF5 (Mod).

2 On the **Modify Variable** screen, in the **Contents** field, change the value of the variable.

   The new contents must be valid for the Natural data format of the modified variable since the format of a variable cannot be modified within the debugger.

   On the **Modify Variable** screen, you can toggle between alphanumeric and hexadecimal representation of the variable value using PF10 (Alpha) and PF11 (Hex).

To modify the contents of a variable via direct command

- Enter the following direct command:

```
MODIFY VARIABLE variable = new value
```

  A message appears that confirms modification of the variable value.

Note: The **Modify Variables** function or the `MODIFY VARIABLE` command can be disallowed by Natural Security as described in *Components of an Environment Profile* in the *Natural Security* documentation.

13 List Object Source

- Maintain Breakpoints 107

This function is used to display the source code of an object and maintain breakpoints. For you to be able to use **List object source**, the corresponding source must be in your current library or in one of its steplibs.

To list the source code of an object

- In the **Debug Main Menu**, enter function code `L` and an object name.

  Or:

  Enter the following direct command:

  ```
  LIST object
  ```

  See also the syntax of `LIST` in *Command Summary and Syntax*.

  The **List Object Source** screen appears and the object source is displayed with all current breakpoints listed in the **Message** column on the right-hand side of the screen.

  Choose PF7 (-) or PF8 (+) to scroll up or down one page.

If you execute a Natural object, the debugger interrupts execution at each breakpoint or watchpoint you have set and the **Debug Break** window appears (see ***Debug Break Window*** in *Concepts of the Debugger*).

To list the source code of an interrupted Natural object

- From the **Debug Break** window, choose function code `L` for **List break**.

  Or:

  If relevant, on a debugger screen, choose PF9 (Li Br) or enter the following direct command:

  ```
  LIST BREAK
  ```

  The **List Object Source** screen appears with the source code of the object displayed at the position where a break (breakpoint or watchpoint) occurred. The name of the breakpoint or watchpoint is displayed in the **Message** column on the right-hand side of the screen. The corresponding source code line is highlighted.

Maintain Breakpoints

The **List object source** function may be used to invoke or directly execute breakpoint maintenance functions from within an object source. For instructions on how to set breakpoints and general information on breakpoints, see **Conditions of Use** in *Breakpoint Maintenance*.

To invoke a breakpoint maintenance function from an object source

1 In the **Debug Main Menu**, enter function code `L` and an object name.

Or:

Enter the following direct command:

```
LIST object
```

See also the syntax of `LIST` in *Command Summary and Syntax*.

The source code of the specified object is displayed.

The names of breakpoints already set are displayed in the **Message** column on the right-hand side of the screen.

To navigate in the source list, enter one of the following commands in the command line:

- + (plus sign) or - (minus sign) to scroll down or up one page,
- `TOP` to scroll to the beginning,
- `BOTTOM` to scroll to the end,
- `LEFT` to scroll to the left,
- `RIGHT` to scroll to the right.

2 In the object source, mark the line(s) desired with any of the commands listed below:

Line Command	Explanation
AC	Activates breakpoints.
DA	Deactivates breakpoints.
DE	Deletes breakpoints.
DI	Displays breakpoints.
MO	Goes to the **Modify Breakpoint** maintenance screen.
SE	Sets breakpoints.
SM	Goes to the **Set Breakpoint** maintenance screen.

Upon successful command execution, a corresponding message is displayed in the **Message** column on the right-hand side of the screen.

14 Error Handling

- Errors during Application Execution 110
- Errors during Debugger Execution 110

This section provides information on handling errors when using the debugger.

Errors during Application Execution

You can use the debugger to analyze any Natural system error that interrupts program execution. With test mode set to `ON` (see ***Switch Test Mode On and Off***) or `DBGERR` set to `ON` (see the *Parameter Reference* documentation), the debugger takes control if an error occurs. In this case, a **Debug Break** window similar to the example below appears:

```
+------------------- Debug Break -------------------+
! Break by NATURAL error 1316                        !
! at line   60 in program SAGTEST (level 1)          !
!                                                    !
!         G   Go                                     !
!         L   List break                             !
!         M   Debug Main Menu                        !
!         N   Next break command                     !
!         R   Run (set test mode OFF)                !
!         S   Step mode                              !
!         V   Variable maintenance                   !
!                                                    !
! Code .. G                                          !
!                                                    !
! Index not within array structure.                  !
! PF2=Step,PF13=Next,PF14=Go,PF15=Menu,PF17=SkipS    !
+----------------------------------------------------+
```

Using the **List break** function, you can display the source code of the program at the position where the last statement was executed. The Natural error number is displayed in the **Message** column on the right-hand side of the screen and the corresponding source code line is highlighted.

You can then, for example, review the contents of the variables in the program to determine the reason for the error.

Errors during Debugger Execution

If an error is detected while debugging an application, the debugger will terminate and invoke a window with an error message similar to the example shown below:

```
+------------------ NATURAL Debug Error ---------------------------+
! NATURAL error 3009 has occurred in the NATURAL Debugger.         !
! Last transaction backed out of database 10. Subcode 3            !
!                                                                  !
! Error occurred on level 5 in line 4150 in                        !
! subprogram DBGTEST in library TEST.                              !
! DBGTEST has been loaded from FNAT=(10,932).                      !
! DBGTEST has been cataloged on 2005-04-12 14:43:07.               !
!                                                                  !
! Debugging terminates.                                            !
! Pass this error to application for error processing ? (Y/N): N   !
+------------------------------------------------------------------+
```

If you confirm this error message with an `N` (No - this is the default setting), the following happens:

- The debugger stops debugging and sets the test mode to `OFF`.
- The Natural runtime system ignores the error and continues executing the application.

If you confirm this message with a `Y` (Yes), the following happens:

- The debugger stops debugging and sets the test mode to `OFF`.
- The Natural runtime system reacts to the error and passes it to the application:

  If an `ON ERROR` statement (see the *Statements* documentation) is used, the application determines how to proceed after an execution time error occurs. For example, in the case of a NAT3009 where a transaction is backed out of a database, the application can take appropriate action.

  If no `ON ERROR` statement is used, the Natural runtime system terminates application execution and returns to a Natural command prompt.

15 Execution Control Commands

- ESCAPE BOTTOM 114
- ESCAPE ROUTINE 114
- EXIT 114
- GO 115
- NEXT 115
- RUN 115
- STEP 115
- STEP SKIPSUBLEVEL 115
- STEP SKIPSUBLEVEL n 116
- STOP 116

This section describes the direct commands the debugger provides for controlling the program flow during a debugging session. For a summary of all commands available with the debugger, refer to ***Command Summary and Syntax***.

The commands listed below only apply when the debugger interrupts program execution.

ESCAPE BOTTOM

This command can only be used when a Natural object has been interrupted within a processing loop.

When you enter this command, the interrupted Natural object will be continued with the first statement following the processing loop.

Note: This command can be disallowed by Natural Security as described in *Components of an Environment Profile* in the *Natural Security* documentation.

ESCAPE ROUTINE

When you enter this command, processing of the interrupted Natural object will be stopped and processing will continue with the object from which the interrupted Natural object was invoked; it will continue with the statement following the corresponding `CALLNAT`, `PERFORM` or `FETCH RETURN` statement.

If you apply the command `ESCAPE ROUTINE` to a main program, Natural ends the program and returns to the command mode.

Note: This command can be disallowed by Natural Security as described in *Components of an Environment Profile* in the *Natural Security* documentation.

EXIT

If you are displaying the **Debug Main Menu** and want to invoke the exit function, choose PF3 (Exit) or enter the execution control command `EXIT`, the debugger returns either to the calling program (that is, to the interrupted Natural object which is then continued) or to a command prompt, if the debugger has been invoked with the direct command `TEST`, or to the corresponding input field if it has been invoked by the terminal command `%<TEST`. However, if a breakpoint or watchpoint is currently active, the next command of this breakpoint or watchpoint is executed.

If you are not in the **Debug Main Menu** and enter the direct command `EXIT` or choose PF3 (Exit), you leave the current function and return to the previous step of your debugging session.

GO

When you enter the direct command `GO` (or choose PF14), the debugger returns control to the execution of the interrupted Natural object. If a breakpoint or watchpoint was active at the time the Natural object was interrupted, the remaining commands of this break or watchpoint are *not* executed.

NEXT

When you enter the direct command `NEXT` (or choose PF13), the next command specified for a breakpoint or watchpoint is executed. If no further command has been specified, program execution continues.

RUN

When you enter the direct command `RUN`, test mode is switched off and program execution continues, without investigating any further breakpoints and watchpoints.

STEP

When you enter the direct command `STEP`, an interrupted Natural object is continued for *n* executable statement. The default value for *n* is 1.

STEP SKIPSUBLEVEL

When you enter the direct command `STEP SKIPSUBLEVEL` upon a statement which invokes another object (for example, `CALLNAT`), processing is continued with the next executable statement in the current object instead of the first executed statement in the invoked object).

If this command is applied to a statement that does not invoke another object, the debugger reacts as if the command `STEP` had been entered.

STEP SKIPSUBLEVEL n

With the command `STEP SKIPSUBLEVEL`, you can specify a superior level number *n*. Step mode then continues within the next object at the specified level. For example: If you enter `STEP SKIPSUBLEVEL 2` in an object at level 4, you continue step mode in the object at level 2.

Object level information can be obtained with the command `OBJCHAIN` as described in the section *Navigation and Information Commands*.

STOP

When you enter the direct command `STOP`, both the debugger and any interrupted Natural object are terminated.

Note: This command can be disallowed by Natural Security as described in *Components of an Environment Profile* in the *Natural Security* documentation.

16 Navigation and Information Commands

- BREAK 118
- FLIP 118
- LAST 118
- OBJCHAIN 118
- ON/OFF 119
- PROFILE 119
- SCAN 120
- SCREEN 120
- SET OBJECT 120
- STACK 120
- SYSVARS 121
- TEST ON/OFF 121

This section describes the direct commands the debugger provides for navigating through the debugging areas, scrolling screen displays, obtaining various information on objects and variables, and specifying profiles. For a summary of all commands available with the debugger, refer to *Command Summary and Syntax*.

BREAK

The command `BREAK` is the default command which is automatically set when creating a new debug entry. It displays the **Debug Break** window described in *Debug Break Window* in the section *Concepts of the Debugger*.

When the command `BREAK` is deleted upon modification of the corresponding debug entry, no **Debug Break** window appears. However, other specified commands are executed and the event count is increased.

FLIP

The command `FLIP` switches between the display of the two PF-key lines (PF1 to PF12 and PF13 to PF24).

LAST

The command `LAST` displays the command last entered. The last three commands are stored and can be recalled.

OBJCHAIN

The command `OBJCHAIN` can only be used when a Natural object has been interrupted.

This command displays the objects on the current level and all superior levels, as well as the current GDA (global data area), if applicable, and provides information on the interruption.

ON/OFF

When you enter the command `ON` or `OFF` in the debugger, test mode is switched on or off respectively. See also ***TEST ON/OFF***.

PROFILE

The command `PROFILE` displays the **User Profile** screen where you can modify the profile of the debugger.

User Profile Screen

The **User Profile** screen provides the following options:

Option	Explanation
Reset debug environment automatically on exit	Specifies an automatic reset of your current debug environment once you exit the debugger. The default is `N` (No).
File for loading/saving debug environments	Specifies to/from which system file debug environments are to be saved/loaded: `FUSER` (default), `FNAT` or `SPAD` (scratch-pad file).
Confirm EXIT/CANCEL before execution	Specifies a confirmation of an `EXIT` or `CANCEL` command before execution. The default is `N` (No).
Stack unknown commands	Specifies that any unknown debug command which is entered (for example, the name of a called program) is to be stacked. If so, once you enter an unknown debug command, you immediately exit the debugger and the command is executed. If this option has not been specified, an unknown debug command leads to a corresponding error message. The default is `Y` (Yes).
Output device	Specifies a printer for the functions **Call statistics maintenance** (see ***Print Objects***) and **Statement execution statistics maintenance** (see ***Print Statements***). The default value is `HARDCOPY`. If you want to route the output to another printer, replace `HARDCOPY` by a valid printer name provided by your Natural system administrator.
Maximum debug buffer size in KB	Specifies the maximum size (in kilobytes) of the debug buffer. The debug buffer is automatically enlarged as required, but only up to the specified maximum. Enter `0` to indicate no limit or enter a value from 4 - 16384 (must be a multiple of 4). If the limit would be exceeded, no further debug entries can be defined and no additional call or statement execution statistics entries are generated.

SCAN

Only applies to the **List object source** function (see *List Object Source*).

This command searches for a string of characters within an object source:

- `SCAN` searches for the value specified which may be delimited by blanks or any characters that are neither letters nor numeric characters.
- `SCAN ABS` results in an absolute scan of the source code for the specified value regardless of what other characters may surround the value.

See also the syntax diagrams in *Command Summary and Syntax*.

SCREEN

When you enter the command `SCREEN` upon interruption of a Natural object, the current screen output of the interrupted Natural object is displayed. ENTER takes you back to debug mode.

SET OBJECT

The command `SET OBJECT` changes the name of the **default object** as described in the relevant section in *Start the Debugger*. See also the syntax of `SET` in the section *Command Summary and Syntax*.

STACK

When you enter the command `STACK`, the contents of the entry at the top of the Natural stack is displayed. Up to 15 individual top entry elements can be displayed. Elements longer than 55 characters are truncated and marked with an asterisk (*).

Note: An error message is displayed if any single element is longer than 249 characters.

SYSVARS

When you enter this command, the current values of a limited set of system variables are displayed.

TEST ON/OFF

The command `TEST ON` or `TEST OFF` switches test mode on or off respectively. In the debugger, you only need to enter `ON` or `OFF` as described above.

Note: The `TEST` command can be disallowed by Natural Security as described in *Command Restrictions* in the section *Library Maintenance* in the *Natural Security* documentation.

17 Command Summary and Syntax

- All Debug Commands 124
- Syntax Diagrams 129

This section describes all debugger commands that directly execute debug functions or navigate in debugger screens.

For an explanation of more complex command structures with user-defined operands, see *Syntax Diagrams* below.

All Debug Commands

The debug commands listed in the table below can be entered in the command line of any debugger screen. An underlined portion of a debug command or subcommand represents its minimum abbreviation.

Command	Subcommand(s)	Explanation
-		Scrolls one page up in a list.
--		Scrolls to the beginning of a list.
TOP		
+		Scrolls one page down in a list.
++		Scrolls to the end of a list.
BOTTOM		
ACTIVATE (**syntax** below)	BREAKPOINT or BP	**Activates breakpoints** as described in *Breakpoint Maintenance*.
	SPY	Activates breakpoints *and* watchpoints: see also **Activate Spy** in *Spy Maintenance*.
	WATCHPOINT or WP	**Activates watchpoints** as described in *Watchpoint Maintenance*.
BM		Invokes the **Breakpoint Maintenance** menu described in *Breakpoint Maintenance*.
BREAK		Displays the **Debug Break** window: see also BREAK in *Navigation and Information Commands*.
CANCEL		Cancels the current operation and/or exits screens without saving modifications.
DBLOG	A or Q	Invokes the DBLOG utility (see the *Utilities* documentation) from within the debugger. To specify a database environment, use one of the subcommands: ■ A = Adabas (this is the default)

Command	Subcommand(s)	Explanation
		■ Q = SQL **Note:** During a debug interrupt, you can only specify one of the subcommands listed above.
DEACTIVATE or DA (**syntax** below)	BREAKPOINT or BP	**Deactivates breakpoints** as described in *Breakpoint Maintenance*.
	SPY	Deactivates breakpoints *and* watchpoints: see also ***Deactivate Spy***.
	WATCHPOINT or WP	**Deactivates watchpoints** as described in *Watchpoint Maintenance*.
DELETE (**syntax** below)	BREAKPOINT or BP	**Deletes breakpoints** as described in *Breakpoint Maintenance*.
	SPY	Deletes breakpoints *and* watchpoints: see also ***Delete Spy***.
	WATCHPOINT or WP	**Deletes watchpoints** as described in *Watchpoint Maintenance*.
	ENVIRONMENT	Deletes the specified debug environment: see also ***Delete Debug Environment***.
DISPLAY (**syntax** below)	BREAKPOINT or BP	**Displays breakpoints** as described in *Breakpoint Maintenance*.
	SPY	Displays breakpoints *and* watchpoints: see also ***Display Spy***.
	WATCHPOINT or WP	**Displays watchpoints** as described in *Watchpoint Maintenance*.
	CALL	Displays statistics on Natural objects invoked during the execution of an application: see also ***Display Called Objects***.
	EXEC	Displays statistics on executed statement lines of invoked Natural objects: see also ***Display Executed Statement Lines***.
	HEXADECIMAL	Displays the contents of variables in hexadecimal format.

Command	Subcommand(s)	Explanation
	NOCALL	Displays statistics on Natural objects that have not been invoked during the execution of an application: see also ***Display Non-Called Objects***.
	NOEXEC	Displays statistics on non-executed statement lines of invoked Natural objects: see also ***Display Non-Executed Statement Lines***.
	OBJECT	Displays statistics on the call frequency of objects: see also ***Display All Objects***.
	STATEMENT	Display statistics on executed and non-executed statement lines of invoked Natural objects: see ***Display All Statement Lines*** .
	VARIABLE	**Displays variables** for interrupted Natural objects as described in *Variable Maintenance*.
	XSTATISTICS	Displays a statistical summary of execution statistics: see also ***Display Statement Execution Statistics***.
EM		Invokes the **Debug Environment Maintenance** menu described in *Debug Environment Maintenance*.
ESCAPE	BOTTOM	Stops processing a loop and escapes to the first statement after the loop: see ESCAPE BOTTOM in *Execution Control Commands*.
	ROUTINE	Stops processing an interrupted Natural object and continues with another object, if available: see ESCAPE ROUTINE in *Execution Control Commands*.
EXIT		Leaves the current screen: see EXIT in *Execution Control Commands*.
FLIP		Switches between the display of the two PF-key lines (PF1 to PF12 and PF13 to PF24).
GO		Returns control to the execution of the interrupted Natural object: see GO in *Execution Control Commands*.
LAST		Displays the command entered last. The last three commands are stored and can be recalled.
LEFT		Shifts to the left side of a source code listing.
LIST (**syntax** below)		Displays the source code of an object.
	BREAK	Shows the object source with the current break. The relevant statement line is highlighted.
	LASTLINE	Shows the object source with the last line executed before the current break.
LOAD (**syntax** below)	ENVIRONMENT	Loads the debug environment specified: see ***Load Debug Environment***.
MENU		Invokes the **Debug Main Menu**.
MODIFY (**syntax** below)	BREAKPOINT or	**Modifies breakpoints** as described in *Breakpoint Maintenance*.

Command	Subcommand(s)	Explanation
	BP	
	SPY	Invokes the **Modify Breakpoint** or **Modify Watchpoint** screen: see also ***Modify Spy*** in *Spy Maintenance*.
	WATCHPOINT or WP	**Modifies watchpoints** as described in *Watchpoint Maintenance*.
	HEXADECIMAL	Modifies the contents of variables in hexadecimal format.
	VARIABLE	Invokes the **Display Variable** screen for modification as described in ***Modify Variable*** (using PF5).
NEXT		Executes the next command specified for a breakpoint or watchpoint.
OBJCHAIN		Displays executed objects at various program levels: see OBJCHAIN in *Navigation and Information Commands*.
ON or OFF		Switches test mode on or off. See also ***Switch Test Mode On and Off***.
PRINT (**syntax** below)	CALL	Prints statistics on Natural objects invoked during the execution of an application: see also ***Display Called Objects***.
	EXEC	Prints statistics on executed statement lines of invoked Natural objects: see also ***Display Executed Statement Lines***.
	NOCALL	Prints statistics on Natural objects that have not been invoked during the execution of an application: see also ***Display Non-Called Objects***.
	NOEXEC	Prints statistics on non-executed statement lines of invoked Natural objects: see also ***Display Non-Executed Statement Lines***.
	OBJECT	Prints statistics on the call frequency of objects: see also ***Display All Objects***.
	STATEMENT	Prints statistics on executed and non-executed statement lines of invoked Natural objects: see also ***Display All Statement Lines***.
	XSTATISTICS	Prints statistics on executed statement lines: see also ***Display Statement Execution Statistics***.
PROFILE		Displays the **User Profile** screen where you can modify the profile of the debugger as described in ***Navigation and Information Commands***.
RESET (**syntax** below)	ENVIRONMENT	Resets the current debug environment: see ***Reset Debug Environment***.
RIGHT		Shifts to the right side of a source code listing.
RUN		Switches off test mode and continues program execution.

Command	Subcommand(s)	Explanation
SAVE (**syntax** below)	ENVIRONMENT	Resets the current environment and saves the debug specifications. See also ***Save Debug Environment***.
SCAN	ABS	Only applies when using the function **List object source** (see *List Object Source*). Searches for a value in the source code of an object: see SCAN in *Navigation and Information Commands* and *Syntax Diagrams* below.
SCREEN		When entered upon interruption of an object, displays the current screen output of the interrupted Natural object. ENTER takes you back to debug mode.
SET (**syntax** below)	BREAKPOINT or BP	Invokes the **Set Breakpoint** screen described in *Breakpoint Maintenance*.
	CALL ON or CALL OFF	Activates or deactivates call statistics as described in *Call Statistics Maintenance*.
	OBJECT	Changes the default object specified for the debugger. See also SET OBJECT in *Navigation and Information Commands*.
	WATCHPOINT or WP	Invokes the **Set Watchpoint** screen described in ***Watchpoint Maintenance***.
	XSTATISTICS ON or XSTATISTICS COUNT or XSTATISTICS OFF	Activates (ON or COUNT) deactivates (OFF) the statement execution statistics as described in ***Set Statement Execution Statistics***.
SM		Invokes the **Spy Maintenance** menu described in *Spy Maintenance*.
STACK		Displays the contents of the entry at the top of the Natural stack: see STACK in *Navigation and Information Commands*.
STEP	[*n*]	Continues an interrupted Natural object for the number (*n*) of executable statements specified with the command. If you do not specify *n*, one executable statement is skipped by default. See also STEP in *Execution Control Commands*.

Command	Subcommand(s)	Explanation
	SKIPSUBLEVEL [*n*]	Continues step-mode processing of Natural objects without entering programs at sub-levels. You can specify a level number (*n*). See also STEP SKIPSUBLEVEL in *Execution Control Commands*.
STOP		Terminates both the debugger and any interrupted Natural object; the NEXT prompt appears.
SYSVARS		Displays the current values of a limited set of system variables (except the database-related system variables). See also ***Display System Variables***.
TEST ON or TEST OFF		Switches test mode on or off. See also ***Switch Test Mode On and Off***.
WM		Invokes the **Watchpoint Maintenance** menu described in *Watchpoint Maintenance*.

Syntax Diagrams

The syntax diagrams listed below refer to more complex command sequences.

For detailed explanations of the symbols used within the syntax descriptions, see the section *System Command Syntax* in the *System Commands* documentation.

For better readability, synonymous keywords are omitted from the syntax diagrams below. An underlined portion of a keyword represents an acceptable abbreviation.

Valid synonyms are:

Keyword	Synonym
BREAKPOINT	BP
DEACTIVATE	DA
WATCHPOINT	WP

- ACTIVATE
- DEACTIVATE
- DELETE
- DISPLAY
- LIST
- LOAD
- MODIFY
- PRINT
- RESET

- SAVE
- SET

ACTIVATE

```
          { SPY          [ { name   } ] }
          {                { number  }   }
ACTIVATE  { BREAKPOINT [ object ] [ line ]   }
          {                                  }
          { WATCHPOINT [      [ object ] variable ]  }
```

DEACTIVATE

```
            { SPY          [ { name   } ] }
            {                { number  }   }
DEACTIVATE  { BREAKPOINT [ object ] [ line ]   }
            {                                  }
            { WATCHPOINT [      [object] variable ]  }
```

DELETE

```
        { SPY          [ { name   } ] }
        {                { number  }   }
        { BREAKPOINT  [ object ] [ line ]   }
DELETE  { WATCHPOINT  [      [ object ] variable ]  }
        {                                  }
        { XSTATISTICS [      [ library ] object  ]  }
        { ENVIRONMENT [ name ]                 }
```

DISPLAY

```
        { SPY            [ { name                    } ] }
        {                  { number                  }   }
        { BREAKPOINT     [ object ] [ line ]              }
        { WATCHPOINT     [      [ object ] variable     ] }
        { CALL           [                              ] }
        { OBJECT         [                              ] }
DISPLAY { NOCALL         [                              ] }
        { XSTATISTICS    [      library [ object ]      ] }
        { STATEMENT      [                              ] }
        { EXEC           [                              ] }
        { NOEXEC         [                              ] }
        { VARIABLE       [      variable-name           ] }
        { HEXADECIMAL    [      [index-specification],... ] }
```

LIST

```
      { LASTLINE      }
LIST  { BREAK         }
      { object [ line ] }
```

LOAD

```
LOAD ENVIRONMENT [ name ]
```

MODIFY

```
        { SPY            [ { name                    } ] }
        {                  { number                  }   }
        { BREAKPOINT     [ object ] [ line ]              }
MODIFY  { WATCHPOINT     [      [ object ] variable     ] }
        { VARIABLE       [      variable [ = new value ] ] }
        { HEXADECIMAL    [                              ] }
```

PRINT

```
        { CALL                                  }
        { OBJECT                                }
        { NOCALL                                }
PRINT   { XSTATISTICS   [ library[object] ]     }
        { STATEMENT                             }
        { EXEC                                  }
        { NOEXEC                                }
```

RESET

```
RESET ENVIRONMENT[name]
```

SAVE

```
SAVE ENVIRONMENT[name]
```

SET

```
    { OBJECT          object                                  }
    { BREAKPOINT      object { line                         }  }
    {                        { label                        }  }
    { WATCHPOINT    [ [object] variable                     ]  }
SET { CALL          { OFF        }                            }
    {               { ON         }                            }
    { XSTATISTICS   { OFF                                     }
    {               { ON         [ library[object] ]          }
    {               { COUNT                                   }
```

18 Preparing Natural for Attached Debugging

- Introduction 134
- Prerequisites for Attached Debugging 134
- Example 135

Introduction

This document provides information on activating the debug attach server (DAS) to debug an external Natural application with NaturalONE.

An external application runs in a Natural environment but stores its sources in a NaturalONE project. The DAS is used to access a NaturalONE project.

For more information on using the DAS, refer to the *NaturalONE* documentation.

Prerequisites for Attached Debugging

The following prerequisites must be met to access the NaturalONE debugger from a z/OS Natural session:

- The Natural session runs in a z/OS environment.
- NaturalONE is installed.
- Natural Development Server is installed and the version installed must support attached debugging.
- Module `NATAD`*`vrs`* (or `NCIAD`*`vrs`* for a CICS session on z/OS) is generated from the Natural Development Server library and can be accessed for the Natural session.
- The profile parameter `DBGAT` is specified, see `DBGAT`.
- The profile parameter `RCA` is set to `NATATDBG`.
- The profile parameter `RCALIAS` is set to `(NATATDBG,NATAD`*`vrs`*`)`, for CICS on z/OS to `(NATATDBG,NCIAD`*`vrs`*`)`.
- The DAS is running and can be addressed through TCP/IP. The DAS is shipped with NaturalONE as `NATDAS.EXE` file.

For detailed information on the profile parameters mentioned above, refer to the *Parameter Reference* documentation.

Example

A Natural batch application is to be debugged with NaturalONE. The DAS server is available under the TCP/IP name DASSERV and listens to port 50882. The NaturalONE debugger has identified itself to the DAS with client ID FRED. The attached debug interface resides in DSN NDV*vrs*.LOAD:

```
//NATBAT  EXEC PGM=NATBATvr
//STEPLIB DD   DISP=SHR,DSN=NATvrs.LOAD
//        DD   DISP=SHR,DSN=NDVvrs.LOAD
//CMPRMIN DD   *
RCA=NATATDBG,RCALIAS=(NATATDBG,NATADvrs)
DBGAT=(ACTIVE=ON,HOST=DASSERV,PORT=50882,CLID=FRED)
/*
```