Natural

Debugger

Version 9.1.1

April 2019

ADABAS & NATURAL

Dieses Dokument gilt für Natural ab Version 9.1.1.

Hierin enthaltene Beschreibungen unterliegen Änderungen und Ergänzungen, die in nachfolgenden Release Notes oder Neuausgaben bekanntgegeben werden.

Dokument-ID: NAT-DEBUG-911-20211014DE

Inhaltsverzeichnis

Vorwort

Mit dem Debugger können Sie Programmfehler erkennen, finden und korrigieren, die Programmausführung testen oder optimieren und einen Natural-Fehler analysieren, der die Programmausführung unterbricht.

Debugger-Tutorial	Erste Schritte mit dem Debugger.
Debugger-Konzept	Grundkonzeption des Debuggers.
Debugger starten	Betriebsanforderungen und Anweisungen zum Aufrufen des Debuggers.
Test-Modus ein- und ausschalten	Einstellung des Test-Modus zum Aktivieren und Deaktivieren des Debugging.
Debug Environment Maintenance	Verwaltung einer Debug-Umgebung: Speichern und Benutzen einer vordefinierten Debug-Umgebung.
Spy Maintenance	Verwaltung der Debug-Einträge: Setzen, Ändern, Löschen und Aktivieren von Haltepunkten (sowohl Breakpoints als auch Watchpoints).
Breakpoint Maintenance	Verwaltung von Breakpoints: Setzen, Ändern, Löschen und Aktivieren von Breakpoints. Erklärung der Breakpoint-Bildschirminhalte.
Watchpoint Maintenance	Verwaltung von Watchpoints: Setzen, Ändern, Löschen und Aktivieren von Watchpoints. Erklärung der Watchpoint-Bildschirminhalte.
Call Statistics Maintenance	Abrufen von Statistiken über aufgerufene Objekte.
Statement Execution Statistics Maintenance	Abrufen von Statistiken über ausgeführte Statement-Zeilen.
Variable Maintenance	Anzeigen und Ändern von Variablen.
List Object Source	Anzeigen eines Objekt-Sourcecode.
Fehlerbehandlung	Behandlung von Fehlern, die während der Ausführung der Anwendung oder des Debuggers auftreten können.
Kommandos zur Ausführungssteuerung	Debugger-Kommandos für die Programmablaufsteuerung.
Kommandos zum Navigieren und Anzeigen von Informationen	Debugger-Kommandos für Bildschirmnavigation, Objektinformationen und Debugger-Profileinstellungen.
Kommandoübersicht und -syntax	Alle Debugger-Kommandos und die entsprechende Kommando-Syntax.
Natural für Attached-Debugging vorbereiten	Verwendung eines Debug-Attach-Servers, der unter NaturalONE läuft.

Anmerkungen:

1. Die Sprache der Debugger-Benutzungsoberfläche ist Englisch. Daher sind im folgenden Dokument alle zur Orientierung erforderlichen englischen Literale beibehalten und ggf. nur in Klammern übersetzt worden.
2. Terminologie: **„Breakpoints“** und **„Watchpoints“** sind Debug-Haltepunkte. **"Spies"** ist der Sammelbegriff für diese Haltepunkte. Die Funktion „Spy“ dient zur Verwaltung der verschiedenen Debug-Einträge.
3. Die in diesem Dokument verwendete Notation *vrs* bzw. *vr* steht als Platzhalter für die betreffende Produktversion (siehe auch Version im *Glossar*).

1 Über diese Dokumentation

Dokumentationskonventionen

Konvention	Beschreibung
Fettschrift	>Kennzeichnet Elemente auf einem Bildschirm.
`Nichtproportionale Schrift`	Kennzeichnet Namen und Orte von Diensten im Format *`Ordner.Unterordner.Dienst`*, Programmierschnittstellen (APIs), Namen von Klassen, Methoden und Properties in Java.
`Kursivschrift`	Kennzeichnet: Variablen, für die Sie situations- oder umgebungsspezifische Werte angeben müssen. Neue Begriffe, wenn sie erstmals im Text auftreten. Verweise auf andere Dokumentationsquellen.
`Nichtproportionale Schrift`	Kennzeichnet: Text, den Sie eingeben müssen. Meldungen, die vom System angezeigt werden. Programmcode.
{ }	Zeigt eine Reihe von Auswahlmöglichkeiten an, von denen Sie eine auswählen müssen. Geben Sie nur die innerhalb der geschweiften Klammern vorhandenen Informationen ein. Geben Sie nicht die Klammersymbole { } ein.
|	Trennt zwei sich gegenseitig ausschließende Auswahlmöglichkeiten in einer Syntaxzeile voneinander ab. Geben Sie eine der Auswahlmöglichkeiten ein. Geben Sie nicht das Symbol | ein.
[ ]	Zeigt eine oder mehrere Optionen an. Geben Sie nur die innerhalb der eckigen Klammern vorhandenen Informationen ein. Geben Sie nicht die Klammersymbole [ ] ein.
...	Zeigt an, dass Sie mehrere Auswahlmöglichkeiten desselben Typs eingeben können. Geben Sie nur die Informationen ein. Geben Sie nicht die drei Auslassungspunkte (...) ein.

Online-Informationen und Support

Dokumentationswebsite der Software AG

Sie finden die Dokumentation zu den Produkten der Software AG auf der Dokumentationswebsite der Software AG unter **https://documentation.softwareag.com**.

Empower, die Produktsupportwebsite der Software AG

Falls Sie noch kein Benutzerkonto für Empower haben, können Sie eine E-Mail an empower@softwareag.com senden. Geben Sie darin Ihren Namen, den Namen Ihrer Firma und deren E-Mail-Adresse an und beantragen Sie die Einrichtung eines Benutzerkontos.

Wenn Sie ein Benutzerkonto erhalten haben, können Sie den eService-Bereich von Empower unter **https://empower.softwareag.com/** aufrufen und dort Support-Fälle online öffnen.

Informationen zu Software AG-Produkten finden Sie auf der Empower-Produktsupportwebsite unter **https://empower.softwareag.com**.

Unter **Products & Documentation** können Sie Anträge bezüglich Produktmerkmalen und Produktverbesserungen einreichen, Informationen über die Verfügbarkeit von Produkten abrufen und Produkte herunterladen.

Im **Knowledge Center** finden Sie Informationen zu Programmkorrekturen (Fixes) und frühzeitige Warnungen, technische Abhandlungen (Papers) und Artikel aus der Wissensdatenbank.

Wenn Sie noch Fragen haben und telefonisch mit uns Kontakt aufnehmen möchten, können Sie im Kontaktverzeichnis des Globalen Supports unter **https://empower.softwareag.com/public_directory.aspx** eine der dort für Ihr Land angegebenen örtlichen oder gebührenfreien Telefonnummern auswählen.

Software AG TECHcommunity

Auf der Website der Software AG TECHcommunity unter **http://techcommunity.softwareag.com** finden Sie Dokumentationen und andere technische Informationen.

- Sie können auf Produktdokumentationen zugreifen, wenn Sie die erforderlichen Authentifizierungsdaten für die TECHcommunity haben. Andernfalls müssen Sie sich registrieren und "Documentation" als Interessengebiet angeben.
- Sie erhalten Zugang zu Artikeln, Code-Beispielen, Demos und Lernprogrammen.
- Sie können an von Software AG-Experten moderierten Online-Diskussionsforen teilnehmen, um Fragen zu stellen, über bewährte Methoden und Prozesse (Best Practices) zu diskutieren und zu erfahren, wie andere Kunden die Technologien der Software AG nutzen.
- Sie können Links auf externe Websites benutzen, die sich mit offenen Standards und Web-Technologien befassen.

Datenschutz

Die Produkte der Software AG stellen Funktionen zur Verarbeitung von personenbezogenen Daten gemäß der Datenschutz-Grundverordnung (DSGVO) der Europäischen Union zur Verfügung. Gegebenenfalls sind in der betreffenden Systemverwaltungsdokumentation entsprechende Schritte dokumentiert.

2 Debugger-Tutorial

In diesem Tutorial werden die grundlegenden Funktionen des Debuggers vorgestellt und verschiedene Debugging-Methoden besprochen. Das Tutorial führt Sie durch ein einfaches Szenario, das demonstriert, wie der Debugger verwendet werden kann, um Laufzeitfehler zu analysieren und die Programmausführung zu kontrollieren.

Es ist wichtig, dass Sie die Sitzungen 1 bis 5 nacheinander durcharbeiten.

Anmerkungen:

1. Der Einfachheit halber zitiert das Tutorial hauptsächlich Direktkommandos zur Demonstration der Debugger-Funktionen und nicht die alternativen Menü-Funktionen.
2. Eine vollständige Beschreibung aller Debugger-Funktionen, die in diesem Tutorial erwähnt werden, finden Sie in den anschließenden Kapiteln der *Debugger*-Dokumentation.

Voraussetzungen

- Sie sollten mit der Programmierung in Natural vertraut sein
- Bevor Sie mit Sitzung 1 beginnen, müssen Sie alle Beispielprogramme (`DEBUG1P` und `DEBUG2P`) und Subprogramme (`DEBUG1N`, `DEBUG2N`, `DEBUG3N` und `DEBUG4N`) erstellen, die im Abschnitt ***Beispiel-Sourcecode*** weiter unten in diesem Tutorial vorhanden sind. Speichern und katalogisieren Sie diese Objekte mit dem Systemkommando `STOW`.

Grundlagen des Debugging

Mit dem Debugger können Sie den Ausführungsfluss eines Natural-Objekts bei einem bestimmten Debug-Event (Ereignis) unterbrechen und Informationen zum aktuellen Status des unterbrochenen Objekts, z.B. das nächste auszuführende Statement, den Wert einer Variablen oder die Hierarchie (Programmebenen) aufgerufener Objekte erhalten.

Grundsätzlich müssen Sie die folgenden zwei Hauptschritte ausführen, um die Kontrolle an den Debugger für die Programmunterbrechung zu übergeben:

1. Aktivieren Sie den Debugger mit dem Systemkommando `TEST ON`.

   Dadurch kann der Debugger die Steuerung für jedes auszuführende Statement erhalten.
2. Setzen Sie einen oder mehrere Debug-Einträge (**Breakpoints** und **Watchpoints**) für die Natural-Objekte, die ausgeführt werden sollen.

   Dadurch kann der Debugger entscheiden, wann er die Kontrolle vom Natural-Laufzeitsystem übernimmt und die Programmausführung stoppt.

Ein Natural-Fehler führt immer zur Unterbrechung des Programms.

Es ist dann keine Debug-Eingabe erforderlich, der Debugger greift automatisch ein.

Im Folgenden finden Sie eine Übersicht über alle möglichen Programmunterbrechungen:

Programmunterbrechung	Erklärung
Breakpoint	Haltepunkt: Bewirkt eine Programmunterbrechung für eine Statement-Zeile in einem Natural-Objekt. Der Debugger unterbricht das Programm, sobald die Statement-Zeile, für die ein Breakpoint gesetzt ist, ausgeführt werden soll, d.h., *bevor* das in dieser Zeile enthaltene Statement verarbeitet wird.
Watchpoint	Haltepunkt: Bewirkt eine Programmunterbrechung für eine Variable in einem Natural-Objekt. Der Debugger unterbricht das Programm, sobald sich der Inhalt der Variablen, für die ein Watchpoint gesetzt ist, geändert hat, d.h., *nachdem* das Statement, das diese Variable referenziert, verarbeitet wird.
Step-Modus	Einzelschritt-Modus: Das Objekt wird schrittweise während der Programmausführung durchlaufen. Der Step-Modus wird durch ein Debugger-Kommando eingeleitet. Dazu muss der Debugger zuvor wegen eines Breakpoint- oder Watchpoint-Haltepunkts die Kontrolle erhalten haben. Im Step-Modus unterbricht der Debugger die Programmausführung, *bevor* ein ausführbares Statement, das in diesem Objekt enthalten ist, verarbeitet wird.
Natural-Fehler	Bewirkt eine automatische Programmunterbrechung.

Sitzung 1 - Analyse eines Natural-Fehlers

Diese Sitzung beschreibt Untersuchungsmethoden für einen Natural-Fehler, der während der Programmausführung auftritt.

Um einen Natural-Fehler zu simulieren:

- Führen Sie an der Eingabeaufforderung `NEXT` das Beispielprogramm `DEBUG1P` aus.

  Es erscheint folgende Natural-Meldung: `DEBUG1N 0180 NAT0954 Abnormal termination S0C7 during program execution.` (Abnormale Beendigung S0C7 während der Programmausführung).

  Die Meldung verweist auf die Zeile 180 im Subprogramm `DEBUG1N`: `BONUS := SALARY * PERCENT / 100`. Dies zeigt an, dass von einer oder mehreren der referenzierten Variablen falsche Werte zurückgegeben werden. Dies ist jedoch noch kein eindeutiger Hinweis darauf,

was das Problem tatsächlich verursacht, und es könnte schwierig sein, die Ursache zu ermitteln, wenn die Variablenwerte aus einer Datenbank abgerufen wurden, so wie es typisch ist für Mitarbeiter-Datensätze (Employees Records).

Um den Debugger für die weitere Untersuchung des Problems zu aktivieren:

1 Geben Sie an der Eingabeaufforderung `NEXT` Folgendes ein:

```
TEST ON
```

Die Meldung `Test mode started.` (Test-Modus gestartet) zeigt an, dass der Debugger aktiviert ist.

Anmerkung: `TEST ON` bleibt aktiv für die Dauer der aktuellen Sitzung oder solange, bis Sie `TEST OFF` eingeben, um den Debugger zu deaktivieren.

2 Führen Sie an der Eingabeaufforderung `NEXT` wieder das Beispielprogramm `DEBUG1P` aus.

Das Fenster **Debug Break** erscheint (Beispiel):

```
+------------------- 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   |
+---------------------------------------------------+
```

Weil ein Natural-Fehler auftritt, greift der Debugger automatisch ein und zeigt das Fenster **Debug Break**.

Am oberen Rand des Fensters werden zusätzliche Informationen dazu angezeigt, wo der Fehler auftritt: Das Modul (`NATARI2`) im Natural-Nucleus (nützlich für den Software AG Technical Support), der Objekttyp (`subprogram`), die Library (`DEBUG`) und die Datenbankkennung und Dateinummer (`10,32`) der Systemdatei.

Außerdem sind im Fenster **Debug Break** Debugger-Funktionen verfügbar, die Sie benutzen können, um z.B. die Programmausführung fortzusetzen (**Go** oder **Run**), das Debugger-

Hauptmenü (**Debug Main Menu**) aufzurufen oder den Step-Modus zu aktivieren. Sie können dazu entweder den entsprechenden Kennbuchstaben im Feld **Code** eingeben und Enter drücken oder die entsprechende PF-Taste drücken.

Um die fehlerhafte Statement-Zeile zu untersuchen:

- Ersetzen Sie im Feld **Code** den Standardeintrag `G` durch `L`, um die Funktion **List break** auszuführen.

  Der Sourcecode des Beispielprogramms `DEBUG1N` wird angezeigt:

```
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
```

  Die Meldung `last line` (letzte Zeile) zeigt an, dass das in Zeile 170 enthaltene Statement das letzte Statement ist, das erfolgreich ausgeführt wurde.

  Das Statement in Zeile 180, welches das Problem verursacht, ist hervorgehoben und mit der Anmerkung `* NAT0954 *` markiert.

  Dies zeigt an, dass der Fehler durch den Inhalt der Variablen `SALARY` (Gehalt) oder `PERCENT` (Prozent) verursacht wird. Höchstwahrscheinlich handelt es sich um `SALARY`, da `PERCENT` korrekt initialisiert ist.

Um den Inhalt der Variablen SALARY zu prüfen:

1 Geben Sie Folgendes in der Kommandozeile ein:

```
DIS VAR SALARY
```

Der Bildschirm **Display Variable** wird für die Variable SALARY angezeigt (Beispiel):

```
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
```

Die Meldung `Variable contains invalid data.` (Variable enthält ungültige Daten) zeigt an, dass der Inhalt der Variablen, die scheinbar leer ist, nicht dem Format der Variablen entspricht. Dies wird deutlich, wenn Sie, wie im folgenden Schritt beschrieben, die hexadezimale Darstellung des Variableninhalts betrachten.

2 Drücken Sie PF11 (Hex), um den hexadezimalen Inhalt der Variablen anzuzeigen.

Der Bildschirm sieht nun ähnlich aus wie im folgenden Beispiel:

```
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
```

Der hexadezimale Wert zeigt, dass die Variable nicht in gepacktem numerischem Format vorliegt, was folglich bei der Programmausführung zu einem Rechenfehler führt. Das Programm `DEBUG1P` übergibt dem Subprogramm `DEBUG1N` einen inkorrekten Wert für die Variable `SALARY`.

Tipp: Sie können `PF10` (Alpha) drücken, um wieder zur alphanumerischen Darstellung zurück zu wechseln.

3 Geben Sie in der Kommandozeile Folgendes ein:

```
GO
```

Das Kommando `GO` bewirkt, dass die Kontrolle vom Debugger an das Natural-Laufzeitsystem zurückgegeben und dass die Ausführung des Programms bis zum Programmende oder bis zum nächsten Debug-Ereignis (Event) fortgesetzt wird. Im vorliegenden Fall gibt es kein zusätzliches Debug-Ereignis. Die Eingabeaufforderung `NEXT` erscheint mit der bekannten Natural-Fehlermeldung (**s.o.**).

Um die Variable `SALARY` im Objekt-Sourcecode zu korrigieren:

1 Öffnen Sie das Programm `DEBUG1P` mit dem Programm-Editor und entfernen Sie das das für `SALARY := 99000` eingegebene Kommentarzeichen (*).

2 Benutzen Sie das Systemkommando `STOW`, um das geänderte Programm zu speichern und zu katalogisieren.

3 Führen Sie das Programm `DEBUG1P` aus.

Der Debugger unterbricht das Programm nicht, obwohl immer noch `TEST ON` gesetzt ist. Das Programm wird erfolgreich ausgeführt und gibt einen Bericht aus:

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

EMPLOYEE RECEIVES:   100800.00
    PLUS BONUS OF:     3465.00




NEXT                                                                 LIB=DEBUG
```

Sitzung 2 - Einen Breakpoint benutzen

Sie können die Programmausführung an einer bestimmten Statement-Zeile unterbrechen, indem Sie einen Breakpoint (Haltepunkt) für diese Zeile setzen.

Um für eine Zeile im Subprogramm `DEBUG1N` einen Breakpoint zu setzen:

1 Geben Sie Folgendes an der Eingabeaufforderung `NEXT` ein:

```
TEST SET BP DEBUG1N 170
```

Die Meldung `Breakpoint DEBUG1N0170 set at line 170 of object DEBUG1N.` erscheint. Sie bestätigt, dass für die Statement-Zeile 170 im Subprogramm `DEBUG1N` ein Breakpoint mit dem Namen `DEBUG1N0170` gesetzt ist.

Anmerkungen:

1. Ein Breakpoint kann nur für ein ausführbares Statement gesetzt werden. Falls Sie versuchen, für ein nicht ausführbares Statement einen Breakpoint zu setzen, erscheint eine entsprechende Fehlermeldung.
2. Normalerweise ist ein Breakpoint nur während der aktuellen Natural-Sitzung gültig. Falls erforderlich, können Sie einen Breakpoint für zukünftige Sitzungen speichern. Weitere Informationen siehe ***Breakpoints und Watchpoints speichern*** in *Weitere Hinweise zur Benutzung des Debuggers*.

2 Führen Sie das Programm `DEBUG1P` aus.

Der Debugger unterbricht nun die Programmausführung an der Statement-Zeile, bei der der neue Breakpoint gesetzt ist. Das Fenster **Debug Break** erscheint:

```
+------------------- 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    |
+----------------------------------------------------+
```

Das Fenster zeigt den Namen des Breakpoints, die entsprechende Statement-Zeile, das Objekt und die Library, in der das Objekt enthalten ist. Es zeigt auch die Bearbeitungsebene (Level) des Subprogramms `DEBUG1N`.

Um das im Fenster Debug Break gezeigte Statement zu betrachten:

- Führen Sie die Funktion **List break** aus.

  Der Sourcecode des Subprogramms `DEBUG1N` wird im Bildschirm **List Object Source** angezeigt:

```
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
```

Die im Fenster **Debug Break** angezeigte Statement-Zeile 170 ist hervorgehoben. In der Spalte **Message** wird der Name des für dieses Statement gesetzten Breakpoints (`DEBUG1N0170`) sowie die zuletzt ausgeführte Statement-Zeile angezeigt (die Zeile 160 ist als `last line` kommentiert). Zur Erinnerung: Ein Breakpoint unterbricht die Programmausführung *vor* dem Statement, für das der gesetzte Breakpoint verarbeitet wird.

Es gibt mehrere Direktkommandos, die Sie auf dem Bildschirm **List Object Source** eingeben können, um weitere Informationen über das aktuelle Objekt zu erhalten. Sie können sich z. B. alle Variablen wie im folgenden Schritt beschrieben anzeigen lassen.

Um eine Liste der in `DEBUG1N` enthaltenen Variablen zu erhalten:

- Geben Sie in der Kommandozeile Folgendes ein:

```
DIS VAR
```

Der Bildschirm **Display Variables** erscheint (Beipiel):

```
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
```

Es werden alle in DEBUG1N definierten Variablen aufgelistet. Die Bemerkung invalid data (ungltige Daten) bei der Variable BONUS können Sie unbeachtet lassen. Im vorliegenden Fall ist es nicht von Bedeutung, ob die Variable BONUS korrekt initialisiert wird, weil sie nur als Ziel-Operand benutzt wird.

Um jedoch ein anderes Debugger-Kommando auszuführen, ändern Sie den Inhalt von BONUS im folgenden Schritt.

Um den Inhalt von BONUS zu prüfen und zu ändern:

1 Geben Sie in der Spalte **Co** neben BONUS folgendes Zeilenkommando ein:

```
MO
```

Oder:

Geben Sie in der Kommandozeile Folgendes ein:

```
MOD VAR BONUS
```

Der Bildschirm **Modify Variable** wird angezeigt (Beispiel):

```
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 Sie können die hexadezimale Anzeige benutzen, um sicherzustellen, dass die Variable nicht in Format P (gepackt numerisch) ist. Drücken Sie dazu PF10 (Alpha), um auf die alphanumerische Anzeige zurückzuschalten.

3 Geben Sie im Feld **Contents** einen Wert im Format P (numerisch gepakt) ein, z.B. `12345.00` und drücken Sie PF5 (Save).

Der Bildschirm sieht nun folgendermaßen aus (Beispiel):

```
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
```

Die Änderung von **Contents** wird durch die Meldung `Variable BONUS modified.` (Variable `BONUS` geändert.) angezeigt.

4 Drücken Sie PF9 (Li Br) oder PF3 (Exit).

Der Bildschirm **List Object Source** erscheint.

5 Geben Sie in der Kommandozeile Folgendes ein:

```
GO
```

Der Debugger gibt die Kontrolle an das Natural-Laufzeitsystem zurück, welches die Ausführung des Programms `DEBUG1P` beendet, weil kein weiteres Debug-Ereignis (Event) auftritt. Der vom Programm erstellte Bericht wird ausgegeben:

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

EMPLOYEE RECEIVES:   100800.00
    PLUS BONUS OF:     3465.00




NEXT                                                                LIB=DEBUG
```

6 Bevor Sie fortfahren und mit der nächsten Sitzung beginnen, löschen Sie bitte alle aktuellen Breakpoints, indem Sie Folgendes an der Eingabeaufforderung `NEXT` eingeben:

```
TEST DEL BP * *
```

Es erscheint eine Meldung, die bestätigt, dass alle Breakpoints (im konkreten Fall nur ein Breakpoint) gelöscht sind.

Sitzung 3 - Einen Watchpoint benutzen

Das Programm `DEBUG1P` und das Subprogramm `DEBUG1N` führen eine Berechnung für die Bonus- und Gehaltszahlung eines einzelnen Mitarbeiters durch. Wenn mehrere Mitarbeiterdatensätze verarbeitet würden, würden Sie wahrscheinlich prüfen, ob die Variable `BONUS` nun korrekt aktualisiert wird. Dies geschieht durch Setzen eines Watchpoints für diese Variable. Ein Watchpoint ermöglicht es dem Debugger, die Programmausführung zu unterbrechen, wenn sich der Inhalt der angegebenen Variablen ändert.

Um einen Watchpoint für die Variable `BONUS` zu setzen:

1 Geben Sie Folgendes an der Eingabeaufforderung `NEXT` ein:

```
TEST SET WP DEBUG1N BONUS
```

Die Meldung `Watchpoint BONUS set for variable EMPLOYEE.BONUS.` erscheint. Sie bestätigt, dass ein Watchpoint für die Variable `BONUS` in Beispiel-Subprogramm `DEBUG1N` gesetzt ist.

Anmerkungen:

1. Wenn Sie in der Kommandozeile eines Debugger-Bildschirms ein Debugger-Direktkommando eingeben, müssen Sie das Schlüsselwort `TEST` weglassen. Beispiel: Anstelle von `TEST SET WP DEBUG1N BONUS` geben Sie dann nur `SET WP DEBUG1N BONUS` ein.
2. Normalerweise ist ein Watchpoint nur während der aktuellen Natural-Sitzung gültig. Falls erforderlich, können Sie einen Watchpoint für zukünftige Sitzungen speichern. Weitere Informationen siehe ***Breakpoints und Watchpoints speichern*** in *Weitere Hinweise zur Benutzung des Debuggers*.

2 Führen Sie das Programm `DEBUG1P` an der Eingabeaufforderung `NEXT` aus.

Der Debugger unterbricht nun die Programmausführung an dem neuen Watchpoint. Das Fenster **Debug Break** erscheint:

```
+------------------- 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    |
+----------------------------------------------------+
```

In dem Fenster wird angezeigt, dass ein Watchpoint in Zeile 180 festgestellt wurde. Diese Zeile enthält das Statement, das die Variable `BONUS` verarbeitet.

Der Debugger unterbrach die Programmausführung, *nachdem* das Statement für `BONUS` verarbeitet war. Erst dann konnte der Debugger erkennen, dass sich der Inhalt der Variablen geändert hat.

3 Führen Sie die Funktion **List break** aus.

Der Bildschirm **List Object Source** erscheint (Beispiel):

```
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
```

Das Statement, welches die Variable BONUS referenziert, ist hervorgehoben. Die Meldung in der Spalte **Message** zeigt den Namen des für die Variable gesetzten Watchpoint.

Um auf Änderungen in BONUS zu prüfen:

1 Geben Sie in der Kommandozeile Folgendes ein:

```
DIS VAR BONUS
```

Der Bildschirm **Display Variable** erscheint. Im Feld **Contents** (Inhalt) wird der Wert 3465.00 angezeigt. Das zeigt, dass sich der Inhalt der Variablen BONUS geändert hat.

2 Drücken Sie PF3 (Exit), um zum Bildschirm **List Object Source** zurückzukehren.

Um auf Änderungen in SALARY zu prüfen:

1 Um den Inhalt der Variablen SALARY in einem späteren Schritt zu prüfen, setzen Sie einen Breakpoint für SALARY. Dazu geben Sie in der Spalte **Co** vor der Zeile 200 Folgendes ein:

```
SE
```

Vom Bildschirm **List Object Source** aus ist die Eingabe eines Zeilenkommandos wie SE eine bequeme Alternative zur Verwendung des Direktkommandos SET BP.

In der Spalte **Message** wird angezeigt, dass ein Breakpoint (BP) für die Zeile 200 gesetzt ist:

```
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 Geben Sie in der Kommandozeile Folgendes ein:

```
GO
```

Das Fenster **Debug Break** erscheint:

```
+------------------- 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 Führen Sie die Funktion **List break** aus.

Der Bildschirm **List Object Source** sieht jetzt wie im folgenden Beispiel aus:

```
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
```

Da es sich hier um einen Breakpoint handelt, ist das Statement, das die Variable SALARY referenziert (und ändert), noch nicht ausgeführt worden. Folglich hat sich der Inhalt der Variablen nicht geändert.

4 Geben Sie DIS VAR SALARY in der Kommandozeile ein, um zu überprüfen, dass der er Inhalt der Variablen unverändert ist.

Der Bildschirm **Display Variable** belegt, dass die Variable SALARY immer noch den Wert 99000 enthält. Dies ist der Anfangswert, der ihr im Programm DEBUG1P zugewiesen wurde.

5 Springen Sie nun zum nächsten Statement, um die Änderung des Variableninhalts zu betrachten. Wählen Sie eine der folgenden Methoden:

Geben Sie in der Kommandozeile Folgendes ein:

```
STEP
```

Oder:

Drücken Sie PF2 (Step).

Der neue Bildschirm sieht ähnlich wie im folgenden Beispiel aus:

```
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
```

Sie haben eine Zeile übersprungen und das nächste ausführbare Statement in Zeile 200 ausgeführt, welches die Variable SALARY ändert. In der Spalte **Message** wird angezeigt, dass Step-

Modus gesetzt ist. Im Step-Modus setzt der Debugger die Programmausführung am nächsten ausführbaren Statement fort.

6 Geben Sie `DIS VAR SALARY` in der Kommandozeile ein, um den Variableninhalt zu prüfen.

Der Bildschirm **Display Variable** erscheint und zeigt im Feld **Contents** einen Wert von `100800.00` an. Das belegt, dass sich der Inhalt der Variablen `SALARY` geändert hat.

7 Geben Sie in der Kommandozeile Folgendes ein:

```
GO
```

Der Debugger gibt die Kontrolle an das Natural-Laufzeitsystem zurück, welches die Ausführung des Programms `DEBUG1P` beendet, weil kein weiteres Debug-Ereignis (Event) auftritt. Der vom Programm erstellte Bericht wird ausgegeben.

Sitzung 4 - Verfolgen des logischen Ablaufs von Programmen

Diese Sitzung beschreibt Debugging-Methoden, die Sie anwenden können, um eine komplexe Natural-Anwendung mit zahlreichen Objekten besser zu verstehen, zu überblicken und zu steuern.

Die Sitzung beginnt mit einer Anleitung zur Analyse des logischen Ablaufs einer Anwendung auf Statement-Ebene. Anschließend wird demonstriert, wie Breakpoints verwendet werden können, um die Reihenfolge der Programmausführung herauszufinden.

Die Anweisungen in dieser Sitzung basieren auf einer einfachen (aber zur Demonstration ausreichenden) Beispielanwendung. Diese besteht aus einem Programm (`DEBUG2P`) und drei Subprogrammen (`DEBUG2N`, `DEBUG3N` und `DEBUG4N`).

Um einen Breakpoint am Programmanfang oder -ende zu setzen:

1 Setzen Sie einen Breakpoint für `DEBUG2P`, indem Sie an der Eingabeaufforderung `NEXT` Folgendes eingeben:

```
TEST SET BP DEBUG2P BEG
```

Die Meldung `Breakpoint DEBUG2P-BEG set at line BEG of object DEBUG2P.` bestätigt, dass ein Breakpoint in `DEBUG2N` gesetzt ist.

Die Verwendung des Schlüsselworts `BEG` anstelle einer bestimmten Zeilennummer bewirkt, dass der Breakpoint am Anfang des Programms, d. h. für das erste auszuführende Statement, gesetzt wird. Dies kann sogar das `DEFINE DATA`-Statement sein, wenn z. B. eine `INIT`-Klausel verwendet wird, die beim Katalogisieren des Programms ein ausführbares Statement erzeugt.

Tipp: Sie können auch das Schlüsselwort `END` angeben, um einen Breakpoint für das letzte auszuführende Statement zu setzen. Dies kann das `END`-Statement sein, aber auch das `FETCH`- oder `CALLNAT`-Statement.

2 Führen Sie das Programm `DEBUG2P` aus.

Das Fenster **Debug Break** erscheint:

```
+------------------- 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   |
+---------------------------------------------------+
```

Der Debugger greift nun am ersten für das Programm gesetzten Breakpoint ein.

3 Führen Sie die Funktion **List break** aus, um den Sourcecode zu prüfen und zu sehen, dass der Debugger jetzt am ersten ausführbaren Statement `NAME := 'MEIER'` eingreift.

Um eine Anwendung schrittweise auszuführen:

1 Setzen Sie im Bildschirm **List Object Source** den Step-Mode, indem Sie entweder PF2 (Step) drücken oder `STEP` in der Kommandozeile eingeben.

Das zuletzt ausgeführte Statement ist mit `last line` (letzte Zeile) kommentiert. Das nächste auszuführende Statement ist hervorgehoben und mit `step mode` kommentiert.

Tipp: Wenn Sie nicht möchten, dass der Debugger bei jedem einzelnen Statement pausiert, sondern eine Anwendung schneller durchläuft, geben Sie im `STEP`-Kommando die Anzahl der Statements an, die Sie überspringen möchten, z.B.: `STEP 2` oder `STEP 10`.

2 Drücken Sie PF2 (Step) mehrmals, bis das `CALLNAT`-Statement mit `step mode` kommentiert ist.

3 Fahren Sie mit PF2 (Step) und führen Sie das `CALLNAT`-Statement aus.

Das aufgerufene Subprogramm DEBUG2N wird angezeigt, wobei das nächste auszuführende Statement hervorgehoben ist:

```
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
```

Alternativ könnten Sie das CALLNAT-Statement überspringen, indem Sie STEP SKIP in der Kommandozeile eingeben.

Sie würden dann nur die Statements in dem aufrufenden Programm DEBUG2 schrittweise ausführen, jedoch nicht die Statements in einem aufgerufenen Programm.

Um die Ebenen zu betrachten, auf denen die Objekte ausgeführt werden:

1 Geben Sie im Bildschirm **List Object Source** für DEBUG2N Folgendes in der Kommandozeile ein:

```
OBJCHAIN
```

Der Bildschirm **Break Information** erscheint (Beispiel):

```
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).
```

Zusätzlich zu den bereits bekannten Objektinformationen zeigt dieser Bildschirm an, ob das Programm einen globalen Datenbereich (GDA, Global Data Area) referenziert.

2 Drücken Sie ENTER, um eine Seite weiter nach unten zu blättern.

Der Bildschirm sieht jetzt aus wie im folgenden Beispiel:

```
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
```

Dieser Bildschirm zeigt die Ebenen (Level) an, auf denen die Objekte ausgeführt werden: Subprogramm `DEBUG2N` wird auf Ebene 2 ausgeführt und das Programm `DEBUG2P` (welches das Subprogramm aufruft) wird auf der übergeordneten Ebene 1 ausgeführt.

3 Drücken Sie ENTER.

Der Bildschirm **List Object Source** erscheint.

4 Geben Sie in der Kommandozeile Folgendes ein:

```
GO
```

Der Debugger gibt die Kontrolle zurück an das Natural-Laufzeitsystem. Dieses beendet die Ausführung von `DEBUG2P`, weil kein weiteres Debug-Ereignis auftritt. Der vom Programm erzeugte Bericht wird ausgegeben:

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

EMPLOYEE RECEIVES:    99300.00
    PLUS BONUS OF:     3565.00




NEXT                                                                LIB=DEBUG
```

5 Löschen Sie alle zurzeit gesetzten Breakpoints, indem Sie Folgendes an der Eingabeaufforderung `NEXT` eingeben:

```
TEST DEL BP * *
```

Es erscheint eine Meldung, die bestätigt, dass alle Breakpoints gelöscht sind.

Um Breakpoints zur Verfolgung der Programmausführung zu setzen:

1 Geben Sie Folgendes an der Eingabeaufforderung `NEXT` ein:

```
TEST SET BP ALL BEG
```

Die Meldung `Breakpoint ALL-BEG set at line BEG of object ALL.` erscheint.

Sie zeigt an, dass Sie einen Breakpoint für das erste ausführbare Statement jedes auszuführenden Objekts gesetzt haben.

2 Führen Sie das Programm `DEBUG2P` aus.

Es erscheint ein **Debug Break**-Fenster für `DEBUG2P`.

3 Führen Sie die Funktion **Go** wiederholt aus.

Jedes Mal, wenn Sie die Funktion **Go** ausführen, wird das nächste aufgerufene Objekt im **Debug Break**-Fenster angezeigt (`DEBUG2N` zuerst und dann `DEBUG3N` und `DEBUG4N`). So können Sie leicht feststellen, welche Objekte an welcher Stelle während der Programmausführung aufgerufen werden. Zusätzlich können Sie für jedes Objekt die Menüfunktionen des **Debug Break**-Fensters anwenden.

4 Wenn die Eingabeaufforderung `NEXT` erscheint, löschen Sie alle zurzeit gesetzten Breakpoints, indem Sie Folgendes eingeben:

```
TEST DEL BP * *
```

Es erscheint eine Meldung, die bestätigt, dass alle Breakpoints gelöscht sind.

Sitzung 5 - Verwendung von Statistiken über die Programmausführung

Mit dem Debugger können Sie statistische Informationen darüber anzeigen, welche Objekte wie oft aufgerufen werden. Zusätzlich können Sie herausfinden, welche Statements ausgeführt werden und wie oft.

Um zu prüfen, welche Objekte während der Programmausführung aufgerufen werden:

1 Geben Sie Folgendes an der Eingabeaufforderung `NEXT` ein:

```
TEST SET CALL ON
```

Die Meldung `Call statistics started.` wird angezeigt. Sie bestätigt, das die Statistik-Funktion aktiviert ist.

2 Führen Sie das Programm DEBUG2P aus.

Der Debugger protokolliert alle ausgeführten Objektaufrufe. Der vom Programm erstellte Bericht wird ausgegeben.

3 Geben Sie Folgendes an der Eingabeaufforderung NEXT ein:

```
TEST DIS CALL
```

Der Bildschirm **Display Called Objects** erscheint (Beispiel):

```
10:43:47              ***** NATURAL TEST UTILITIES *****             2007-09-06
Test Mode ON               - Aufgerufene Objekte anzeigen-            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
```

Der Bildschirm zeigt eine Liste aller ausgeführten Objekte: Das aufrufende Programm (DEBUG2P) und alle anderen aufgerufenen Objekte (DEBUG2N, DEBUG3N und DEBUG4N). Außerdem wird angezeigt, wie oft ein Objekt aufgerufen wird (CALLS), der Typ des aufgerufenen Objekts, wo und unter welcher Natural-Version das Objekt gespeichert ist, ob Source-Objekte und katalogisierte Objekte existieren und wann das Objekt katalogisiert wurde.

4 Drücken Sie PF3 (Exit) oder PF12 (Canc) mehrmals, bis die Eingabeaufforderung NEXT erscheint.

Um zu prüfen, welche Statements während der Programmausführung ausgeführt werden:

1 Geben Sie Folgendes an der Eingabeaufforderung NEXT ein:

```
TEST SET XSTAT COUNT
```

Die Meldung `Statement execution counting started for library/object */*.` bestätigt, dass die Statistik-Funktion für alle Objekte aktiviert ist, die in der aktuellen Library und in allen, mit dieser Library verketteten Steplibs enthalten sind.

2 Führen Sie das Programm DEBUG2P aus.

Der Debugger protokolliert alle von dem Programm verarbeiteten Statements. Anschließend wird der vom Programm erstellte Bericht ausgegeben.

3 Geben Sie Folgendes am Eingabeaufforderungszeichen `NEXT` ein:

```
TEST DIS XSTAT
```

Der Bildschirm **List Statement Execution Statistics** erscheint (Beispiel):

```
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
```

Der Bildschirm zeigt eine Liste. Diese enthält die Anzahl der Aufrufe (`Obj. Called n Times`), die Anzahl der ausführbaren Statements (`Exec(ecut)able`), die Anzahl der ausgeführten Statements (`Executed`), den Prozentsatz der ausgeführten Statements im Verhältnis zur Gesamtzahl der ausführbaren Statements (`%`) und die Gesamtzahl der ausgeführten Statements (`Total No. Executions`).

4 Geben Sie in der Spalte **Co** neben `DEBUG4N` Folgendes ein:

```
DS
```

Folgender Statistik-Bildschirm wird angezeigt (Beispiel):

```
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
```

Der Bildschirm zeigt an, wie oft ein Statement ausgeführt wurde und welche ausführbaren Statements nicht bearbeitet wurden.

Weitere Hinweise zur Benutzung des Debuggers

Dieser Abschnitt enthält zusätzliche Hinweise zur Benutzung des Debuggers.

- Zeitstempel von Objekten
- Breakpoints und Watchpoints speichern
- Debug-Hauptmenü (Main Menu) für Verwaltungsfunktionen
- Hilfe zu Kommandos in Maintenance-Bildschirmen
- Während der Programmunterbrechung verfügbare Funktionen
- Next-Option für zusätzliche Kommandos während der Programmunterbrechung
- Große Variablen und Arrays anzeigen
- Debugger-Statistikberichte drucken

- Debugger im Batch-Modus benutzen

Zeitstempel von Objekten

Ein katalogisiertes Objekt, das nicht genau mit dem Source-Objekt übereinstimmt, kann zu Fehlern bei der Fehlersuche führen. Wenn Sie sicherstellen wollen, dass Source-Objekt und katalogisiertes Objekt übereinstimmen, speichern und katalogisieren Sie sie mit dem Systemkommando `STOW`.

Weitere Informationen siehe *Voraussetzungen für den Betrieb*.

Breakpoints und Watchpoints speichern

Sie können die in der aktuellen Sitzung gesetzten Breakpoints und Watchpoints als Debug-Umgebung speichern und diese Umgebung zur Verwendung in einer zukünftigen Sitzung laden. Dies ist hilfreich, wenn Sie eine Anwendung wiederholt mit denselben Debug-Einträgen testen wollen.

Weitere Informationen siehe Abschnitt *Debug Environment Maintenance (Verwaltung der Debug-Umgebung)*.

Debug-Hauptmenü (Main Menu) für Verwaltungsfunktionen

Alle Verwaltungsfunktionen des Debuggers, wie z. B. das Setzen eines Breakpoint oder das Erstellen einer Statistik, können entweder mit einem Direktkommando oder mit den Verwaltungsfunktionen im Hauptmenü (**Debug Main Menu**) ausgeführt werden. Um das Hauptmenü aufzurufen, haben Sie folgende Eingabemöglichkeiten:

- `TEST` an einer Eingabeaufforderung.
- `MENU` in der Kommandozeile eines Debugger-Bildschirms.
- `M` im Feld **Code** im Fenster **Debug Break**.

Hilfe zu Kommandos in Maintenance-Bildschirmen

Sie können sich eine Liste der in einem Debugger-Maintenance-Bildschirm zur Verfügung stehenden Direktkommandos anzeigen lassen, indem Sie `PF1` (Help) drücken oder in der Kommandozeile ein Fragezeichen (?) eingeben.

In einem Debugger-Maintenance-Bildschirm mit Listeneinträgen stehen außerdem Zeilenkommandos zur Verfügung, die Sie benutzen können, um einen Eintrag zu bearbeiten. Die Eingabe eines Zeilenkommandos erfolgt neben dem betreffenden Eintrag in der Spalte **Co**. Sie können in dieser Spalte ein Fragezeichen (?) eingeben, um eine Liste der gültigen Zeilenkommandos angezeigt zu bekommen.

Während der Programmunterbrechung verfügbare Funktionen

Dieser Abschnitt enthält eine Liste der wichtigsten, während der Programmunterbrechung verfügbaren Funktionen. Sie können sie entweder im Fenster **Debug Break** oder in der Kommandozeile eines Debugger-Maintenance-Bildschirms ausführen.

Code im Debug-Fenster	Alternatives Direktkommando	Funktion
G	GO	Fortsetzung der Programmausführung, bis das nächste Debug-Ereignis auftritt.
L	LIST BREAK	Auflistung der Objekt-Source bei der Statement-Zeile, in der das Debug-Ereignis auftritt.
N	NEXT	Ausführung des nächsten Break-Kommandos, wenn es für einen Breakpoint oder Watchpoint angegeben wurde. Siehe auch *Next-Option für zusätzliche Kommandos während der Programmunterbrechung*.
R	RUN	Ausschalten des Test-Modus und Fortsetzen der Programmausführung.
S	STEP	Zeilenweise Verarbeitung der ausführbaren Statements.
V	DIS VAR	Anzeige einer Liste der Variablen, die für das unterbrochene Objekt definiert sind.

Next-Option für zusätzliche Kommandos während der Programmunterbrechung

Wenn Sie einen Breakpoint oder Watchpoint anzeigen oder ändern, werden Sie feststellen, dass an jeden von ihnen das Debugger-Kommando BREAK angehängt ist. Dieses Kommando ruft das **Debug-Break**-Fenster auf und darf nicht entfernt werden. Sie können jedoch zusätzliche Debugger-Kommandos angeben, die während der Programmunterbrechung nach dem BREAK-Kommando ausgeführt werden sollen. Ein zusätzlicher Befehl wird ausgeführt, wenn Sie entweder das Kommando NEXT in der Kommandozeile oder den Funktionscode N im **Debug Break**-Fenster eingeben.

Sie können die Debugger-Kommandos, wie im folgenden Beispiel gezeigt, im Feld **Commands** des entsprechenden Breakpoint- oder Watchpoint-Maintenance-Bildschirm eingeben:

```
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
```

Im obigen Beispiel weist das Kommando `STACK` den Debugger an, den Natural Stack zu untersuchen. Das Kommando `DIS VAR BONUS` weist den Debugger an, die angegebene Variable anzuzeigen. Dies ist z. B. hilfreich, wenn Sie in einer Schleife einen Breakpoint setzen und immer nur den Wert einer bestimmten Variablen sehen wollen. Sie müssen dann nicht wiederholt das Kommando `DIS VAR` eingeben.

Weitere Informationen siehe Beschreibung des Feldes **Commands** in den Abschnitten *Felder und Spalten in Breakpoint-Bildschirmen* und *Felder und Spalten in Watchpoint-Bildschirmen*.

Große Variablen und Arrays anzeigen

Der Bildschirm **Display Variable** zeigt alle Definitionen einer Variablen und zeigt ihren Inhalt in alphanumerischem oder hexadezimalem Format. Informationen zu Anzeigemöglichkeiten bei großen Variablen, deren Inhalt über den aktuellen Bildschirm hinausgeht, oder bei Variablen mit Array-Definitionen siehe Abschnitt *Variable anzeigen - einzeln*.

Debugger-Statistikberichte drucken

Sie können die vom Debugger erstellten Statistikberichte drucken oder auf einen PC herunterladen.

Weitere Informationen siehe ***Objekte drucken*** im Abschnitt *Call Statistics Maintenance (Statistiken über gerufene Objekte)* und ***Statements drucken*** im Abschnitt *Statement Execution Statistics Maintenance*.

Debugger im Batch-Modus benutzen

Der Debugger ist hauptsächlich für die interaktive Bedienung im Online-Modus ausgelegt. Zwar können Sie prinzipiell alle Debugger-Funktionen im Batch-Modus ausführen, die Verarbeitung von Online-Vorgängen im Batch-Modus (z.B. der Gebrauch von PF-Tasten) kann aber eine komplexe Batch-Programmierung erfordern. Es gibt jedoch Debugger-Funktionen, bei denen Batch-Verarbeitung eine komfortable Alternative darstellt. Eine Möglichkeit ist zum Beispiel das Sammeln und Drucken von Statistikdaten über eine Anwendung, siehe Abschnitt ***Beispiel für das Erstellen und Drucken von Statistiken im Batch-Modus*** im Abschnitt *Batch-Verarbeitung*.

Sourcecode-Beispiele

Dieser Abschnitt enthält den Sourcecode der Beispiel-Programme und -Subprogramme, die in den Sitzungen 1 bis 5 benötigt werden.

Programm `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
```

Subprogramm 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
```

Programm 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
```

Subprogramm 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                                                                  ↩
```

Subprogramm DEBUG3N

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

Subprogramm 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 Debugger-Konzept

Der Debugger übernimmt die Kontrolle über eine Natural-Sitzung zu Debugging-Zwecken, während ein Natural-Objekt ausgeführt wird. Dies ermöglicht es Ihnen, den Verarbeitungsablauf eines Programms zu verfolgen und verschiedene Programmuntersuchungen durchzuführen.

Sie können in einem Programm Stellen angeben, an denen der Debugger anhalten soll, indem Sie für das betreffende Programm Debug-Einträge (Breakpoints oder Watchpoints) setzen.

Wenn die Programmausführung pausiert, können Sie den Inhalt der im Programm verwendeten Variablen oder Parameter überprüfen, um die Programmlogik zu analysieren, oder Sie können den Grund für einen Natural-Fehler ermitteln.

In den folgenden Abschnitten finden Sie Informationen über die Funktionen, die im Debugger zur Verfügungs stehen.

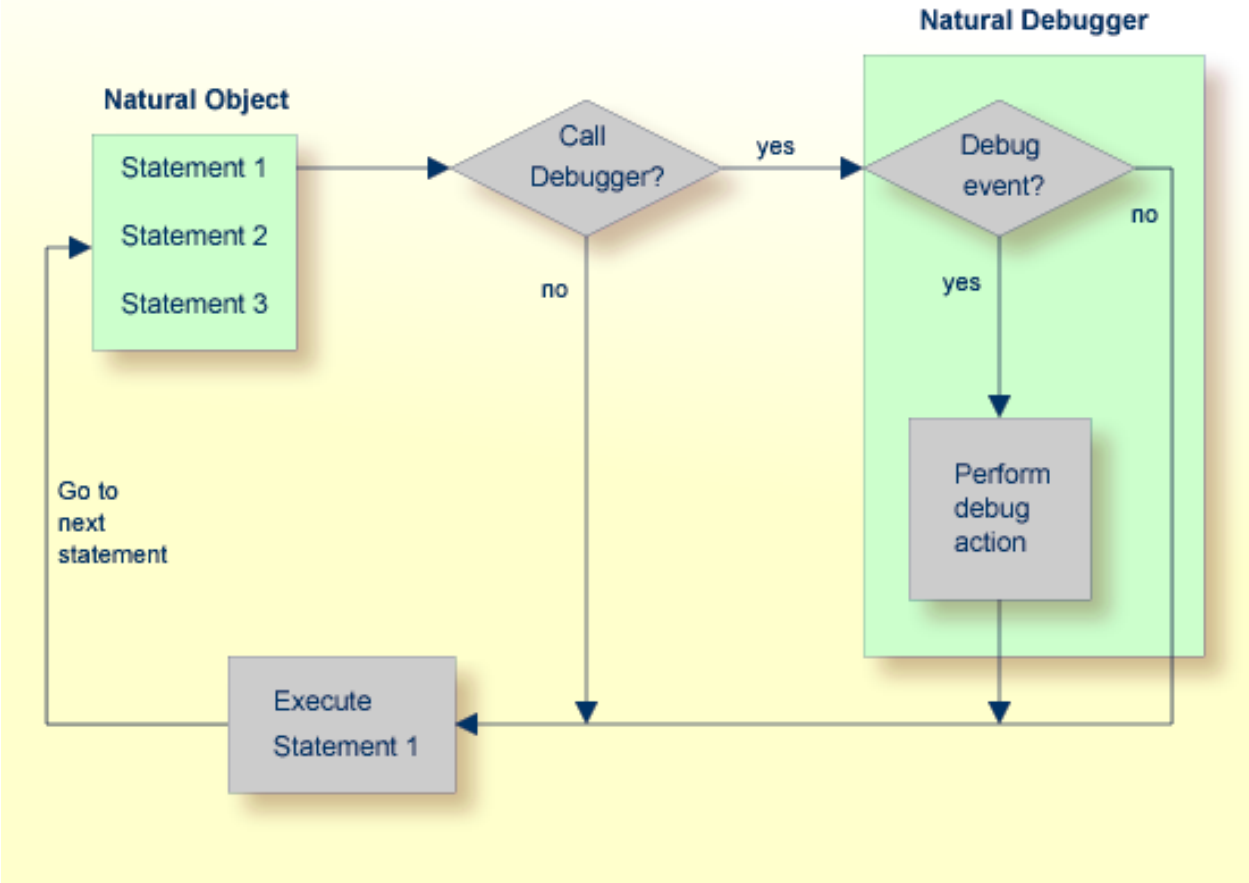
Funktionen zur Kontrolle über die Natural-Sitzung

Der Debugger enthält die Kontrolle über eine Natural-Sitzung, wenn der Test-Modus auf `ON` gesetzt ist (siehe ***Test-Modus ein- und ausschalten***). Ist der Natural-Profilparameter `DBGERR` auf `ON` gesetzt, wird der Debugger aufgerufen, wenn ein Natural-Fehler auftritt, und zwar unabhängig davon, ob Debug-Einträge vorhanden sind oder ob der Test-Modus auf `ON` oder `OFF` gesetzt ist

Wenn der Debugger die Kontrolle über eine Sitzung hat, führt er eine oder mehrere der folgenden Funktionen aus:

- Er prüft die Debug-Einträge.
- Er unterbricht ein Natural-Objekt an der Statement-Zeile, für die ein Breakpoint gesetzt wurde.
- Er unterbricht ein Natural-Objekt, wenn sich der Wert einer Variablen, für die ein Watchpoint gesetzt wurde, geändert hat.
- Er zeigt Informationen zu den gefundenen Debug-Einträgen (Watchpoint und/oder Breakpoint) an.
- Er liefert Statistiken zu den aufgerufenen Natural-Objekten.
- Er liefert Statistiken zu den in einem Natural-Objekt aufgerufenen Statements.
- Er unterbricht ein Natural-Objekt, wenn ein Natural-Fehler auftritt. Siehe auch Abschnitt ***Fehlerbehandlung***.

Die folgende Grafik zeigt ein Beispiel für den Ablauf bei der Ausführung eines Objekts mit dem Debugger:



Debug-Einträge (Spies)

Debug-Einträge werden in der Debugger-Umgebung auch als **Spies** bezeichnet. Es stehen zwei Arten von Spies zur Verfügung: Breakpoints und Watchpoints.

Folgende Themen werden behandelt:

- Verwaltung und Validierung der Debug-Einträge
- Namen von Debug-Einträgen
- Anfangszustand oder aktueller Zustand
- Zähler für Debug-Ereignisse

- Kommandos für Debug-Einträge

Verwaltung und Validierung der Debug-Einträge

Debug-Einträge für die aktuelle Debug-Sitzung können mit den entsprechenden Debugger-Maintenance-Funktionen, die in den entsprechenden Abschnitten der Debugger-Dokumentation beschrieben sind, gesetzt, geändert, aufgelistet, angezeigt, aktiviert, deaktiviert und gelöscht werden. Debug-Einträge können auch zur späteren Verwendung gespeichert werden, siehe Abschnitt *Debug Environment Maintenance (Verwaltung der Debug-Umgebung)*.

Die Validierung von Debug-Einträgen erfolgt entweder sofort bei der Definition eines Breakpoints oder Watchpoints auf dem entsprechenden Maintenance-Bildschirm oder während der Programmausführung.

Falls während der Programmausführung eine Validierung fehlschlägt, erscheint im Fenster **Debug Break** die Anmerkung `Check for invalid spy definition` (Auf ungültige Spy-Definition prüfen), siehe ***Debug Break-Fenster***. Außerdem wird der ungültige Breakpoint oder Watchpoint in den relevanten Bildschirmen zur Verwaltung der Breakpoints oder Watchpoints markiert.

Wird ein Debug-Eintrag gesetzt oder geändert, speichert Natural intern die Library, die Datenbankkennung und die Dateinummer, wo sich das Objekt befindet. Das Objekt kann sich in der aktuellen Library oder in einer ihrer Steplibs befinden. Wird später ein Objekt mit demselben Namen aus einer anderen Library ausgeführt, dann wird der entsprechende Debug-Eintrag nicht ausgeführt.

Namen von Debug-Einträgen

Der Debugger vergibt für jeden Debug-Eintrag einen Namen und eine eindeutige Nummer (Spy-Nummer). Der einem Debug-Eintrag zugewiesene Name (auch als Spy-Name bezeichnet) kann entweder ein vom Benutzer angegebener Name oder ein Standardname sein, der vom Debugger erstellt wird. Ein Debug-Eintrag kann über seine Nummer mit den entsprechenden Debugger-Kommandos ausgewählt werden. Wenn mehr als ein Debug-Eintrag an einer bestimmten Statement-Zeile ausgeführt werden muss, werden diese in aufsteigender Reihenfolge ihrer Nummern ausgeführt.

Anfangszustand oder aktueller Zustand

Jeder Debug-Eintrag hat einen Anfangszustand und einen aktuellen Zustand. Mögliche Werte sind `A` (aktiv) und `I` (inaktiv). Der Anfangswert wird angegeben, wenn Sie den Breakpoint oder Watchpoint setzen oder ändern. Er bestimmt den Zustand des Debug-Eintrags beim Start der Umgebung oder nach einem Reset. Während der Debug-Sitzung kann der Zustand mit den Debug-Kommandos `ACTIVATE` und `DEACTIVATE` geändert werden (siehe auch die Syntax-Diagramme in ***Debug-Kommandoübersicht und -syntax***).

Zähler für Debug-Ereignisse

Jeder Debug-Eintrag hat einen Ereigniszähler, der jedes Mal erhöht wird, wenn der Debug-Eintrag ausgeführt wird. Ein Debug-Eintrag wird nicht ausgeführt, wenn der aktuelle Zustand inaktiv ist. Die Ereignisanzahl des Breakpoint oder Watchpoint wird ebenfalls nicht erhöht.

Die Anzahl der Ausführungen eines Debug-Eintrags kann auf zwei Arten eingeschränkt werden:

- Es kann eine Anzahl von Auslassungen angegeben werden, bevor der Debug-Eintrag ausgeführt wird. Der Debug-Eintrag wird dann so lange ignoriert, bis die Ereignisanzahl höher ist als die angegebene Anzahl der Auslassungen.
- Es kann eine maximale Anzahl von Ausführungen angegeben werden, so dass der Debug-Eintrag ignoriert wird, sobald die Ereignisanzahl die angegebene Anzahl der Ausführungen überschreitet.

Kommandos für Debug-Einträge

Zu jedem Debug-Eintrag (Breakpoint oder Watchpoint) können bis zu sechs Debug-Kommandos angegeben werden. Diese Kommandos werden zur Ausführungszeit des Breakpoint oder Watchpoint ausgeführt. Sie können alle Debugger-Kommandos verwenden, die während einer Debug-Unterbrechung angewendet werden können. Das Standardkommando ist das `BREAK`-Kommando, mit dem das **Debug Break**-Fenster angezeigt wird. Siehe folgenden Abschnitt *Debug Break-Fenster*.

Vorsicht: Wenn Sie beim Setzen eines Debug-Eintrags das Kommando `BREAK` löschen und kein Kommando eingeben, das einen Dialog auslöst, gibt es keine Möglichkeit, die Steuerung während einer Programmunterbrechung zu übernehmen.

Debug Break-Fenster

Wenn der Debugger die Kontrolle über eine Sitzung übernimmt, wird ein **Debug Break**-Fenster angezeigt, zum Beispiel:

```
+------------------- 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    !
+----------------------------------------------------+
```

Das Fenster **Debug Break** zeigt den Typ und den Namen des Debug-Eintrags, der die Unterbrechung verursacht hat (d. h. den Namen des entsprechenden Breakpoint oder Watchpoint), seine Sourcecode-Zeilennummer und den Namen des unterbrochenen Natural-Objekts.

Zusätzlich können unten im **Debug Break**-Fenster Meldungen erscheinen, die entweder einen Natural-Fehler anzeigen (siehe auch *Fehler während der Anwendungsausführung* in *Fehlerbehandlung*) und/oder auf die Möglichkeit eines ungültigen Debug-Eintrags hinweisen.

Die Funktionen, die im Fenster Debug Break zur Verfügung stehen, sind in der folgenden Tabelle beschrieben. Weitere Informationen siehe *Kommandos zur Ausführungssteuerung*.

Funktion	Code	Beschreibung
Go	**G**	Fortsetzung der Programmausführung, bis zum nächsten angebenen Debug-Eintrag.
List break	**L**	Auflistung des Sourcecode des zurzeit aktiven Natural-Objekts. Das zuletzt ausgeführte Statement ist hervorgehoben.
Debug Main Menu	**M**	Aufruf des **Debug Main Menu**, in dem Sie alle Funktionen benutzen können, die Sie zur Verwaltung von Debug-Einträgen benötigen, bei denen die Steuerung übernommen werden soll.
Next break command	**N**	Ausführung des nächsten Kommandos, das für den aktuellen Breakpoint oder Watchpoint angegeben ist.
Run (set test mode OFF)	**R**	Fortsetzung der Ausführung des Natural-Objekts mit ausgeschaltetem Test-Modus.
Step mode	**S**	Fortsetzung der Ausführung des Natural-Objekts im Step-Modus (Einzelschritt-Modus).
Variable maintenance	**V**	Anzeige der Variablen in dem zurzeit aktiven Natural-Objekt und Änderung des Inhalts dieser Variablen.

4 Debugger starten

Dieser Abschnitt beschreibt Grundvoraussetzungen für den Betrieb und enthält eine grobe Richtlinie für die Vorgehensweise bei der Planung der Anwendung des Debugger.

Debugger unter Natural Security

Die Benutzung des Debugger kann durch Natural Security kontrolliert werden:

- Sie können den Debugger gegen nicht authorisierte Benutzung schützen, indem Sie den Gebrauch des Systemkommandos `TEST`, das den Debugger aufruft, unterbinden, siehe *Command Restrictions* im Abschnitt *Library Maintenance* in der *Natural Security*-Dokumentation.
- Sie können die Benutzung des Debuggers unterbinden oder einschränken, siehe Abschnitt *Components of an Environment Profile* in der *Natural Security*-Dokumentation.

Voraussetzungen für den Betrieb

Der Debugger wird nur aufgerufen, wenn Sie ein katalogisiertes Objekt ausführen, das in der aktuellen Library in der aktuellen Natural-Systemdatei befindet. Der Debugger wird nicht aufgerufen, wenn Sie im Arbeitsbereich enthaltenen Sourcecode mit dem Kommando `RUN` ausführen.

Ein effizientes und korrektes Debugging setzt voraus, dass der Sourcecode im Source-Objekt mit dem kompilierten Sourcecode im katalogisierten Objekt übereinstimmt, was mit dem Systembefehl `STOW` gewährleistet werden kann. Wenn Sie ein Source-Objekt ändern, nachdem Sie es katalogisiert haben, besteht die Möglichkeit, dass ein Debug-Eintrag (Breakpoint oder Watchpoint) nicht richtig funktioniert, weil sich das referenzierte Statement oder die referenzierte Variable geändert hat oder nicht mehr existiert. Wenn der Debugger feststellt, das ein Source-Objekt einen früheren Zeitstempel hat als das entsprechende katalogisierte Objekt erscheint die Warnung `Time stamps of source and cataloged object do not match`. (Zeitstempel des Source-Objekts und des katalogisierten Objekts passen nicht zueinander).

Der Debugger untersucht alle Natural-Objekte, die in der aktuellen Library oder in einer ihrer Steplibs enthalten sind. Der Debugger untersucht keine Natural-Objekte, die in der Natural-System-Library `SYSLIB` oder `SYSLIBS` gespeichert sind.

Für die Verwendung des Debugger gilt folgende Einschränkung:

- Der Debugger kann nur bei Objekten der Natural-Version 2.3 und höher, jedoch nicht bei Natural-Objekten, die mit einer früheren Version katalogisiert wurden, angewendet werden.

  Der Debugger unterstützt nur Debug-Umgebungen, die mit Natural-Version 2.3 und höher erstellt wurden. Debug-Umgebungen, die mit einer früheren Version erstellt wurden, werden ignoriert. Weitere Informationen zu Debug-Umgebungen siehe ***Debug Environment Maintenance (Verwaltung der Debug-Umgebung)***.

Batch-Verarbeitung

Obwohl der Debugger hauptsächlich für den interaktiven Gebrauch im Online-Modus ausgelegt ist, können die Debugger-Kommandos auch zur Batch-Ausführung benutzt werden, z.B. zum Setzen von Breakpoints oder Watchpoints.

Anmerkung: Bei der Batch-Verarbeitung gelten Einschränkungen, die zur Folge haben können, dass ein Debugger-Kommando zurückgewiesen wird. Beispielsweise unterstützt der Debugger nicht die Kommandos `++` und `+4`.

Beispiel für das Erstellen und Drucken von Statistiken im Batch-Modus

Das folgende Beispiel demonstriert den Gebrauch von Debugger-Direktkommandos im Batch-Modus, um einen Bericht über alle Aufrufstatistiken zu erstellen und zu drucken:

```
//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
/*
```

Debugger aufrufen

Um den Debugger aufzurufen:

1 Erstellen Sie eine Debug-Umgebung für ein Natural-Objekt oder eine Natural-Anwendung:

- Rufen Sie das Hauptmenü **Debug Main Menu** auf, indem Sie das Natural-Systemkommando `TEST` eingeben.

  Oder:
  Setzen Sie aus einer laufenden Anwendung das Terminalkommando `%<TEST` ab.

- Benutzen Sie die Funktionen des Hauptmenüs **Debug Main Menu**, um für ein Natural-Objekt oder eine Natural-Anwendung anzugeben:

  Debug Environment Maintenance (Verwaltung der Debug-Umgebung)
  Spy Maintenance (Verwaltung der Debug-Einträge)
  Breakpoint Maintenance (Verwaltung der Breakpoints)
  Watchpoint Maintenance (Verwaltung der Watchpoints)
  Call Statistics Maintenance (Statistiken über gerufene Objekte)
  Statement Execution Statistics Maintenance (Statistiken über ausgeführte Statement-Zeilen)
  Variable Maintenance
  List Object Source

2 Aktivieren Sie den Debugger:

- Geben Sie an einer Eingabeaufforderung das Kommando `TEST ON` ein.

  Oder:
  Geben Sie im Hauptmenü **Debug Main Menu** den Funktionscode `T` ein.

3 Führen Sie das Natural-Objekt oder die Natural-Anwendung aus.

  Der Debugger hält die Programmausführung an den angegebenen Debug-Einträgen an und ruft das **Debug Break**-Fenster auf.

Um den Debugger zur Fehlerbehandlung aufzurufen:

- Setzen Sie zu Beginn der Sitzung den Natural-Profilparameter `DBGERR` auf `ON`.

  Siehe auch *DBGERR - Automatischer Debugger-Start bei Laufzeitfehler* in der Natural-*Parameter-Referenz*-Dokumentation.

  Oder:

  Geben Sie während der Sitzung an einer Eingabeaufforderung das Kommando `TEST ON` ein oder geben Sie in einem **Debug Maintenance**-Hauptmenü den Funktionscode `T` ein.

  Der Debubber ruft das **Debug Break**-Fenster auf, wenn ein Natural-Fehler auftritt.

Siehe auch Abschnitt ***Fehlerbehandlung***.

Standard-Objekt

Die Maintenance-Funktionen des Debuggers, wie sie in den entsprechenden Abschnitten beschrieben sind, beziehen sich auf Objekte, die Sie entweder in den entsprechenden Namensfeldern von Menüs oder mit Direktkommandos angeben. Wenn Sie keinen Objektnamen angeben, nimmt der Debugger standardmäßig den Namen des aktuellen Objekts an, wie er im Feld **Object** oben rechts im Hauptmenü **Debug Main Menu** angezeigt wird. Bei Angabe eines Standardobjekts, ist kein Objektname in Direktkommandos und Menüoptionen erforderlich, die zur Angabe von Breakpoints oder Watchpoints verwendet werden. Um das Standardobjekt zu ändern, siehe Syntax des Kommandos `SET` im Abschnitt ***Debug-Kommandoübersicht und -syntax***.

5 Test-Modus ein- und ausschalten

Um eine zuvor erstellte Debug-Umgebung zu aktivieren, müssen Sie den Test-Modus auf `ON` setzen.

Um den Test-Modus auf ON oder OFF zu setzen:

- Geben Sie in einem **Debug Maintenance**-Hauptmenü den Funktionscode `T` ein, um den Test-Modus auf `ON` (Ein) oder `OFF` (Aus) zu setzen.

  Oder:

  Geben Sie eines der folgenden Direktkommandos ein:

  ```
  TEST ON
  ```

  oder

  ```
  TEST OFF
  ```

Wenn Sie bei eingeschaltetem Test-Modus (`ON`) ein Natural-Objekt ausführen, prüft der Debugger kontinuierlich alle Debug-Einträge ab, ob eventuell eine Maßnahme erforderlich ist.

Wenn Sie bei ausgeschaltetem Test-Modus (`OFF`) ein Natural-Objekt ausführen, werden alle Debug-Einträge ignoriert.

Das Kommando `TEST` und somit die gesamte Anwendung können durch Natural Security geschützt werden, siehe *Command Restrictions* im Abschnitt *Library Maintenance* in der *Natural Security*-Dokumentation.

6 Debug Environment Maintenance (Verwaltung der Debug-Umgebung)

Da eine Debug-Umgebung hauptsächlich aus Debug-Einträgen besteht, wird sie erstellt, indem Breakpoints und Watchpoints gesetzt werden, wie in den entsprechenden Maintenance-Abschnitten beschrieben.

Nachdem eine Debug-Umgebung erstellt worden ist, kann sie zur anschließenden Benutzung gespeichert werden. Die Datei, in der Debug-Umgebungen gespeichert werden, kann mit dem Debugger-Kommando `PROFILE` angegeben werden (siehe *Kommandos zum Navigieren und Anzeigen von Informationen*). Sie können eine Debug-Umgebung auch löschen oder ihre Zähler auf die Anfangswerte zurücksetzen.

Anmerkung: Siehe auch die Einschränkungen, die für die Benutzung gelten, im Abschnitt *Voraussetzungen für den Betrieb*.

Die folgenden Elemente sind ebenfalls Teil einer Debug-Umgebung und werden daher jedes Mal gespeichert oder geladen, wenn Sie eine Debug-Umgebung speichern oder laden:

- Die Test-Modus-Einstellung (`ON` oder `OFF`),
- alle Optionen, die mit dem Debugger-Kommando `PROFILE` gesetzt werden können (mit Ausnahme der Datei zum Laden und Speichern von Debug-Umgebungen),
- die Einstellungen der Funktion **Statement execution statistics maintenance** (`ON`, `OFF` oder `COUNT`).

Um die Funktion Debug Environment Maintenance aufzurufen:

- Geben Sie im Hauptmenü **Debug Main Menu** den Funktionscode `E` ein.

  Oder:

  Geben Sie folgendes Direktkommando ein:

  ```
  EM
  ```

  Das Menü **Debug Environment Maintenance** erscheint.

Dieser Abschnitt beschreibt die Funktionen, die im Menü **Debug Environment Maintenance** zur Verfügung stehen und enthält Anleitungen, wie Sie Maintenance-Funktionen in verschiedenen Libraries ausführen können.

Bei jeder gewählten Funktion müssen Sie den Namen der Debug-Umgebung angeben, die Sie verwalten wollen.

Test-Modus ON/OFF setzen

Siehe Abschnitt *Test-Modus ein- und ausschalten*.

Debug-Umgebung laden

Um eine Debug-Umgebung aus Ihrer Benutzer-Systemdatei (FUSER) zu laden:

- Geben Sie im Menü **Debug Environment Maintenance** den Funktionscode L und den Namen der Umgebung ein.

  Oder:

  Geben Sie folgendes Direktkommando ein:

  ```
  LOAD ENVIRONMENT name
  ```

  Die angegebene Debug-Umgebung wird geladen.

Falls Sie keinen Namen angeben, wird die Standardumgebung mit dem Namen Noname geladen.

Um eine Liste aller verfügbaren Debug-Umgebungen zu erhalten, können Sie einen Stern (*) eingeben. In der Liste können Sie die gewünschte Umgebung mit dem Zeilenkommando LO markieren, um sie in den Debug-Pufferspeicher zu laden, oder mit dem Zeilenkommando DE, um sie zu löschen.

Debug-Umgebung speichern

Um eine Debug-Umgebung zu speichern:

- Geben Sie im Menü **Debug Environment Maintenance** den Funktionscode S und den Namen der Umgebung ein.

  Oder:

  Geben Sie folgendes Direktkommando ein:

  ```
  SAVE ENVIRONMENT name
  ```

Die angegebene Umgebung wird **zurückgesetzt** (siehe unten) und in der Datei gespeichert, die mit dem Debugger-Kommando `PROFILE` als Speicherort angegeben wird (siehe Abschnitt *Kommandos zum Navigieren und Anzeigen von Informationen*).

Wenn Sie keinen Namen angeben, wird die Standardumgebung mit dem Namen `Noname` gespeichert.

Falls bereits eine Debug-Umgebung mit dem angegebenen Namen existiert, werden Sie aufgefordert, zu bestätigen, dass die alte Umgebung überschrieben wird.

Debug-Umgebung zurücksetzen

Die Debug-Umgebung sollte vor jedem Testlauf zurückgesetzt werden. Das Zurücksetzen der Umgebung bewirkt Folgendes:

- Die aktuellen Zustände aller Debug-Einträge werden auf ihren Anfangszustand gesetzt.
- Alle Ereigniszählungen werden auf Null gesetzt.
- Die Aufrufstatistiken im Debug-Pufferspeicher werden gelöscht. Weitere Informationen siehe Abschnitt *Call Statistics Maintenance (Statistiken über gerufene Objekte)*.

Um eine Debug-Umgebung zurückzusetzen:

- Geben Sie im Menü **Debug Environment Maintenance** den Funktionscode `R` und den Namen der Umgebung ein.

  Oder:

  Geben Sie folgendes Direktkommando ein:

  ```
  RESET ENVIRONMENT name
  ```

  Die angegebene Debug-Umgebung wird zurückgesetzt.

Wenn Sie keinen Umgebungsnamen angeben, wird die aktuelle Debug-Umgebung zurückgesetzt.

Debug-Umgebung löschen

Um eine Debug-Umgebung zu löschen:

1 Geben Sie im Menü **Debug Environment Maintenance** den Funktionscode `D` und den Namen der Umgebung ein.

Oder:

Geben Sie folgendes Direktkommando ein:

```
DELETE ENVIRONMENT name
```

Ein Bestätigungsfenster erscheint.

2 Geben Sie im Bestätigungsfenster `Y` (Yes) ein, um die Löschung zu bestätigen.

Die angegebene Debug-Umgebung wird gelöscht.

Wenn Sie keinen Umgebungsnamen angeben, wird die aktuelle Debug-Umgebung gelöscht.

Debug-Umgebung in verschiedenen Libraries verwalten

Mit den Funktionen der Natural-Utility `SYSMAIN` können Sie Debug-Umgebungen zwischen verschiedenen Libraries und/oder Systemdateien kopieren oder verschieben und eine Debug-Umgebung löschen, auflisten oder umbenennen.

Wenn eine Debug-Umgebung von einer Library in eine andere verschoben oder kopiert worden ist, beziehen sich die Breakpoints und Watchpoints immer noch auf die alte (Quell-)Library. Um die Debug-Umgebung an die neue (Ziel-)Library anzupassen, müssen Sie die entsprechenden Breakpoints oder Watchpoints ändern (siehe auch ***Breakpoint ändern*** im Abschnitt *Breakpoint Maintenance (Verwaltung der Breakpoints)* bzw. ***Watchpoint ändern*** im Abschnitt *Watchpoint Maintenance (Verwaltung der Watchpoints)*). Beim Ausführen der `Modify`-Funktion brauchen Sie keine der existierenden Definitionen zu verändern. Wenn Sie das Kommando `Save` (`PF5`) zum Speichern ausführen, ändert sich die Library-Referenz automatisch auf die neue Library. Sie erkennen die Änderung am Eintrag im Feld **Library** im Bildschirm **Modify Breakpoint** bzw. **Modify Watchpoint**.

Verwandtes Thema:

- *Processing Debug Environments - SYSMAIN Utility*, *Utilities*-Dokumentation

7 Spy Maintenance (Verwaltung der Debug-Einträge)

Mit der Funktion **Spy Maintenance** (Verwaltung der Debug-Einträge) können Sie alle Debug-Einträge („Spies"), d.h. Breakpoints *und* Watchpoints, aktivieren, deaktivieren, auflisten oder löschen. Zudem bietet Ihnen die Funktion **Spy Maintenance** einen alternativen Zugang zu den Bildschirmen zur Verwaltung der Breakpoints und Watchpoints. Die Beschreibungen dieser Bildschirme finden Sie in den Abschnitten *Watchpoint Maintenance (Verwaltung der Watchpoints)* und *Breakpoint Maintenance (Verwaltung der Breakpoints)*.

Um die Funktion Spy Maintenance aufzurufen:

- Geben Sie im Hauptmenü **Debug Main Menu** den Funktionscode `S` ein.

  Oder:

  Geben Sie folgendes Direktkommando ein:

  ```
  SM
  ```

  Das Menü **Spy Maintenance** erscheint.

Die Funktionen, die im Menü **Spy Maintenance** zur Verfügung stehen, werden in den folgenden Abschnitten beschrieben:

Test-Modus ON/OFF setzen

Siehe Abschnitt *Switch Test Mode On and Off*.

Spy-Funktion aktivieren

Um den aktuellen Status angegebener Debug-Einträge (Spies) auf aktiv zu setzen:

- Geben Sie im Menü **Spy Maintenance** den Funktionscode `A` und eine Spy-Nummer *oder* einen Spy-Namen ein.

  Oder:

  Benutzen Sie das Direktkommando `ACTIVATE`. Syntax-Beschreibung siehe Abschnitt *Debug-Kommandoübersicht und -syntax*.

Wenn Sie die Spy-Nummer *oder* den Spy-Namen nicht eingeben, werden *alle* Spies (Breakpoints und Watchpoints) aktiviert.

Spy deaktivieren

Um den aktuellen Status angegebener Debug-Einträge (Spies) auf inaktiv zu setzen:

- Geben Sie im Menü **Spy Maintenance** den Funktionscode `B` und eine Spy-Nummer *oder* einen Spy-Namen ein.

  Oder:

  Benutzen Sie das Direktkommando `DEACTIVATE`. Syntax-Beschreibung siehe Abschnitt ***Debug-Kommandoübersicht und -syntax***.

Wenn Sie die Spy-Nummer *oder* den Spy-Namen nicht eingeben, werden *alle* Spies (Breakpoints und Watchpoints) deaktiviert.

Debug-Einträge (Spies) löschen

Um angegebene Debug-Einträge (Spies) zu löschen:

- Geben Sie im Menü **Spy Maintenance** den Funktionscode `C` und eine Spy-Nummer *oder* einen Spy-Namen ein.

  Oder:

  Benutzen Sie das Direktkommando `DELETE`. Syntax-Beschreibung siehe Abschnitt ***Debug-Kommandoübersicht und -syntax***.

Wenn Sie die Spy-Nummer *oder* den Spy-Namen nicht eingeben, werden *alle* Spies (Breakpoints und Watchpoints) gelöscht.

Debug-Einträge (Spies) anzeigen

Um angegebene Debug-Einträge (Spies) anzuzeigen:

- Geben Sie im Menü **Spy Maintenance** den Funktionscode `D` und eine Spy-Nummer *oder* einen Spy-Namen ein.

  Oder:

Benutzen Sie das Direktkommando `DISPLAY`. Syntax-Beschreibung siehe Abschnitt *Debug-Kommandoübersicht und -syntax*.

Ist der angegebene Debug-Eintrag (Spy) eindeutig, erscheint der Bildschirm **Display Breakpoint** bzw. **Display Watchpoint**, und es werden alle Angaben zu diesem Breakpoint bzw. Watchpoint angezeigt.

Wenn der angegebene Debug-Eintrag (Spy) nicht eindeutig ist, wird eine Liste der betreffenden Debug-Einträge (Spies) angezeigt. In der Liste können Sie einen Debug-Eintrag (Spy) mit einem Zeilenkommando markieren, um folgende Aktionen auszuführen:

Kommando	Aktion
`AC`	Debug-Eintrag aktivieren
`DA`	Debug-Eintrag deaktivieren
`DI`	Debug-Eintrag anzeigen
`MO`	Debug-Eintrag ändern
`DE`	Debug-Eintrag löschen

Wenn Sie die Spy-Nummer *oder* den Spy-Namen nicht eingeben, werden *alle* Spies (Breakpoints und Watchpoints) angezeigt.

Debug-Einträge (Spies) ändern

Um angegebene Debug-Einträge (Spies) zu ändern:

- Geben Sie im Menü **Spy Maintenance** den Funktionscode `M` und eine Spy-Nummer *oder* einen Spy-Namen ein.

  Oder:

  Benutzen Sie das Direktkommando `MODIFY`. Syntax-Beschreibung siehe Abschnitt *Debug-Kommandoübersicht und -syntax*.

  Ist der angegebene Debug-Eintrag (Spy) eindeutig, erscheint der Bildschirm **Modify Breakpoint** bzw. **Modify Watchpoint**, und die Angaben zu diesem Breakpoint bzw. Watchpoint können geändert werden.

  Wenn der angegebene Debug-Eintrag (Spy) nicht eindeutig ist, wird eine Liste der betreffenden Debug-Einträge (Spies) angezeigt. In der Liste können Sie einen Debug-Eintrag (Spy) mit einem Zeilenkommando markieren, um folgende Aktionen auszuführen:

Kommando	Aktion

AC	Debug-Eintrag aktivieren
DA	Debug-Eintrag deaktivieren
DI	Debug-Eintrag anzeigen
MO	Debug-Eintrag ändern
DE	Debug-Eintrag löschen

Wenn Sie die Spy-Nummer *oder* den Spy-Namen nicht eingeben, werden *alle* Spies (Breakpoints und Watchpoints) zum Auswählen und Ändern angezeigt.

8 Breakpoint Maintenance (Verwaltung der Breakpoints)

Ein Breakpoint (Haltepunkt) bewirkt die Unterbrechung der Ausführung eines Natural-Objekts an einer bestimmten Statement-Zeile.

In diesem Abschnitt erfahren Sie, wie und wann Sie Breakpoints setzen.

Beachten Sie, dass Sie die hier beschriebenen Verwaltungsfunktionen auch aus einem Objekt-Sourcecode heraus aufrufen können, indem Sie die Funktion **List object source** benutzen (siehe *List Object Source (Objekt-Sourcecode anzeigen)*)

Um die Breakpoint Maintenance (Verwaltung der Breakpoints) aufzurufen:

- Geben Sie im Hauptmenü **Debug Main Menu** den Funktionscode `B` ein.

  Oder:

  Geben Sie folgendes Direktkommando ein:

  ```
  BM
  ```

  Das Menü **Breakpoint Maintenance** erscheint.

Dieses Kapitel beschreibt Bedingungen für die Benutzung der Breakpoint-Verwaltung, die im Menü **Breakpoint Maintenance** zur Verfügung stehenden Funktionen und die in einem Breakpoint-Bildschirm vorhandenen Felder und Spalten.

Verwendungsbedingungen

Einen Breakpoint können Sie setzen, indem Sie den Namen des zu verarbeitenden Natural-Objekts und im Sourcecode des Objekts die Zeilennummer angeben, wo der Breakpoint ausgeführt werden soll.

Sobald ein Breakpoint angegeben wurde, bleibt er für die gesamte Natural-Sitzung gesetzt, es sei denn, Sie löschen ihn.

Ein Breakpoint bezieht sich auf eine bestimmte Zeilennummer im Sourcecode. Deshalb kann eine nachträgliche Änderung des Sourcecode zur Folge haben, dass der Breakpoint nicht mehr für das gewünschte Statement gilt und somit das Natural-Objekt nicht mehr an der gewünschten Stelle unterbrochen wird. Um dieses Problem bei Programmschleifen zu umgehen, können in diesen Schleifen Statement-Labels gesetzt werden (siehe auch Beispiel mit Statement-Labels im *Leitfaden zur Programmierung*). Für diese Statement-Labels gesetzte Breakpoints werden auf die korrekte Zeilennummer angepasst, wenn Statement-Zeilen eingefügt oder gelöscht werden.

Der eindeutige Bezeichner für einen Breakpoint ist die vom Debugger zugewiesene Spy-Nummer.

An folgenden Stellen können keine Breakpoints gesetzt werden:

- In Kommentarzeilen,
- in einer beliebigen Statement-Zeile außer der ersten Zeile (wenn ein einzelnes Statement mehr als eine Programmzeile belegt,
- in Zeilen, die lediglich eines der folgenden Statements enthalten:
  - `AT BREAK OF`
  - `AT END OF DATA`
  - `AT END OF PAGE`
  - `AT START OF DATA`
  - `AT TOP OF PAGE`
  - `BEFORE BREAK`
  - `DECIDE`

    Siehe auch die Einschränkungen, die für die Benutzung gelten, im Abschnitt ***Voraussetzungen für den Betrieb***.
  - `DEFINE SUBROUTINE`
  - `DEFINE WINDOW`
  - `FORMAT`
  - `IF NO RECORDS FOUND`
  - `ON ERROR`
  - `OPTIONS`

Ob es möglich ist oder nicht, Breakpoints für Zeilen zu setzen, die mit dem Natural Optimizer Compiler kompiliert wurden, ist abhängig von der Einstellung der Option `NODBG` im `OPTIONS`-Statement. Weitere Informationen siehe `OPTIONS`-Statement im Abschnitt *Switching on the Optimizer Compiler* in der *Natural Optimizer Compiler*-Dokumentation.

Test-Modus ON/OFF setzen

Siehe Abschnitt ***Test-Modus ein- und ausschalten***.

Breakpoint aktivieren

Um den aktuellen Status angegebener Breakpoints auf aktiv zu setzen:

- Geben Sie im Menü **Breakpoint Maintenance** den Funktionscode `A` und einen Objektnamen und/oder eine Zeilennummer ein.

  Oder:

  Benutzen Sie das Direktkommando `ACTIVATE`. Syntax-Beschreibung siehe Abschnitt *Debug-Kommandoübersicht und -syntax*.

  Wenn Sie den Objektnamen oder eine Zeilennummer nicht eingeben, werden *alle* Breakpoints aktiviert.

Breakpoint deaktivieren

Um den aktuellen Status angegebener Breakpoints auf inaktiv zu setzen:

- Geben Sie im Menü **Breakpoint Maintenance** den Funktionscode `B` und einen Objektnamen und/oder eine Zeilennummer ein.

  Oder:

  Benutzen Sie das Direktkommando `DEACTIVATE`. Syntax-Beschreibung siehe Abschnitt *Debug-Kommandoübersicht und -syntax*.

  Wenn Sie den Objektnamen oder eine Zeilennummer nicht eingeben, werden *alle* Breakpoints deaktiviert.

Breakpoint löschen

Um angegebene Breakpoints zu löschen:

- Geben Sie im Menü **Breakpoint Maintenance** den Funktionscode `C` und einen Objektnamen und/oder eine Zeilennummer ein.

  Oder:

Benutzen Sie das Direktkommando DELETE. Syntax-Beschreibung siehe Abschnitt ***Debug-Kommandoübersicht und -syntax***.

Wenn Sie den Objektnamen oder eine Zeilennummer nicht eingeben, werden *alle* Breakpoints gelöscht.

Breakpoint anzeigen

Um einen Breakpoint anzuzeigen:

- Geben Sie im Menü **Breakpoint Maintenance** den Funktionscode D, einen Objektnamen und/oder eine Zeilennummer ein.

  Wenn Sie den Objektnamen nicht angeben, wird das **Standard-Objekt** (falls angegeben) benutzt.

  Oder:

  Benutzen Sie das Direktkommando DISPLAY. Syntax-Beschreibung siehe Abschnitt *Debug-Kommandoübersicht und -syntax*.

  Wenn für das angegebene Objekt und Zeilennummer ein Breakpoint gesetzt worden ist, erscheint der Bildschirm **Display Breakpoint** mit allen Breakpoint-Definitionen, zum Beispiel:

```
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 -

Commands ... BREAK









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

Falls kein eindeutiger Breakpoint gefunden wird, erscheint der Bildschirm **List Breakpoints** (Beschreibung siehe unten).

Beschreibung der Felder im Bildschirm **Display Breakpoint** und der Spalten im Bildschirm **List Breakpoints** siehe *Felder und Spalten in Breakpoint-Bildschirmen*.

Um Breakpoints aufzulisten:

- Geben Sie im Menü **Breakpoint Maintenance** den Funktionscode `D`, einen Objektnamen oder eine Zeilennummer ein. Sie können Stern-Notation (*) benutzen, um einen Bereich von Objektnamen anzugeben, zum Beispiel: `ABC*`. Wenn Sie nur einen Stern (*) eingeben, werden alle Objektnamen ausgewählt. Wenn Sie keinen Objektnamen angeben, wird das **Standard-Objekt** (falls angegeben) benutzt.

  Oder:

  Benutzen Sie das Direktkommando `DISPLAY`. Syntax-Beschreibung siehe Abschnitt *Debug-Kommandoübersicht und -syntax*.

  Der Bildschirm **List Breakpoints** erscheint. Darin werden alle Breakpoints aufgelistet, die für das bzw. die Objekte oder eine Zeilennummer gesetzt sind, zum Beispiel:

```
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
```

Die Liste ist in aufsteigender Reihenfolge nach den Debug-Eintrag-Nummern (**Spy-Nummern**) sortiert, die in der Spalte **No.** enthalten sind.

Informationen zu den Spalten im Bildschirm **List Breakpoints** und zu den Zeilenkommandos, die bei einem Listeneintrag ausgeführt werden können, siehe *Felder und Spalten in Breakpoint-Bildschirmen*.

Breakpoint ändern

Um einen Breakpoint zu ändern:

1 Geben Sie im Menü **Breakpoint Maintenance** den Funktionscode `M`, einen Objektnamen und eine Zeilennummer ein. Wenn Sie keinen Objektnamen angeben, wird das **Standard-Objekt** (falls angegeben) benutzt.

Oder:

Benutzen Sie das Direktkommando `MODIFY`. Syntax-Beschreibung siehe Abschnitt *Debug-Kommandoübersicht und -syntax*.

Wenn ein eindeutiger Breakpoint angegeben wurde, erscheint der Bildschirm **Modify Breakpoint**, in dem Sie die Feldeinträge ändern können. Beschreibung der Felder im Bildschirm **Modify Breakpoint** siehe *Felder und Spalten in Breakpoint-Bildschirmen*.

Wird kein eindeutiger Breakpoint gefunden, dann erscheint der Bildschirm **List Breakpoints** (siehe Abschnitt *Breakpoint anzeigen*).

2 Wenn Sie die Bearbeitung der Breakpoint-Definitionen beendet haben, drücken Sie PF3 (Exit) oder PF5 (Save), um die Änderungen zu speichern. Informationen zu Validierungen an Debug-Einträgen siehe Abschnitt *Verwaltung und Validierung der Debug-Einträge*.

Wenn Sie PF12 (Canc) drücken, wird die Bearbeitung abgebrochen und der Breakpoint bleibt unverändert.

Breakpoint setzen

Um einen Breakpoint für eine Sitzung zu setzen:

- Geben Sie im Menü **Breakpoint Maintenance** den Funktionscode S, einen Objektnamen und/oder eine Zeilennummer ein.

  Oder:

  Benutzen Sie das Direktkommando SET. Syntax-Beschreibung siehe Abschnitt *Debug-Kommandoübersicht und -syntax*.

  Wenn Sie keinen Objektnamen, sondern eine gültige Zeilennummer angeben, wird der Name des **Standard-Objekts** (falls angegeben) benutzt. Falls kein Standard-Objekt angegeben ist, erscheint ein Auswahlfenster, in dem alle in der aktuellen Library enthaltenen Objekte angezeigt werden.

  Ein für einen Copycode gesetzter Breakpoint kann jedoch nur dann auf Gültigkeit geprüft werden, wenn ein Programm ausgeführt wird, das den Copycode enthält. Informationen zu Validierungen an Debug-Einträgen siehe Abschnitt *Verwaltung und Validierung der Debug-Einträge*.

  Der Breakpoint erhält das Standard-Kommando (BREAK), sein Anfangsstatus und sein aktueller Status werden auf aktiv gesetzt und es werden keine Ausführungseinschränkungen angegeben. Achtung: Wenn Sie beim Setzen eines Breakpoint das Kommando BREAK löschen und kein Kommando eingeben, das einen Dialog öffnet, hat der Debugger keine Möglichkeit, während der Programmunterbrechung die Steuerung zu übernehmen.

Felder und Spalten in Breakpoint-Bildschirmen

Die folgende Tabelle enthält die Beschreibung der Felder in den Bildschirmen **Display Breakpoint** und **Modify Breakpoint** und die Spalten im Bildschirm **List Breakpoints**:

Feld	Spalte	Erläuterung
Test Mode		Gibt an, ob der Test-Modus auf `ON` oder `OFF` gesetzt ist.
Object		Zeigt den Namen des **Standard-Objekts** (siehe *Debugger starten*), falls angegeben.
	Co	Feld zur Eingabe eines der folgenden Zeilenkommandos: **Kommando Aktion** `AC` Breakpoint aktivieren `DA` Breakpoint deaktivieren `DI` Breakpoint anzeigen `MO` Breakpoint ändern `DE` Breakpoint löschen `?` Gültige Zeilenkommandos auflisten `.` Breakpoint-Bildschirm verlassen
Spy number	**No.**	Eine eindeutige Nummer, die vom Debugger beim Setzen des Breakpoint zugewiesen wird.
Initial state	**Stat I**	Gibt den Anfangszustand (Initial) und den aktuellen Zustand (Current) des Breakpoint an: Aktiv (`A`) oder Inaktiv (`I`) an.
Current state	**Stat C**	
Breakpoint name	**BP Name**	Der Name des Breakpoint. Gültige Werte: 1 bis 12 Zeichen. Der Standard-Name für einen Breakpoint besteht aus dem Namen des Objekts und der Zeilennummer.
DBID/FNR	**DBID**	Die Datenbankkennung (DBID) und die Dateinummer (FNR) der Systemdatei, in der das Natural-Objekt gespeichert ist.
	FNR	
Library	**Library**	Der Name der Library, die das Objekt enthält.
Object name	**Object**	Der Name des Objekts, das in der aktuellen Library oder in einer ihrer Steplibs verfügbar ist.
Line number	**Line**	Die Zeilennummer eines Statement im Sourcecode des Objekts. Siehe auch *Verwendungsbedingungen*. Als Zeilennummern können Sie auch `BEG`, `END` oder `ALL` angeben: `BEG` Gibt den Breakpoint an, der die Programmausführung bei dem ersten Statement, das in einem Objekt ausgeführt wird, unterbrechen soll.

Feld	Spalte	Erläuterung
		BEG-Breakpoints können nicht für Copycode angegeben werden. END Gibt den Breakpoint an, der die Programmausführung bei dem letzten Statement, das in einem Objekt ausgeführt wird, unterbrechen soll, zum Beispiel, ein END- oder ein FETCH-Statement. END-Breakpoints können nicht für Copycode angegeben werden. ALL Gibt an, dass ein Breakpoint die Programmausführung bei jeder Zeile, die ein ausführbares Statement enthält, unterbrechen soll.
Label		Bezieht sich auf ein zuvor im Sourcecode eines Objekts gesetztes Label für Statements, die Verarbeitungsschleifen definieren, siehe auch *Systemvariablen anzeigen* weiter oben. Gültige Werte: 1 bis 32 Zeichen.
Skips before execution	**Skips**	Legt fest, dass der Breakpoint erst dann ausgeführt werden soll, wenn die entsprechende Statement-Zeile eine bestimmte Anzahl von Malen ausgeführt wurde. Gültige Werte: 0 (Standardwert) bis 32767.
Max number executions	**Execs**	Ein beliebiger Wert größer als Null (0) legt die maximale Anzahl der Breakpoint-Ausführungen fest. Gültige Werte: 0 (Standardwert) bis 32767.
Number of activations	**Count**	Gibt an, wie viele Male ein Breakpoint für die betreffende Statement-Zeile aktiviert wurde. Der Zähler wird zurückgesetzt, wenn das Programm auf Level 1 gestartet wird.
Error in definition	**E**	Zeigt an, dass die Statement-Zeile in der Breakpoint-Definition während der Programmausführung nicht in dem katalogisierten Objekt gefunden werden kann. Dieser Fehler kann verursacht werden, wenn ein Objekt während des Debugging-Vorgangs geändert und neu katalogisiert wird.
Commands		Bis zu sechs Debug-Kommandos. Geben Sie jeweils ein Kommando pro Zeile ein. Eine Zusammenfassung der zur Verfügung stehenden Kommandos finden Sie im Abschnitt *Debug-Kommandoübersicht und -syntax*. **Vorsicht:** Wenn Sie beim Setzen eines Breakpoint das Kommando BREAK löschen und kein Kommando eingeben, das einen Dialog öffnet, hat der Debugger keine Möglichkeit, während der Programmunterbrechung die Steuerung zu übernehmen.

9 Watchpoint Maintenance (Verwaltung der Watchpoints)

Ein Watchpoint ist ein Haltepunkt, bei dem die Ausführung eines Natural-Objekts immer dann unterbrochen wird, wenn sich der Wert einer Variablen ändert.

Darüber hinaus können Sie die Unterbrechung, so wie im Abschnitt ***Watchpoint-Operatoren*** beschrieben, von einer Bedingung mit Bezug zu einem spezifischen Variablenwert abhängig machen (siehe auch *Watchpoint setzen* weiter unten).

Durch die Verwendung von Watchpoints können Sie unbeabsichtigte Änderungen von Variablen entdecken, die durch Fehler enthaltende Objekte verursacht werden.

Eine Variable gilt als geändert, wenn sich ihr aktueller Wert entweder von dem Wert unterscheidet, der beim letzten Auslösen des Watchpoints aufgezeichnet wurde, oder wenn er sich vom Anfangswert unterscheidet. Die vergleichende Validierung von Watchpoint-Werten ist auf eine Feldlänge von 253 Byte beschränkt. Bei großen Variablen, die die die maximale Länge überschreiten, werden nur die ersten 253 Bytes für den Vergleich herangezogen.

Die Definition eines Watchpoint erfolgt durch Angabe des Namens des Natural-Objekts und des Namens der betreffenden Variablen.

Der eindeutige Bezeichner für einen Watchpoint ist die vom Debugger vergebene Spy-Nummer.

Sobald ein Watchpoint festgelegt wurde, bleibt er für die gesamte Sitzung festgelegt, es sei denn, Sie löschen ihn.

Um die Funktion zur Verwaltung der Watchpoints aufzurufen:

- Geben Sie im Hauptmenü **Debug Main Menu** den Funktionscode `W` ein.

  Oder:

  Geben Sie folgendes Direktkommando ein:

  ```
  WM
  ```

  Das Menü **Watchpoint Maintenance** erscheint.

Dieses Kapitel beschreibt die im Menü **Watchpoint Maintenance** zur Verfügung stehenden Funktionen und die in einem Watchpoint-Bildschirm vorhandenen Felder und Spalten.

Test-Modus ON/OFF setzen

Siehe Abschnitt *Test-Modus ein- und ausschalten*.

Watchpoint aktivieren

Um den aktuellen Zustand angegebener Watchpoints auf aktiv zu setzen:

- Geben Sie im Menü **Watchpoint Maintenance** den Funktionscode `A`, einen Objektnamen und/oder einen Variablennamen ein.

  Oder:

  Benutzen Sie das Direktkommando `ACTIVATE`. Syntax-Beschreibung siehe Abschnitt *Debug-Kommandoübersicht und -syntax*.

Wenn Sie kein Objekt oder keine Variable angeben (oder den standardmäßig vorhandenen Stern (*) im Feld **Variable** lassen), werden *alle* Watchpoints aktiviert.

Watchpoint deaktivieren

Um den aktuellen Zustand angegebener Watchpoints auf inaktiv zu setzen:

- Geben Sie im Menü **Watchpoint Maintenance** den Funktionscode `B`, einen Objektnamen und/oder einen Variablennamen ein.

  Oder:

  Benutzen Sie das Direktkommando `DEACTIVATE`. Syntax-Beschreibung siehe Abschnitt *Debug-Kommandoübersicht und -syntax*.

Wenn Sie kein Objekt oder keine Variable angeben (oder den standardmäßig vorhandenen Stern (*) im Feld **Variable** lassen), werden *alle* Watchpoints deaktiviert.

Watchpoint löschen

Um einen Watchpoint zu löschen:

- Geben Sie im Menü **Watchpoint Maintenance** den Funktionscode `C`, einen Objektnamen und/oder einen Variablennamen ein.

  Oder:

  Benutzen Sie das Direktkommando `DELETE`. Syntax-Beschreibung siehe Abschnitt *Debug-Kommandoübersicht und -syntax*.

Wenn Sie kein Objekt oder keine Variable angeben (oder den standardmäßig vorhandenen Stern (*) im Feld **Variable** lassen), werden *alle* Watchpoints gelöscht.

Watchpoint anzeigen

Um einen Watchpoint anzuzeigen:

1 Geben Sie im Menü **Watchpoint Maintenance** den Funktionscode `D`, einen Objektnamen und/oder einen Variablennamen ein. Wenn Sie keinen Objektnamen eingeben, wird das **Standard-Objekt** (falls angegeben) verwendet.

  Oder:

  Benutzen Sie das Direktkommando `DISPLAY`. Syntax-Beschreibung siehe Abschnitt *Debug-Kommandoübersicht und -syntax*.

  Wenn für das angegebene Objekt und den Variablennamen ein Watchpoint gesetzt worden ist, erscheint der Bildschirm **Display Watchpoint** mit allen Watchpoint-Definitionen, zum Beispiel:

```
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 -

Commands ... BREAK









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

Die Felder im Bildschirm **Display Watchpoint** sind im Abschnitt *Felder und Spalten in Watchpoint-Bildschirmen* beschrieben.

Wird kein eindeutiger Watchpoint gefunden, dann erscheint der Bildschirm **List Watchpoints** (siehe unten).

2 Im Bildschirm **Display Watchpoint** können Sie sich die mit dem Watchpoint-Operator angegebene Bedingung für die Watchpoint-Aktivierung ansehen (siehe auch *Watchpoint-Operatoren*):

Drücken Sie PF10 (Alpha), um den Operator und/oder den Operandenwert in alphanumerischem Format anzuzeigen.

Oder:

Drücken Sie PF11 (Hex), um den Operator und/oder den Operandenwert in hexadezimalem Format anzuzeigen.

Sie können PF22 (Cmds) drücken, um zur Standardansicht des Bildschirms **Display Watchpoint** zurückzukehren. Dort befindet sich das Eingabefeld **Commands**.

Um Watchpoints aufzulisten:

- Geben Sie im Menü **Watchpoint Maintenance** den Funktionscode D, einen Objektnamen und/oder einen Variablennamen ein. Sie können Stern-Notation (*) benutzen, um einen Bereich

von Objektnamen und/oder Variablennamen anzugeben, zum Beispiel: `ABC*`. Wenn Sie nur einen Stern (*) eingeben, werden alle Namen ausgewählt. Wenn Sie keinen Objektnamen angeben, wird das **Standard-Objekt** (falls angegeben) benutzt.

Oder:

Benutzen Sie das Direktkommando `DISPLAY`. Syntax-Beschreibung siehe Abschnitt *Debug-Kommandoübersicht und -syntax*.

Der Bildschirm **List Watchpoints** erscheint (s. Beispiel). Er enthält eine Liste aller Watchpoints, die für das oder die angegebenen Objekte oder den Variablennamen gesetzt wurden:

```
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
```

Die Liste ist in aufsteigender Reihenfolge nach den **Spy-Nummern** in der Spalte **No.** sortiert.

Informationen zu den Spalten im Bildschirm **List Watchpoints** und zu den Zeilenkommandos, die bei einem Listeneintrag ausgeführt werden können, siehe *Felder und Spalten in Watchpoint-Bildschirmen*.

Watchpoint ändern

Um einen Watchpoint zu ändern:

1 Geben Sie im Menü **Watchpoint Maintenance** den Funktionscode `M`, einen Objektnamen und einen Variablennamen ein. Wenn Sie keinen Objektnamen eingeben, wird das **Standard-Objekt** (falls angegeben) verwendet.

Oder:

Benutzen Sie das Direktkommando `MODIFY`. Syntax-Beschreibung siehe Abschnitt *Debug-Kommandoübersicht und -syntax*.

Wenn ein eindeutiger Watchpoint angegeben wurde, erscheint der Bildschirm **Modify Watchpoint**, in dem Sie die Feldeinträge ändern können. Beschreibung der Felder im Bildschirm **Modify Watchpoint** siehe *Felder und Spalten in Watchpoint-Bildschirmen*.

Wird kein eindeutiger Watchpoint gefunden, dann erscheint der Bildschirm **List Watchpoints** (siehe Abschnitt *Watchpoint anzeigen*).

2 Im Bildschirm **Modify Watchpoint** können Sie die mit dem Watchpoint-Operator angegebene Bedingung für die Aktivierung eines Watchpoint ändern (siehe auch *Watchpoint-Operatoren*):

Drücken Sie PF10 (Alpha), um den Operator und/oder den Operandenwert in alphanumerischem Format zu ändern.

Oder:

Drücken Sie PF11 (Hex), um den Operator und/oder den Operandenwert in hexadezimalem Format zu ändern.

Drücken Sie PF22 (Cmds), um zur Standardansicht des Bildschirms **Modify Watchpoint** zurückzukehren. Dort befindet sich das Eingabefeld **Commands**.

3 Wenn Sie die Bearbeitung der Watchpoint-Definitionen beendet haben, drücken Sie PF3 (Exit) oder PF5 (Save), um die Änderungen zu speichern.

Wenn Sie PF12 (Canc) drücken, wird die Bearbeitung abgebrochen und der Watchpoint bleibt unverändert.

Watchpoint setzen

Um einen Watchpoint für eine Sitzung hinzuzufügen:

- Geben Sie im Menü **Watchpoint Maintenance** den Funktionscode `S`, einen Objektnamen und einen Variablennamen ein.

  Oder:

  Benutzen Sie das Direktkommando `SET`. Syntax-Beschreibung siehe Abschnitt *Debug-Kommandoübersicht und -syntax*.

  Oder:

  Bevor Sie ein Natural-Objekt ausführen:

  - Rufen Sie den Bildschirm **List Object Source** auf (siehe *List Object Source (Objekt-Sourcecode anzeigen)*).
  - Positionieren Sie in der Spalte **Source** den Cursor auf einem Variablennamen und drücken Sie PF18 (Se Wp).

  Wenn Sie keinen Objektnamen, aber einen gültigen Variablennamen angeben, wird der Name des **Standard-Objekts** (siehe Abschnitt *Debugger starten*) benutzt.

  Falls kein Standard-Objekt angegeben ist, erscheint ein Auswahlfenster, in dem alle in der aktuellen Library enthaltenen Objekte angezeigt werden.

  Wenn die Namen des Objekts und der Variablen korrekt angegeben sind, wird der Watchpoint sofort gesetzt und eine entsprechende Informationsmeldung wird im Bildschirm angezeigt. Ein für eine dynamische Variable oder ein X-Array gesetzter Watchpoint wird nur während der Programmausführung validiert. Weitere Informationen siehe *Verwaltung und Validierung der Debug-Einträge*.

  Der Watchpoint erhält das Standard-Kommando (`BREAK`), sein Anfangszustand und sein aktueller Zustand werden auf aktiv gesetzt und es werden keine Ausführungseinschränkungen angegeben. Achtung: Wenn Sie beim Setzen eines Watchpoint das Standardkommando `BREAK` löschen und kein Kommando eingeben, das einen Dialog öffnet, hat der Debugger keine Möglichkeit, während der Programmunterbrechung die Steuerung zu übernehmen.

Dieses Kapitel behandelt folgende Themen:

- **Watchpoint-Operatoren**

Watchpoint-Operatoren

Sie können für die Aktivierung eines Watchpoint eine Bedingung angeben, indem Sie in einem Watchpoint-Verwaltungsbildschirm einen (Vergleichs-)Operator und (falls relevant) einen dazugehörigen Operanden eingeben.

Um Watchpoint-Operatoren anzugeben:

1 Drücken Sie im Bildschirm **Set Watchpoint** oder **Modify Watchpoint** des gewählten Watchpoint die Funktionstaste PF10 (Alpha), wenn Sie einen Operanden eines Operators in alphanumerischem Format angeben möchten.

Oder:

Drücken Sie im Bildschirm **Watchpoint setzen** oder **Modify Watchpoint** des gewählten Watchpoint die Funktionstaste PF11 (Hex), wenn Sie einen Operanden eines Operators in hexadezimalem Format angeben möchten.

In der unteren Bildschirmhälfte erscheinen zwei Eingabefelder.

2 Im linken Eingabefeld geben Sie einen der Watchpoint-Operatoren ein, die in der folgenden Tabelle aufgelistet sind.

Im rechten Eingabefeld geben Sie den Operandenwert ein, der mit der Variablen verglichen werden soll. Bei Watchpoints mit Operatoren, die für dynamische Variablen (alphanumerisch oder binär) angegeben sind, werden die Operandenwerte von links nach rechts verglichen. Weil die Feldlänge einer dynamischen Variablen variiert, können Sie bis zu 253 Bytes als Vergleichswert eingeben. Ist die aktuelle Länge der dynamischen Variablen kürzer als die maximale Vergleichswertlänge von 253 Bytes, dann erfolgt der Vergleich nur in der aktuellen Länge der dynamischen Variablen.

Operator	Erklärung
MOD	Änderung (*Modification*). Aktiviert den Watchpoint jedes Mal, wenn eine Änderung der Variablen erfolgt. Dies ist die Standardeinstellung.
EQ	Gleich (*Equal to*). Aktiviert den Watchpoint, wenn die Variable geändert worden ist und wenn der aktuelle Wert der Variablen gleich dem angegebenen Operatorwert ist.
NE	Ungleich (*Not equal to*). Aktiviert den Watchpoint, wenn die Variable geändert worden ist und wenn der aktuelle Wert der Variablen nicht gleich dem angegebenen Operatorwert ist.
GT	Größer (*Greater than*). Aktiviert den Watchpoint, wenn die Variable geändert worden ist und wenn der aktuelle Wert der Variablen größer als der angegebene Operatorwert ist.
GE	Größer gleich (*Greater than or equal to*).

Operator	Erklärung
	Aktiviert den Watchpoint, wenn die Variable geändert worden ist und wenn der aktuelle Wert der Variablen größer als oder gleich dem angegebenen Operatorwert ist.
LT	Kleiner (Less than). Aktiviert einen Watchpoint, wenn die Variable geändert worden ist und wenn der aktuelle Wert der Variablen kleiner als der angegebene Operatorwert ist.
LE	Kleiner gleich (*Less than or equal to*). Aktiviert einen Watchpoint, wenn die Variable geändert worden ist und wenn der aktuelle Wert der Variablen kleiner als oder gleich dem angebenen Operatorwert ist.
INV	Ungültiger Inhalt (*Invalid contents*). Aktiviert den Watchpoint jedes Mal, wenn der einer Variablen des Typs N, P, D oder T zugewiesene Wert nicht die folgenden Bedingungen erfüllt: N Numerisch, nicht gepackt (*Numeric unpacked*). P Numerisch, gepackt (*Packed numeric*). D Datumsbereich (*Date range*) von 1582-01-01 bis 2700-12-31. T Zeitbereich (*Time range*) von 1582-01-01 00:00:00.0 bis 2700-12-31 23:59:59.9.

Sie können PF22 (Cmds) drücken, um zur Standardansicht des Bildschirms to **Watchpoint setzen** bzw. **Modify Watchpoint** zurückzukehren. Dort befindet sich das Eingabefeld **Commands**.

3 Drücken Sie PF5 (Save), um die Operator-Definitionen zu speichern.

Oder:

Drücken Sie PF12 (Canc), um die Operator-Definitionen unverändert zu lassen und den Bildschirm **Modify Watchpoint** zu verlassen.

Felder und Spalten in Watchpoint-Bildschirmen

Die folgende Tabelle enthält die Beschreibungen der Felder in den Bildschirmen **Display Watchpoint** und **Modify Watchpoint** und der Spalten im Bildschirm **List Watchpoint**:

Feld	Spalte	Erläuterung
Test Mode		Gibt an, ob der Test-Modus auf ON oder OFF gesetzt ist.
Object		Zeigt den Namen des **Standard-Objekts** (siehe *Debugger starten*), falls angegeben.
	Co	Feld zur Eingabe eines der folgenden Zeilenkommandos: **Kommando Aktion** AC Watchpoint aktivieren

Feld	Spalte	Erläuterung
		DA Watchpoint deaktivieren DI Watchpoint anzeigen MO Watchpoint ändern DE Watchpoint löschen ? Gültige Zeilenkommandos auflisten . Watchpoint-Bildschirm verlassen
Spy number	**No.**	Eine eindeutige Nummer, die vom Debugger beim Setzen des Watchpoint zugewiesen wird.
Initial state	**Stat I**	Gibt den Anfangszustand (Initial) und den aktuellen Zustand (Current) des Watchpoint an: Aktiv (A) oder Inaktiv (I) an.
Current state	**Stat C**	
Watchpoint name	**WP Name**	Der Name des Watchpoint. Der Standard-Name für einen Watchpoint ist der Name der betroffenen Variablen. Gültige Werte: 1 bis 12 Zeichen. Namen, deren Länge die Feldgröße übersteigt, werden nach 12 Zeichen abgeschnitten. Im Bildschirm **List Watchpoints** wird der Name des Watchpoint in der ersten Zeile über dem Namen der Variablen aufgelistet.
DBID/FNR	**DBID**	Die Datenbankkennung (DBID) und die Dateinummer (FNR) der Systemdatei, in der das Natural-Objekt gespeichert ist.
	FNR	
Library	**Library**	Der Name der Library, die das Objekt enthält.
Object name	**Object**	Der Name des Objekts, das in der aktuellen Library oder in einer ihrer Steplibs verfügbar ist. Wenn Sie eine Systemvariable als Watchpoint angeben möchten, geben Sie im Feld **Object name** einen Stern (*) ein.
Variable name		Der Name einer Benutzervariablen, einer globalen Variablen oder einer Systemvariablen. Wenn die Variable Teil einer Gruppe ist, kann sie den Gruppennamen als Präfix haben. Wenn Sie eine Systemvariable angeben möchten, geben Sie im Feld **Object name** einen Stern (*) ein. Für ein Array muss eine Indexbeschreibung angegeben werden (Watchpoints können nur für einzelne Elemente definiert werden.) Im Bildschirm **List Watchpoints** wird der Variablenname in der zweiten Zeile, unterhalb dem Watchpoint-Namen aufgelistet. Weitere Informationen siehe *Variable Maintenance*.

Feld	Spalte	Erläuterung
Skips before execution	**Skips**	Legt fest, dass der Watchpoint erst ausgeführt wird, wenn die für den Watchpoint gesetzte Bedingung erfüllt ist (siehe auch **Watchpoint-Operatoren**). Gültige Werte: `0` (Standardwert) bis `32767`.
Max number executions	**Execs**	Ein beliebiger Wert größer als Null (`0`) legt die maximale Anzahl der Watchpoint-Ausführungen fest. Gültige Werte: `0` (Standardwert) bis `32767`.
Number of activations	**Count**	Gibt an, wie viele Male die Watchpoint-Bedingung entsprechend der Angabe beim **Watchpoint-Operator** erfüllt wurde. Der Zähler wird zurückgesetzt, wenn das Programm auf Level 1 gestartet wird.
Format/length		Das Natural-Datenformat und die Datenlänge der Variablen, z.B. A10.
Persistent	**P**	Kennzeichnet einen Watchpoint als persistent. Persistente Watchpoints sind nicht auf das Natural-Objekt beschränkt, für das sie definiert sind, sondern gelten zusätzlich für alle untergeordneten Programmebenen. Persistente Watchpoints sind nur sinnvoll bei Variablen, die `BY REFERENCE` und nicht `BY VALUE RESULT` an ein Subprogramm übergeben werden, siehe relevante Parameterbeschreibung des `CALLNAT`-Statement im Abschnitt *Parameter - operand2*, in der *Statements*-Dokumentation. **Einschränkung:** Persistente Watchpoints sind nicht erlaubt bei Variablen, die in einer Parameter-Kontextklausel definiert sind. Gültiger Wert: `Y` (Yes) oder `N` (No). `N` ist der Standardwert.
Act. level		Bezieht sich auf **Persistent**. Zeigt die Programmebene an, auf der ein persistenter Watchpoint automatisch aktiviert wurde.
Error in definition	**E**	Zeigt eine ungültige Watchpoint-Definition an. Dieser Fehler kann auftreten, wenn das ausführende Programm während des Debugging-Vorgangs neu katalogisiert wird, nachdem die betreffende Variablendefinition geändert wurde. Ein Watchpoint, der für eine dynamische Variable oder ein X-Array (*eXtensible array* = erweiterbares Array) gesetzt ist, wird nur während der Programmausführung auf Gültigkeit geprüft.
Commands		Bis zu sechs Debug-Kommandos. Geben Sie jeweils ein Kommando pro Zeile ein. Eine Zusammenfassung der zur Verfügung stehenden Kommandos finden Sie im Abschnitt ***Debug-Kommandoübersicht und -syntax***.

Feld	Spalte	Erläuterung
		Vorsicht: Wenn Sie das Kommando BREAK löschen und kein Kommando eingeben, das einen Dialog öffnet, hat der Debugger keine Möglichkeit, während der Programmunterbrechung die Kontrolle zu erhalten.

10 Call Statistics Maintenance (Statistiken über gerufene Objekte)

Mit dieser Funktion erhalten Sie Statistikinformationen darüber, welche Natural-Objekte während der Ausführung einer Anwendung gerufen wurden und wie oft ein Objekt gerufen wurde. Die Call-Statistikinformationen werden nach dem Zurücksetzen der Debug-Umgebung gelöscht.

Um die Funktion Call Statistics Maintenance aufzurufen:

- Geben Sie Im Menü **Debug Main Menu** den Funktionscode `C` ein.

  Oder:

  Geben Sie folgendes Direktkommando ein:

  ```
  CS
  ```

  Das Menü **Call Statistics Maintenance** wird angezeigt.

Die im Menü **Call Statistics Maintenance** zur Verfügung stehenden Funktionen werden in den folgenden Abschnitten erklärt. Alle zugehörigen Druckfunktionen sind jedoch im Abschnitt *Objekte drucken* beschrieben.

Test-Modus ON/OFF setzen

Siehe Abschnitt *Test-Modus ein- und ausschalten*.

Call-Statistik ON/OFF setzen

Wenn während der Ausführung eines Natural-Objekts die Funktion **Call Statistics** auf `ON` gesetzt ist, werden alle Calls, die an ein bestimmtes Objekt gehen, gezählt und die resultierenden Statistikinformationen können anschließend angezeigt oder gedruckt werden.

Um die Funktion Call Statistics auf ON oder OFF zu setzen:

- Geben Sie im Menü **Call Statistics Maintenance** den Funktionscode `C` ein, um alle Call-Statistiken zu aktivieren oder zu deaktivieren.

  Oder:

  Geben Sie folgendes Direktkommando ein:

  ```
  SET CALL ON
  ```

  oder

  ```
  SET CALL OFF
  ```

Anmerkung: Ist die die Funktion **Call Statistics** ausgeschaltet und wurde keine Call-Statistik erstellt oder wurden alle Call-Statistiken durch Zurücksetzen der Debug-Umgebung gelöscht, dann werden die für Statementsausführungsstatistik gespeicherten Informationen (siehe *Statement Execution Statistics Maintenance (Statistiken über ausgeführte Statement-Zeilen)*) für die Anzeige verwendet. Damit können Sie die während der Ausführung einer Natural-Anwendung nicht aufgerufenen Natural-Objekte auffinden.

Alle Objekte anzeigen

Diese Funktion liefert eine Übersicht über die Aufrufhäufigkeit (*Call Frequency*) aller in einer Library enthaltenen Objekte.

Um die Aufrufhäufigkeit (*Call Frequency*) aller in einer Library enthaltenen Objekte anzuzeigen:

- Geben Sie im Menü **Call Statistics Maintenance** den Funktionscode `1` und einen Library-Namen ein.

  Oder:

  Geben Sie folgendes Direktkommando ein:

  ```
  DISPLAY OBJECT library
  ```

  Siehe auch die Syntax des Direktkommandos `DISPLAY` in ***Debug-Kommandoübersicht und -syntax***.

  Wenn Sie keinen Library-Namen angeben, wird standardmäßig die Library angenommen, bei der Sie gerade angemeldet sind.

  Der Bildschirm **Display Called Objects** erscheint. Siehe Beispiel für **Display Called Objects** weiter unten.

  Der Bildschirm **Display Called Objects** enthält eine Auflistung aller Objekte in der angegebenen Library und zeigt in der Spalte **Calls** (ganz rechts) ihre Aufrufhäufigkeit. Für jedes Call-Statement, z.B. `FETCH` oder `CALLNAT` wird ein Eintrag mit dem Namen und einer Zählervariablen in den Debug-Pufferspeicher geschrieben. Der Zähler wird dann bei jedem Aufruf (Call) des entsprechenden Objekts erhöht.

Aufgerufene Objekte anzeigen

Der von dieser Funktion aufgerufene Bildschirm entspricht dem **Display Call Statistics**-Bildschirm, es werden jedoch nur diejenigen Objekte angezeigt, die aufgerufen worden sind.

Um aufgerufene Objekte einer Library anzuzeigen:

- Geben Sie im Menü **Call Statistics Maintenance** den Funktionscode `2` und einen Library-Namen ein.

  Oder:

  Geben Sie folgendes Direktkommando ein:

  ```
  DISPLAY CALL library
  ```

  Siehe auch die Syntax des Direktkommandos `DISPLAY` in ***Debug-Kommandoübersicht und -syntax***.

  Der Bildschirm **Display Called Objects** erscheint (Beispiel):

```
16:06:53              ***** NATURAL TEST UTILITIES *****             2002-02-15
Test mode ON               - Display Called Objects -
                                                                          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
```

Wenn Sie keinen Library-Namen angeben, wird standardmäßig die Library angenommen, bei der Sie gerade angemeldet sind.

Nicht aufgerufene Objekte anzeigen

Der von dieser Funktion aufgerufene Bildschirm entspricht dem **Display Call Statistics**-Bildschirm, es werden jedoch nur diejenigen Objekte angezeigt, die *nicht* aufgerufen worden sind.

Um nicht aufgerufene Objekte anzuzeigen:

- Geben Sie im Menü **Call Statistics Maintenance** den Funktionscode `3` und einen Library-Namen ein.

Oder:

Geben Sie folgendes Direktkommando ein:

```
DISPLAY NOCALL library
```

Siehe auch die Syntax des Direktkommandos DISPLAY in *Debug-Kommandoübersicht und -syntax*.

Wenn Sie keinen Library-Namen angeben, wird standardmäßig die Library angenommen, bei der Sie gerade angemeldet sind.

Siehe Beispiel-Bildschirm für **Display Called Objects**.

Objekte drucken

Mit den Druckfunktionen können Sie eine generierte Liste der Aufrufstatistiken direkt an einen Drucker weiterleiten oder die Liste auf einen PC herunterladen. Sie geben einen Drucker als Ausgabegerät im Bildschirm **User Profile** (Benutzerprofil) des Debuggers an. Benutzen Sie das Debugger-Kommando PROFILE (siehe Abschnitt *Kommandos zum Navigieren und Anzeigen von Informationen*), um diesen Bildschirm aufzurufen.

Wenn Sie keinen Library-Namen angeben, wird standardmäßig die Library angenommen, bei der Sie gerade angemeldet sind.

Wie unter *Druckoptionen* weiter unten angegeben, können Sie zum Aufrufen einer Druckfunktion entweder einen Funktionscode im Menü **Statement Execution Statistics Maintenance**, ein Zeilenkommando im Bildschirm **Display Statement Lines** oder ein Direktkommando an der Eingabeaufforderung eingeben.

Druckoptionen

Druckfunktion	Funktionscode	Direktkommando
All Objects (alle Objekte)	**4**	`PRINT OBJECT library`
Called Objects (gerufene Objekte)	**5**	`PRINT CALL library`
Non-Called Objects (nicht gerufene Objekte)	**6**	`PRINT NOCALL library`

Siehe auch die Syntax des Direktkommandos PRINT in *Debug-Kommandoübersicht und -syntax*.

Verwandtes Thema

- *Beispiel für das Erstellen und Drucken von Statistiken im Batch-Modus* im Abschnitt *Batch-Verarbeitung*

Beispiel für einen PC Download

Wenn in Ihrer Umgebung die Produkte Entire Connection und Natural Connection installiert sind, können Sie, wie nachfolgend beschrieben, eine Statistikliste auf einen PC herunterladen.

Um eine Liste auf einen PC herunterzuladen:

1 Beim Start der Sitzung: Geben Sie den Profilparameter `PRINT` folgendermaßen ein:

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

2 Nach dem Start der Sitzung: Benutzen Sie das folgende Terminalkommando, um die Verbindung zum PC zu aktivieren:

```
%+
```

3 Rufen Sie den Debugger auf und aktivieren Sie ihn.

4 Rufen Sie den Bildschirm **User Profile** auf, indem Sie das Debugger-Kommando `PROFILE` eingeben (siehe *Kommandos zum Navigieren und Anzeigen von Informationen*).

5 Ersetzen Sie im Bildschirm **User Profile** im Feld **Output device** den aktuellen Eintrag durch `PCPRNT01` und drücken Sie PF3 (Exit), um die Einstellungen zu speichern.

6 Aktivieren Sie die Funktion **Call Statistics** und führen Sie die Anwendung aus, für die der Debugger Statistikdaten sammeln soll.

7 Wählen Sie im Statistik-Bildschirm eine Druckfunktion aus.

In dem Entire Connection-Fenster, das dann erscheint, können Sie die Ausgabedatei und das PC-Verzeichnis angeben.

11 Statement Execution Statistics Maintenance (Statistiken über ausgeführte Statement-Zeilen)

Mit dieser Funktion erhalten Sie Statistikinformationen darüber, welche Statement-Zeilen aufgerufener Natural-Objekte ausgeführt wurden. Außerdem zeigt die Funktion, wie oft ein Objekt aufgerufen und wie oft eine Statement-Zeile ausgeführt wurde.

Statistikinformationen über die Ausführung von Statements können für folgende Zwecke verwendet werden:

- Aufspüren von „totem“ Programmcode (der nie ausgeführt wird) in einer Anwendung,
- Abschätzen des Abdeckungsgrads eines Anwendungstests (wie viele Statement-Zeilen sind nicht wenigstens einmal beim Testen ausgeführt worden),
- Orten von häufig ausgeführten Programmabschnitten, die Auswirkungen auf die Performance der Anwendung haben könnten.

Um die Funktion Statement Execution Statistics Maintenance aufzurufen:

- Geben Sie im Hauptmenü **Debug Main Menu** den Funktionscode `X` ein.

  Oder:

  Geben Sie folgendes Direktkommando ein:

  ```
  XS
  ```

  Das Menü **Statement Execution Statistics Maintenance** wird angezeigt.

Die im Menü **Statement Execution Statistics Maintenance** zur Verfügung stehenden Funktionen werden im folgenden Abschnitt erklärt. Alle zugehörigen Druckfunktionen sind jedoch im Abschnitt *Objekte drucken* beschrieben.

Test-Modus ON/OFF setzen

Siehe Abschnitt *Test-Modus ein- und ausschalten*.

Funktion Statement Execution Statistics auf ON/OFF/COUNT setzen

Mit dieser Funktion können Sie Statistiken über ausgeführte Statement-Zeilen von Natural-Objekten aktivieren.

Folgende Themen werden behandelt:

- Einstelloptionen

- Statement-Ausführungsstatistiken aktivieren und deaktivieren

Einstelloptionen

Wenn ein Natural-Objekt ausgeführt wird und die Funktion **Statement Execution Statistics** auf `ON` oder `COUNT` gesetzt ist, werden alle in einem bestimmten Objekt ausgeführten Statement-Zeilen in einem Statistikbericht aufgelistet.

- Wenn die Option `ON` gesetzt ist, hält der Debugger nur fest, ob eine spezifische Statement-Zeile ausgeführt wurde oder nicht.
- Wenn die Option `COUNT` gesetzt ist, zählt der Debugger, wie oft eine Statement-Zeile ausgeführt wurde

Sie können eine Library und einen Objektnamen angeben, um die Statement-Ausführungsstatistik auf die gewünschten Natural-Objekte einzuschränken. Standardmäßig werden Statistikdaten für alle Objekte in der aktuellen Library gesammelt. Sie können Stern-Notation (*) benutzen, um einen Bereich von Namen anzugeben.

Wenn Sie die Statement-Ausführungsstatistik von `ON` nach `COUNT` oder umgekehrt schalten, sind bereits vorhandene Statistiken davon nicht betroffen, d.h., ihr Status bleibt auf `ON` oder `COUNT`.

Die gesammelten Statistikdaten werden im Debug-Pufferspeicher gespeichert. Die Menge an Speicherplatz, die benötigt wird, um statistische Informationen für ein Objekt zu speichern, ist ungefähr:

(Anzahl der Sourcecode-Zeilen) / 8 + 100 Bytes, wenn **Statement Execution Statistics** auf `ON` gesetzt ist,
und
(Anzahl der Sourcecode-Zeilen) * 4 + 100 Bytes, wenn **Statement Execution Statistics** auf `COUNT` gesetzt ist.

Wenn Sie ein Natural-Objekt ändern, indem Sie Zeilen einfügen oder löschen und die Zeilen nicht neu nummerieren, bevor Sie es mit dem Kommando `STOW` speichern und katalogisieren, kann sich die für die Statistikdaten des Objekts benötigte Speichermenge erhöhen. Um dies zu vermeiden, können Sie in Ihrem Editor-Profil die Option **Auto Renumber** auf `Y` (Yes) setzen (siehe *Editor-Profil* in der *Editoren*-Dokumentation) oder das Systemkommando `CATALL` (siehe *Systemkommandos*-Dokumentation) benutzen, und zwar bei (standardmäßig) eingeschalteter Option **Renumber source-codes lines** (Sourcecode-Zeilen neu nummerieren).

Sie können das Debugger-Kommando `PROFILE` benutzen (siehe *Kommandos zum Navigieren und Anzeigen von Informationen*), um die Größe des Debug-Pufferspeichers einzuschränken. Wenn die Option **Statement Execution Statistics** auf `COUNT` gesetzt ist, werden bei Objekten mit mehr als 8000 Statement-Zeilen keine Statement-Ausführungsstatistiken erfasst.

Statement-Ausführungsstatistiken unterliegen als Teil der Debug-Umgebung den Auswirkungen der Direktkommandos `SAVE ENVIRONMENT` und `LOAD ENVIRONMENT` (siehe auch Abschnitt *Debug Environment Maintenance - Debug-Umgebung verwalten*).

Statement-Ausführungsstatistiken aktivieren und deaktivieren

Dieser Abschnitt beschreibt, wie Sie Statement-Ausführungsstatistiken aktivieren und deaktivieren.

Sie können eine Library und/oder einen Objektnamen angeben, um die Statement-Ausführungsstatistik auf die gewünschten Natural-Objekte einzuschränken. Standardmäßig werden Statistikdaten für alle Objekte in der aktuellen Library gesammelt. Sie können Stern-Notation (*) benutzen, um einen Bereich von Namen anzugeben.

Um Statement-Ausführungsstatistiken zu aktivieren:

- Geben Sie im Menü **Statement Execution Statistics Maintenance** den Funktionscode `S`, den Namen einer Library und/oder den Namen eines Objekts an. Ändern Sie den Wert im Feld **State** auf `ON`.

  Oder:

  Geben Sie eines der folgenden Direktkommandos ein:

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

  oder

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

  Siehe auch die Syntax des Direktkommandos `SET` in *Debug-Kommandoübersicht und -syntax*.

Wenn Sie keine Library und/oder ein Objekt angeben, werden die Statistikdaten zu allen Objekten in Ihrer aktuellen Library aktiviert.

Um Statement-Ausführungsstatistiken zu deaktivieren:

- Geben Sie im Menü **Statement Execution Statistics Maintenance** den Funktionscode `S`, den Namen einer Library und/oder den Namen eines Objekts an. Ändern Sie den Wert im Feld **State** auf `OFF`.

  Oder:

  Geben Sie folgendes Direktkommando ein:

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

  Siehe auch die Syntax des Direktkommandos `SET` in *Debug-Kommandoübersicht und -syntax*.

Wenn Sie keine Library und/oder ein Objekt angeben, werden die Statistikdaten zu allen Objekten in Ihrer aktuellen Library deaktiviert.

Statement-Ausführungsstatistiken löschen

Um Statement-Ausführungsstatistiken zu löschen:

- Geben Sie im Menü **Statement Execution Statistics Maintenance** den Funktionscode `C`, den Namen einer Library und/oder den Namen eines Objekts an.

  Oder:

  Geben Sie folgendes Direktkommando ein:

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

  Siehe auch die Syntax des Direktkommandos `DELETE` in ***Debug-Kommandoübersicht und -syntax***.

Wenn Sie keine Library und/oder ein Objekt angeben, werden die Statistikdaten zu allen Objekten in Ihrer aktuellen Library gelöscht.

Statement-Ausführungsstatistiken anzeigen

Mit dieser Funktion können Sie einen Bildschirm aufrufen, der eine Liste von Statement-Ausführungsstatistiken enthält.

Um Statement-Ausführungsstatistiken anzuzeigen:

1. Geben Sie im Menü **Statement Execution Statistics Maintenance** den Funktionscode `D`, den Namen einer Library und/oder den Namen eines Objekts an.

   Oder:

   Geben Sie folgendes Direktkommando ein:

   ```
   DISPLAY XSTATISTICS
   ```

   Der Bildschirm **List Statement Execution Statistics** wird angezeigt:

```
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
```

Zu jedem Objekt werden folgende Informationen angezeigt:

- Aufrufhäufigkeit (*Called n Times*),
- Anzahl der ausführbaren Statements (*Exec able*),
- Anzahl der ausgeführten Statements (*Executed*),
- Prozentsatz (%) der ausgeführten Statements bezogen auf die Gesamtzahl der ausführbaren Statements,
- Gesamtzahl der ausgeführten Statements (*Total No. Executions*).

Falls Daten fehlen oder möglicherweise inkonsistent sind, wird der betroffene Listeneintrag hervorgehoben dargestellt.

2 Sie können in der Liste einen Eintrag mit einem Zeilenkommando zur Weiterverarbeitung markieren:

Zeilenkommando	Erklärung
DE	Statement-Ausführungsstatistik löschen, s. **oben**.
DS	**Alle Statement-Zeilen anzeigen**.
DX	**Nur die ausgeführten Statement-Zeilen anzeigen**.
DN	**Nur die nicht ausgeführten Statement-Zeilen anzeigen**.
I	Informationen zum katalogisierten Objekt und zu Fehlern anzeigen.
PS	Alle Statement-Zeilen drucken.
PX	Nur die ausgeführten Statement-Zeilen drucken.
PN	Nur die nicht ausgeführten Statement-Zeilen drucken.

Weitere Informationen zu Druckfunktionen siehe ***Statements drucken***.

Im folgenden Abschnitt werden die Bildschirme beschrieben, die mit den Anzeigekommandos aufgerufen werden können:

- Alle Statement-Zeilen anzeigen
- Nur die ausgeführten Statement-Zeilen anzeigen

- Nur die nicht ausgeführten Statement-Zeilen anzeigen

Alle Statement-Zeilen anzeigen

Der Bildschirm **Display Statement Lines** zeigt den Objekt-Sourcecode und gibt an, ob eine Statement-Zeile ausgeführt worden ist oder nicht.

Um den Bildschirm Display Statement Lines anzuzeigen:

- Markieren Sie im Bildschirm **List Statement Execution Statistics** den gewünschten Eintrag mit dem Zeilenkommando `DS`.

  Oder:

  Geben Sie folgendes Direktkommando ein:

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

  Siehe auch die Syntax des Direktkommandos `DISPLAY` in ***Debug-Kommandoübersicht und -syntax***.

  Der Bildschirm **Display Statement Lines** erscheint. Wenn die Funktion **Statement Execution Statistics** auf `COUNT` gesetzt worden ist, wird die Ausführungshäufigkeit der Statements-Zeile angezeigt. Beispiel:

```
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
```

Falls kein eindeutiges Objekt angegeben wurde, erscheint der Bildschirm **List Statement Execution Statistics**.

Nur die ausgeführten Statement-Zeilen anzeigen

Der Bildschirm **Display Executed Statement Lines** entspricht dem Bildschirm **Display Statement Lines**, aber es werden nur die Statement-Zeilen angezeigt, die ausgeführt worden sind.

Um den Bildschirm Display Executed Statement Lines aufzurufen:

- Markieren Sie im Bildschirm **List Statement Execution Statistics** den gewünschten Eintrag mit dem Zeilenkommando `DX`.

  Oder:

  Geben Sie folgendes Direktkommando ein:

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

  Siehe auch die Syntax des Direktkommandos `DISPLAY` in *Debug-Kommandoübersicht und -syntax*.

  Falls kein eindeutiges Objekt angegeben wurde, erscheint der Bildschirm **List Statement Execution Statistics**.

Nur die nicht ausgeführten Statement-Zeilen anzeigen

Der Bildschirm **Non-Executed Statement Lines** entspricht dem Bildschirm **Display Statement Lines**, aber es werden nur die Statement-Zeilen angezeigt, die nicht ausgeführt worden sind.

Um den Bildschirm Display Non-Executed Statement Lines aufzurufen:

- Markieren Sie im Bildschirm **List Statement Execution Statistics** den gewünschten Eintrag mit dem Zeilenkommando `DN`.

  Oder:

  Geben Sie folgendes Direktkommando ein:

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

  Siehe auch die Syntax des Direktkommandos `DISPLAY` in *Debug-Kommandoübersicht und -syntax*.

  Falls kein eindeutiges Objekt angegeben wurde, erscheint der Bildschirm **List Statement Execution Statistics**.

Statements drucken

Mit den Druckfunktionen können Sie eine generierte Liste der Statement-Ausführungsstatistiken direkt an einen Drucker weiterleiten oder die Liste auf einen PC herunterladen. Sie geben einen Drucker als Ausgabegerät im Bildschirm **User Profile** (Benutzerprofil) des Debuggers an. Benutzen Sie das Debugger-Kommando `PROFILE` (siehe Abschnitt *Kommandos zum Navigieren und Anzeigen von Informationen*), um diesen Bildschirm aufzurufen.

Wenn Sie keinen Library-Namen angeben, wird standardmäßig die Library angenommen, bei der Sie gerade angemeldet sind.

Wie unter *Druckoptionen* weiter unten angegeben, können Sie zum Aufrufen einer der Druckfunktionen entweder einen Funktionscode im Menü **Statement Execution Statistics Maintenance**, ein Zeilenkommando im Bildschirm **Display Statement Lines** oder ein Direktkommando eingeben.

Druckoptionen

Druckfunktion	Funktionscode	Zeilenkommando	Direktkommando
Statement-Ausführungsstatistik drucken	**1**		`PRINT XSTATISTICS library (object)`
Alle Statements drucken	**2**	`PS`	`PRINT STATEMENT library (object)`
Ausgeführte Statements drucken	**3**	`PX`	`PRINT EXEC library (object)`
Nicht ausgeführte Statements drucken	**4**	`PN`	`PRINT NOEXEC library (object)`

Siehe auch die Syntax des Direktkommandos `PRINT` im Abschnitt ***Debug-Kommandoübersicht und -syntax***.

Verwandte Themen:

- ***Beispiel für einen PC Download*** in *Objekte drucken* im Abschnitt *Call Statistics Maintenance (Statistiken über gerufene Objekte)*
- ***Beispiel für das Erstellen und Drucken von Statistiken im Batch-Modus*** im Abschnitt *Batch-Verarbeitung*

12 Variable Maintenance (Anzeigen und Ändern von Variablen)

Mit dieser Funktion können Sie innerhalb des Debuggers Variablen anzeigen und ändern, wenn ein Natural-Objekt unterbrochen wurde.

Die Funktion **Variable maintenance** zeigt für das unterbrochene Natural-Objekt Benutzervariable, globale Variablen und die datenbankbezogenen Systemvariablen `*COUNTER`, `*ISN` und `*NUMBER` mit Natural-Datenformaten, Längen und Inhalten an.

Benutzervariable, globale Variablen und datenbankbezogene Systemvariablen anzeigen

Dieser Abschnitt beschreibt, wie Sie entweder den Bildschirm **Display Variables** (Übersicht) mit einer Liste aller Variablen oder den Bildschirm **Display Variable** (einzeln) mit allen Einzelheiten zu einer bestimmten Variablen aufrufen.

Variablen anzeigen - Übersicht

Um eine Übersicht über Benutzervariablen, globale Variablen und datenbankbezogene Systemvariablen anzuzeigen:

- Geben Sie im Hauptmenü **Debug Main Menu** oder im Fenster **Debug Break** den Funktionscode `V` ein.

  Oder:

  Geben Sie folgendes Direktkommando ein:

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

Der Bildschirm **Display Variables** (Übersicht) erscheint mit einer Liste der Variablen, die für das unterbrochene Natural-Objekt angegeben sind. Lange Werte können im Bildschirm abgeschnitten angezeigt werden. Bei Arrays wird nur der Inhalt der ersten Ausprägung angezeigt.

Sie können mit den Funktionstasten PF10 (Alpha) und PF11 (Hex) zwischen der alphanumerischen und der hexadezimalen Darstellung des Variableninhalts wechseln.

Mit PF5 (Zoom) können Sie zwischen der verkürzten Anzeige einer Variablen und der vollständigen Namensanzeige mit Gruppenname, Variablenname und Indizes (falls relevant) umschalten.

Für *variable* können Sie auch eine Systemvariable angeben. Weitere Informationen siehe ***Systemvariablen anzeigen***.

Variable anzeigen - einzeln

Um eine einzelne Variable vollständig anzuzeigen:

- Wählen Sie im Bildschirm **Display Variables** (Übersicht) eine Variable aus und markieren Sie sie mit dem Zeilenkommando `DI`.

  Oder:

  Geben Sie folgendes Direktkommando ein:

  ```
  DISPLAY VARIABLE variable
  ```

  Oder:

  Positionieren Sie im Bildschirm **List Object Source** in der Spalte **Source** den Cursor auf einem Variablennamen und drücken Sie PF18 (Di Va).

  - Bei der Benutzung von PF18 (Di Va) gelten die folgenden Einschränkungen:

    Wenn sich ein Variablenname (einschließlich der Ausprägungen eines Arrays) über mehr als eine Zeile erstreckt, wird nur der Inhalt der ersten Zeile ausgewertet.
    Wenn auf einen Array-Namen kein Index folgt, wird das gesamte Array angezeigt.
    Wenn der Inhalt eines Arrays konstant ist, z.B. Array (3,2,6), wird nur diese Ausprägung angezeigt.
    Wenn der Inhalt eines Arrays variabel ist, z.B. Array (i,j) oder Array (3:i), werden die Variablen ausgewertet, bevor die jeweiligen Ausprägungen des Arrays angezeigt werden.

  Oder:

  Positionieren Sie im Bildschirm **List Object Source** in der **Source** den Cursor auf einem Variablennamen und drücken Sie ENTER.

  - Bei der Benutzung von ENTER gelten die gleichen Einschränkungen wie bei PF18, siehe oben. Allerdings wird die Variable oder die Variablen-Ausprägung in einem Fenster **Display Variable** statt über den Bildschirm **Display Variable** (einzeln) angezeigt, wenn der Index für ein Array nicht mehr als eine Ausprägung angibt. Wenn der Index für ein Array mehr als eine Ausprägung angibt, werden die Daten über den Bildschirm **Display Variable** (einzeln) angezeigt.

  Oder:

  Anstatt den Cursor manuell zu positionieren und ENTER zu drücken, können Sie zur einfacheren Bedienung auch Entire Connection benutzen. Hier positioniert ein Doppelklick mit der linken Maustaste den Cursor und simuliert die ENTER-Taste.

Es erscheint der Bildschirm **Display Variable** (einzeln) bzw. ein Fenster mit allen relevanten Angaben zur jeweiligen Variablen.

Wenn Daten über ein Fenster angezeigt werden und die Länge des Variableninhalts 256 Bytes überschreitet, werden nur die ersten 256 Bytes angezeigt. Beim Bildschirm **Display Variable** (einzeln) gibt es keine solchen Einschränkungen und Sie können, wie nachfolgend beschrieben, durch den gesamten Inhalt der Variablen navigieren.

Um den gesamten Inhalt der Variablen anzuzeigen oder im Inhalt zu navigieren:

- Mit PF22 können Sie vorwärts, mit PF23 rückwärts navigieren.

  Oder:

  Geben Sie im Feld **Position** einen numerischen Wert ein, damit die Anzeige an einer bestimmten Stelle beginnt.

Mit PF10 (Alpha) und PF11 (Hex) können Sie zwischen alphanumerischer und hexadezimaler Darstellung des Variableninhalts wechseln.

Um alle Ausprägungen eines Arrays mittels Bildschirmfunktionen anzuzeigen:

- Wählen Sie im Bildschirm **Display Variables** eine Variable aus und markieren Sie sie mit dem Zeilenkommando `DI`.

  Oder:

  Mit PF7 (-) und PF8 (+) können Sie zwischen den einzelnen Ausprägungen hin und her wechseln.

Um eine oder mehrere Ausprägungen eines Arrays mittels Direktkommando anzuzeigen:

- Benutzen Sie folgendes Direktkommando:

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

  Dabei ist *variable-name* der Name der Variablen und *index-specification* eines der Folgenden: eine Index-Notation, ein Index-Bereich oder ein Stern (*) für alle Ausprägungen einer Dimension. Variablen, die Teil einer Indexangabe (*index-specification*) sind, werden ausgewertet, bevor die jeweiligen Ausprägungen angezeigt werden.

  Beispiele:

`DISPLAY VARIABLE ARRAY1(*)`	Eindimensionales Array: Zeigt alle Ausprägungen des eindimensionalen Arrays `ARRAY1` an.
`DISPLAY VARIABLE ARRAY1(1)` or	Eindimensionales Array: Zeigt die erste Ausprägung des eindimensionalen Arrays `ARRAY1` an.

`DISPLAY VARIABLE ARRAY1`	
`DISPLAY VARIABLE ARRAY2(2,3:4)`	Zweidimensionales Array: Zeigt die zweite Ausprägung der ersten Dimension und die Index-Notation der zweiten Dimension des zweidimensionalen Arrays `ARRAY2` an.
`DISPLAY VARIABLE ARRAY3(1,3:4,*)`	Dreidimensionales Array: Zeigt die erste Ausprägung der ersten Dimension, die Index-Notation der zweiten Dimension und alle Ausprägungen der dritten Dimension des dreidimensionalen Arrays `ARRAY3` an.
`DISPLAY VARIABLE ARRAY4(I,J + 1)`	Zweidimensionales Array: Zeigt die Ausprägung des zweidimensionalen Arrays `ARRAY4` an, angegeben durch den Wert des Ausdrucks `J + 1`.

Systemvariablen anzeigen

Um Systemvariablen anzuzeigen (außer datenbankbezogenen Systemvariablen):

- Geben Sie folgendes Direktkommando ein:

  ```
  SYSVARS
  ```

  Der Bildschirm **System Variables** erscheint mit einem eingeschränkten Satz Systemvariablen.

Um eine einzelne Systemvariable anzuzeigen:

- Benutzen Sie folgendes Direktkommando:

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

  Dabei ist *system-variable-name* der Name der Systemvariablen, der auch mit dem Direktkommando `SYSVARS` angezeigt werden kann.

Bei Variablen des Typs Handle wird der Name der Class der Instanz, auf die sich das Handle bezieht, in alphanumerischer Darstellung angezeigt. Wenn der Name der Class nicht verfügbar ist, wird stattdessen der Globally Unique Identifier (GUID) angezeigt. Wenn die Class innerhalb von Natural definiert wurde, wird der Name der Class oder die GUID mit dem Suffix `(NAT)` versehen.

Der Inhalt von Properties (Eigenschaften) einer Instanz kann im Debugger nicht angezeigt werden.

Variable ändern

Diese Funktion gilt nicht bei Systemvariablen.

Mit dieser Funktion können Sie den Wert von Benutzervariablen, globalen Variablen und datenbankbezogenen Variablen ändern.

Um den Inhalt einer Variablen vom Bildschirm Modify Variable aus zu ändern:

1 Rufen Sie den Bildschirm **Modify Variable** auf, indem Sie die betreffende Variable mit dem Zeilenkommando `MO` markieren.

Oder:

Drücken Sie im Bildschirm **Display Variable** die Taste PF5 (Mod).

2 Ändern Sie im Bildschirm **Modify Variable** im Feld **Contents** den Wert der Variablen.

Der neue Inhalt muss in Bezug auf das Natural-Datenformat der geänderten Variablen gültig sein, weil innerhalb des Debuggers keine Formatänderung bei einer Variablen vorgenommen werden kann.

Sie können im Bildschirm **Modify Variable** mit PF10 (Alpha) und PF11 (Hex) zwischen alphanumerischer und hexadezimaler Darstellung des Variablenwerts umschalten.

Um den Inhalt einer Variablen mittels Direktkommando zu ändern:

- Geben Sie folgendes Direktkommando ein:

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

Eine Meldung erscheint, die die Änderung des Variablenwerts bestätigt.

Anmerkung: Die Benutzung der Funktion **Modify Variables** oder des Kommandos `MODIFY VARIABLE` kann durch Natural Security unterbunden werden. Siehe Abschnitt *Components of an Environment Profile* in der *Natural Security*-Dokumentation.

13 List Object Source (Objekt-Sourcecode anzeigen)

Mit der Funktion **List object source** können Sie den Sourcecode eines Objekts anzeigen und Breakpoints verwalten. Dazu muss sich die entsprechende Source in Ihrer aktuellen Library oder in einer ihrer Steplibs befinden.

Um den Objekt-Sourcecode eines Objekts anzuzeigen:

- Geben Sie im Hauptmenü **Debug Main Menu** den Funktionscode `L` und einen Objektnamen ein.

  Oder:

  Geben Sie folgendes Direktkommando ein:

  ```
  LIST object
  ```

  Siehe auch die Syntax des Direktkommandos `LIST` in *Debug-Kommandoübersicht und -syntax*.

  Der Bildschirm **List Object Source** erscheint und der Sourcecode des Objekts wird angezeigt, wobei rechts im Bildschirm in der Spalte **Message** alle Breakpoints aufgelistet werden.

  Mit PF7 (-) oder PF8 (+) können Sie seitenweise nach oben oder unten blättern.

Wenn Sie ein Natural-Objekt ausführen, unterbricht der Debugger die Ausführung bei jedem von Ihnen gesetzten Breakpoint oder Watchpoint und das Fenster **Debug Break** erscheint (siehe *Debug Break-Fenster* in *Debugger-Konzept*).

Um den Sourcecode eines unterbrochenen Natural-Objekts aufzulisten:

- Wählen Sie im Fenster **Debug Break** den Funktionscode `L` für **List break**.

  Oder:

  Falls relevant: Drücken Sie auf einem Debugger-Bildschirm PF9 (Li Br) oder geben Sie folgendes Direktkommando ein:

  ```
  LIST BREAK
  ```

  Der Bildschirm **List Object Source** erscheint. Darin wird der Sourcecode an der Position angezeigt, an der eine Unterbrechung (Breakpoint oder Watchpoint) auftrat. Rechts im Bildschirm in der der Spalte **Message** wird der Name des Breakpoint oder Watchpoint angezeigt. Der entsprechende Sourcecode wird hervorgehoben dargestellt.

Breakpoints verwalten - Maintain Breakpoints

Sie können die Funktion **List object source** benutzen, um innerhalb des Objekt-Sourcecode Funktionen zum Verwalten oder direkten Ausführen von Breakpoint-Verwaltungsfunktionen aufzurufen oder direkt auszuführen. Eine Anleitung zum Setzen von Breakpoints und allgemeine Informationen zu Breakpoints finden Sie im Abschnitt ***Systemvariablen anzeigen*** in *Breakpoint Maintenance (Verwaltung der Breakpoints)*.

Um eine Breakpoint-Verwaltungsfunktion aus einer Objekt-Source heraus aufzurufen:

1 Geben Sie im Hauptmenü **Debug Main Menu** den Funktionscode `L` und einen Objektnamen ein.

Oder:

Geben Sie folgendes Direktkommando ein:

```
LIST object
```

Siehe auch die Syntax des Direktkommandos `LIST` in ***Debug-Kommandoübersicht und -syntax***.

Der Sourcecode des angegebenen Objekts wird angezeigt.

Rechts im Bildschirm in der der Spalte **Message** werden Namen von bereits gesetzten Breakpoints angezeigt.

- Zum Navigieren in der Source-Liste können Sie eines der folgenden Kommandos in der Kommandozeile eingeben:

  + (Pluszeichen) oder - (Minuszeichen), um seitenweise nach unten bzw. oben zu blättern.

  `TOP`, um an den Anfang zu blättern.

  `BOTTOM`, um an das Ende zu blättern.

  `LEFT`, um nach Links zu blättern.

  `RIGHT`, um nach Rechts zu blättern.

2 Markieren Sie in der Objekt-Source eine oder mehrere Zeilen mit einem der folgenden Kommandos:

Zeilenkommando	Erklärung
AC	Aktiviert Breakpoints.
DA	Deaktiviert Breakpoints.
DE	Löscht Breakpoints.
DI	Zeigt Breakpoints an.
MO	Verzweigt in den Verwaltungsbildschirm **Modify Breakpoint**.
SE	Setzt Breakpoints.
SM	Verzweigt in den Verwaltungsbildschirm **Set Breakpoint**.

Nach erfolgreicher Kommandoausführung wird in der Spalte **Message** rechts im Bildschirm eine entsprechende Meldung angezeigt.

14 Fehlerbehandlung

Dieser Abschnitt enthält Informationen zur Behandlung von Fehlern, die bei der Benutzung des Debuggers auftreten können.

Fehler während der Anwendungsausführung

Sie können den Debugger benutzen, um einen Natural-Systemfehler zu analysieren, der die Programmausführung unterbricht. Bei Test-Modus auf `ON` (siehe ***Test-Modus ein- und ausschalten***) oder Natural-Profilparameter `DBGERR` auf `ON` (siehe *DBGERR - Automatischer Debugger-Start bei Laufzeitfehler* in der *Parameter-Referenz*-Dokumentation) übernimmt der Debugger die Steuerung, wenn ein Fehler auftritt. In diesem Fall erscheint ein **Debug Break**-Fenster, zum Beispiel:

```
+------------------- 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   !
+---------------------------------------------------+
```

Sie können die Funktion **List break** benutzen, um den Sourcecode des Programms an der Stelle anzuzeigen, an der das letzte Statement ausgeführt wurde. Die Natural-Fehlernummer wird rechts im Bildschirm in der Spalte **Message** angezeigt und die entsprechende Sourcecode-Zeile ist hervorgehoben.

Sie können dann beispielsweise den Inhalt der Variablen in dem Programm untersuchen, um den Grund für den Fehler zu bestimmen.

Fehler während der Debugger-Ausführung

Falls während der Debugger-Ausführung in einer Anwendung (nachfolgend: „Debugging“) ein Fehler festgestellt wird, beendet der Debugger seine Arbeit und ruft ein Fenster mit einer Fehlermeldung ähnlich der im folgenden Beispiel gezeigten auf:

```
+------------------ 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   !
+------------------------------------------------------------------+
```

Wenn Sie diese Fehlermeldung durch Eingabe von `N` (No - dies ist die Standardeinstellung) bestätigen, geschieht Folgendes:

- Der Debugger beendet das Debugging und schaltet den Test-Modus aus (`OFF`).
- Das Natural-Laufzeitsystem ignoriert den Fehler und setzt die Ausführung der Anwendung fort.

Wenn Sie die Fehlermeldung durch Eingabe von `Y` (Yes) bestätigen, geschieht Folgendes:

- Der Debugger beendet das Debugging und schaltet den Test-Modus aus (`OFF`).
- Das Natural-Laufzeitsystem reagiert auf den Fehler und übergibt ihn an die Anwendung:

  Wenn ein `ON ERROR`-Statement (siehe *Statements*-Dokumentation) verwendet wird, bestimmt die Anwendung das weitere Vorgehen, nachdem ein Fehler zur Ausführungszeit aufgetreten ist. Zum Beispiel, im Fall eines Fehlers NAT3009, bei dem eine Transaktion aus der Datenbank zurückgesichert wird (*Back out)*, kann die Anwendung entsprechende Maßnahmen ergreifen.

  Falls kein `ON ERROR`-Statement verwendet wird, beendet das Natural-Laufzeitsystem die Ausführung der Anwendung und kehrt zur Anzeige einer Natural-Eingabeaufforderung zurück.

15 Kommandos zur Ausführungssteuerung

Dieser Abschnitt beschreibt die Direktkommandos, die im Debugger für die Steuerung des Programmablaufs während einer Debug-Sitzung zur Verfügung stehen. Eine Zusammenfassung aller im Debugger verfügbaren Kommandos finden Sie im Abschnitt ***Debug-Kommandoübersicht und -syntax***.

Die im Folgenden aufgeführten Kommandos gelten nur, wenn der Debugger die Programmausführung unterbricht.

ESCAPE BOTTOM

Dieses Kommando kann nur dann benutzt werden, wenn ein Natural-Objekt innerhalb einer Verarbeitungsschleife unterbrochen worden ist.

Wenn Sie dieses Kommando eingeben, wird das unterbrochene Natural-Objekt mit dem ersten Statement, das auf die Verarbeitungsschleife folgt, fortgesetzt.

Anmerkung: Natural Security kann die Ausführung dieses Kommandos unterbinden, siehe *Components of an Environment Profile* in der *Natural Security*-Dokumentation.

ESCAPE ROUTINE

Wenn Sie dieses Kommando eingeben, wird die Verarbeitung des unterbrochenen Natural-Objekts gestoppt und die Verarbeitung wird ab dem Objekt fortgesetzt, von dem aus das unterbrochene Natural-Objekt aufgerufen wurde. Sie wird mit dem Statement fortgesetzt, das auf das entsprechende `CALLNAT`-, `PERFORM`- oder `FETCH RETURN`-Statement folgt.

Wenn Sie das Kommando `ESCAPE ROUTINE` bei einem Hauptprogramm anwenden, beendet Natural das Programm und kehrt zum Kommandoeingabemodus zurück.

Anmerkung: Natural Security kann die Ausführung dieses Kommandos unterbinden, siehe *Components of an Environment Profile* in der *Natural Security*-Dokumentation.

EXIT

Wenn Sie aus dem Hauptmenü **Debug Main Menu** heraus die Exit-Funktion aufrufen möchten, können Sie PF3 (Exit) drücken oder das Ausführungssteuerungskommando `EXIT` eingeben. Der Debugger kehrt dann an eine der folgenden Stellen zurück:

- entweder zum aufrufenden Programm (d.h. zum unterbrochenen Natural-Objekt, das dann fortgesetzt wird),

- oder zu einer Kommandoeingabeaufforderung, wenn der Debugger mit dem Direktkommando TEST aufgerufen worden ist,
- oder zu dem entsprechenden Eingabefeld, wenn der Debugger mit dem Terminalkommando %<TEST aufgerufen worden ist.

Wenn aber zurzeit ein Breakpoint oder Watchpoint aktiv ist, wird das nächste Kommando dieses Breakpoint oder Watchpoint ausgeführt.

Wenn Sie sich jedoch nicht im Hauptmenü **Debug Main Menu** befinden und das Direktkommando EXIT eingeben oder PF3 (Exit) drücken, dann verlassen Sie die aktuelle Funktion und kehren zum vorigen Schritt Ihrer Debugging-Sitzung zurück.

GO

Wenn Sie das Direktkommando GO eingeben (oder PF14 drücken), gibt der Debugger die Steuerung an die Ausführung des unterbrochenen Natural-Objekts zurück. Wenn zum Zeitpunkt, als das Natural-Objekt unterbrochen wurde, ein Breakpoint oder Watchpoint aktiv war, werden die verbleibenden Kommandos dieses Breakpoint oder Watchpoint *nicht* ausgeführt.

NEXT

Wenn Sie das Direktkommando NEXT eingeben (oder PF13 drücken), wird das nächste für einen Breakpoint oder Watchpoint angegebene Kommando ausgeführt. Falls kein weiteres Kommando angegeben ist, wird die Programmausführung fortgesetzt.

RUN

Wenn Sie das Direktkommando RUN eingeben, wird der Test-Modus ausgeschaltet und die Programmausführung fortgesetzt, ohne dass weitere Breakpoints oder Watchpoints untersucht werden.

STEP

Wenn Sie das Direktkommando `STEP` eingeben, wird ein unterbrochenes Natural-Objekt um *n* ausführbare Statements fortgesetzt. Der Standardwert für *n* ist 1.

STEP SKIPSUBLEVEL

Wenn Sie das Direktkommando `STEP SKIPSUBLEVEL` bei einem Statement eingeben, das ein anderes Objekt aufruft (zum Beispiel, `CALLNAT`), erfolgt die Fortsetzung der Verarbeitung mit dem nächsten ausführbaren Statement im aktuellen Objekt, anstatt mit dem ersten ausgeführten Statement im aufgerufenen Objekt.

Wenn Sie dieses Direktkommando bei einem Statement anwenden, das kein anderes Objekt aufruft, reagiert der Debugger so, als ob Sie das Direktkommando `STEP` eingegeben hätten.

STEP SKIPSUBLEVEL n

Mit dem Direktkommando `STEP SKIPSUBLEVEL` können Sie eine übergeordnete Level-Nummer *n* angeben. Der Step-Modus wird dann beim nächsten Objekt auf der angegebenen Ebene fortgesetzt. Beispiel: Wenn Sie `STEP SKIPSUBLEVEL 2` in einem Objekt auf Ebene 4 eingeben, dann setzen Sie den Step-Modus in dem Objekt auf Ebene 2 fort.

Informationen zur Ebene eines Objekts können Sie mit dem Kommando `OBJCHAIN` aufrufen. Siehe Abschnitt *Kommandos zum Navigieren und Anzeigen von Informationen*.

STOP

Wenn Sie das Direktkommando `STOP` eingeben, werden sowohl der Debugger als auch ein unterbrochenes Natural-Objekt beendet.

Anmerkung: Natural Security kann die Ausführung dieses Kommandos unterbinden, siehe *Components of an Environment Profile* in der *Natural Security*-Dokumentation.

16 Kommandos zum Navigieren und Anzeigen von Informationen

Dieser Abschnitt beschreibt die Direktkommandos, die der Debugger zum Navigieren durch die Debugging-Bereiche, zum Blättern in den Bildschirmanzeigen, zum Abrufen verschiedener Informationen über Objekte und Variablen und zum Angeben von Profilen bereitstellt.

Eine Übersicht über alle mit dem Debugger verfügbaren Kommandos finden Sie unter *Debug-Kommandoübersicht und -syntax*.

BREAK

Das Kommando BREAK ist das Standard-Kommando, das automatisch beim Anlegen eines neuen Debug-Eintrags gesetzt wird. Es bewirkt die Anzeige des **Debug Break**-Fensters. Beschreibung siehe *Debug Break-Fenster* im Abschnitt *Debugger-Konzept*.

Wenn beim Ändern eines Debug-Eintrags das Kommando BREAK gelöscht wird, erscheint kein **Debug Break**-Fenster. Es werden jedoch sonstige angegebene Kommandos ausgeführt und der Ereigniszähler wird erhöht.

FLIP

Das Kommando FLIP schaltet in Bildschirmen die Anzeige der beiden PF-Tastenzeilen um: (PF1 bis PF12 und PF13 bis PF24).

LAST

Das Kommando LAST zeigt das zuletzt eingegebene Kommando an. Die letzten drei Kommandos werden gespeichert und können angezeigt und erneut ausgeführt werden.

OBJCHAIN

Das Kommando OBJCHAIN kann nur benutzt werden, wenn ein Natural-Objekt unterbrochen worden ist.

Dieses Kommando zeigt die Objekte auf der aktuellen Ebene und auf allen übergeordneten Ebenen sowie, falls zutreffend, die aktuelle Global Data Area (GDA) an. Außerdem werden Informationen über die Unterbrechung angezeigt.

ON/OFF

Wenn Sie im Debugger das Kommando `ON` oder `OFF` eingeben, wird der Test-Modus ein- bzw. ausgeschaltet. Siehe auch *TEST ON/OFF*.

PROFILE

Das Kommando `PROFILE` ruft den Bildschirm **User Profile** auf. Dort können Sie die Profileinstellungen für den Debugger ändern.

User Profile-Bildschirm

Der Bildschirm **User Profile** bietet folgende Optionen:

Option	Erklärung
Reset debug environment automatically on exit	Gibt an, dass ein automatischer Reset Ihrer Debug-Umgebung erfolgt, wenn Sie den Debugger verlassen.
File for loading/saving debug environments	Gibt an, in welche bzw. aus welcher Systemdatei Debug-Umgebungen gespeichert bzw. geladen werden sollen: `FUSER` (Standardeinstellung), `FNAT` oder `SPAD` (Scratch-Pad File).
Confirm EXIT/CANCEL before execution	Gibt an, dass vor der Ausführung eines `EXIT`- oder `CANCEL`-Kommandos eine Bestätigungsabfrage erfolgen soll. Die Standardeinstellung ist `N` (No).
Stack unknown commands	Gibt an, dass ein unbekanntes Debug-Kommando, das eingegeben wird (z.B. der Name eines gerufenen Programms) im Natural-Stack zwischengespeichert werden soll. Wenn dies der Fall ist, wird nach der Eingabe eines unbekannten Debug-Kommandos der Debugger sofort verlassen und das Kommando wird ausgeführt. Wenn diese Option nicht angegeben worden ist, führt ein unbekanntes Debug-Kommando zu einer entsprechenden Fehlermeldung. Die Standardeinstellung ist `Y` (Yes).
Output device	Angabe eines Druckers für die Funktionen **Call statistics maintenance** (siehe *Objekte drucken*) und **Statement execution statistics maintenance** (siehe *Statements drucken*). Der Standardwert ist `HARDCOPY`.

Option	Erklärung
	Wenn Sie die Ausgabe an einen anderen Drucker leiten möchten, müssen Sie `HARDCOPY` durch den Namen eines anderen, von Ihrem Systemadministrator verfügbar gemachten gültigen Druckers ersetzen.
Maximum Debug buffer size in KB	Gibt die maximale Größe (in Kilobytes) des Debug-Pufferspeichers an. Der Debug-Pufferspeicher wird bedarfsabhängig automatisch vergrößert, jedoch nur bis zur hier angegebenen maximalen Größe. Geben Sie `0` ein, um anzugeben, das keine Einschränkung besteht, oder einen Wert von `4 - 16384` (muss ein Vielfaches von 4 sein). Bei Überschreiten des Limits können keine weiteren Debug-Einträge definiert werden, und es werden keine zusätzlichen Call- oder Statement-Ausführungsstatistiken generiert.

SCAN

Kann nur bei der Funktion **List object source** angewendet werden, siehe *List Object Source (Objekt-Sourcecode anzeigen)*.

Dieses Kommando ermöglicht die Suche nach einer Zeichenkette im Sourcecode eines Objekts.

- `SCAN` sucht nach dem Wert, der begrenzt wird durch Leerzeichen oder beliebige Zeichen, bei denen es sich weder um Buchstaben noch um numerische Zeichen handelt.
- `SCAN ABS` führt zu einer absoluten Durchsuchung des Sourcecode nach dem angegeben Wert, unabhängig davon welche anderen Zeichen den Wert umgeben.

Siehe auch die Syntax-Diagramme in *Debug-Kommandoübersicht und -syntax*.

SCREEN

Wenn Sie bei einer Unterbrechung eines Natural-Objekts das Kommando `SCREEN` eingeben, wird die aktuelle Bildschirmausgabe des unterbrochenen Natural-Objekts angezeigt. Wenn Sie ENTER drücken, erfolgt die Rückkehr in den Debug-Modus.

SET OBJECT

Das Kommando `SET OBJECT` dient zum Ändern des **Standard-Objekts**, wie im entsprechenden Abschnitt in ***Debugger starten*** beschrieben. Siehe auch die Syntax des Direktkommandos `SET` im Abschnitt ***Debug-Kommandoübersicht und -syntax***.

STACK

Wenn Sie das Kommando `STACK` eingeben, wird der Inhalt des Eintrags, der sich oben auf dem Natural-Stack befindet, angezeigt. Es können bis zu 15 oben auf dem Stack abgelegte Eintragselemente angezeigt werden. Elemente, die mehr als 55 Zeichen haben, werden abgeschnitten und mit einem Stern (*) markiert.

Anmerkung: Wenn irgendein Einzelelement länger als 249 Zeichen ist, wird eine Fehlermeldung angezeigt.

SYSVARS

Wenn Sie das Kommando `SYSVARS` eingeben, werden die aktuellen Werte eines begrenzten Satzes an Systemvariablen angezeigt.

TEST ON/OFF

Das Kommando `TEST ON` bzw. `TEST OFF` dient zum **Ein- bzw. Ausschalten des Test-Modus**. Im Debugger brauchen Sie, wie weiter **oben** beschrieben, nur `ON` oder `OFF` einzugeben.

Die Benutzung des Kommandos `TEST` kann durch Natural Security unterbunden werden, siehe *Command Restrictions* im Abschnitt *Library Maintenance* in der *Natural Security*-Dokumentation.

17 Debug-Kommandoübersicht und -syntax

Dieser Abschnitt beschreibt alle Debugger-Kommandos, mit denen Sie Debugging-Funktionen per Direktaufruf ausführen oder in Debugger-Bildschirmen navigieren können.

Eine Erklärung komplexerer Kommandostrukturen mit benutzerdefinierten Operanden finden Sie im Abschnitt *Syntax-Diagramme* weiter unten.

Alle Debug-Kommandos

Die in der folgenden Tabelle aufgelisteten Kommandos können in der Kommandozeile eines beliebigen Debugger-Bildschirms eingegeben werden. Ein unterstrichener Teil eines Debug-Kommandos oder -Unterkommandos stellt dessen minimale Abkürzung dar.

Kommando	Unterkommando(s)	Erklärung
-		In einer Liste eine Seite nach oben blättern.
--		An den Anfang einer Liste blättern.
TOP		
+		In einer Liste eine Seite nach unten blättern
++		An das Ende einer Liste blättern.
BOTTOM		
ACTIVATE (**Syntax** siehe unten)	BREAKPOINT oder BP	**Breakpoints aktivieren**, siehe *Breakpoint Maintenance (Verwaltung der Breakpoints)*.
	SPY	Breakpoints *und* Watchpoints aktivieren, siehe auch **Spy aktivieren** in *Spy Maintenance (Verwaltung der Debug-Einträge)*.
	WATCHPOINT oder WP	**Watchpoints aktivieren**, siehe *Watchpoint Maintenance (Verwaltung der Watchpoints)*.
BM		Menü **Breakpoint Maintenance** aufrufen, siehe *Breakpoint Maintenance (Verwaltung der Breakpoints)*.
BREAK		Fenster **Debug Break** anzeigen, siehe auch BREAK in *Kommandos zum Navigieren und Anzeigen von Informationen*.
CANCEL		Aktuelle Operation abbrechen und/oder Bildschirme ohne Speicherung der Änderungen verlassen.

Kommando	Unterkommando(s)	Erklärung
DBLOG	A oder Q oder D	Utility DBLOG aus dem Debugger heraus aufrufen (siehe *Utilities*-Dokumentation). Um eine Datenbankumgebung anzugeben, können Sie eines der folgenden Unterkommandos benutzen: ■ A = Adabas (Standardeinstellung) ■ Q = SQL ■ D = DL/I **Anmerkung:** Während einer Debug-Unterbrechung können Sie nur eines der oben genannten Unterkommandos angeben.
DEACTIVATE oder DA (**Syntax** siehe unten)	BREAKPOINT oder BP	**Breakpoint deaktivieren**, siehe *Breakpoint Maintenance (Verwaltung der Breakpoints)*.
	SPY	Breakpoints *und* Watchpoints deaktivieren, siehe auch ***Spy deaktivieren***.
	WATCHPOINT oder WP	**Watchpoint deaktivieren**, siehe auch *Watchpoint Maintenance (Verwaltung der Watchpoints)*.
DELETE (**Syntax** siehe unten)	BREAKPOINT oder BP	**Breakpoint löschen**, siehe auch *Breakpoint Maintenance (Verwaltung der Breakpoints)*.
	SPY	Breakpoints *und* Watchpoints löschen, siehe auch ***Spy löschen***.
	WATCHPOINT oder WP	**Watchpoint löschen**, siehe *Watchpoint Maintenance (Verwaltung der Watchpoints)*.
	ENVIRONMENT	Angegebene Debug-Umgebung löschen, siehe auch ***Debug-Umgebung löschen***.
DISPLAY (**Syntax** siehe unten)	BREAKPOINT oder BP	**Breakpoint anzeigen**, siehe *Breakpoint Maintenance (Verwaltung der Breakpoints)*.
	SPY	Breakpoints *und* Watchpoints anzeigen, siehe auch ***Spy anzeigen***.
	WATCHPOINT oder WP	**Watchpoint anzeigen**, siehe *Watchpoint Maintenance (Verwaltung der Watchpoints)*.

Kommando	Unterkommando(s)	Erklärung
	CALL	Statistiken anzeigen zu Natural-Objekten, die während der Ausführung einer Operation aufgerufen worden sind, siehe auch ***Aufgerufene Objekte anzeigen***.
	EXEC	Statistiken anzeigen zu ausgeführten Statement-Zeilen von aufgerufenen Natural-Objekten, siehe auch ***Nur die ausgeführten Statement-Zeilen anzeigen***.
	HEXADECIMAL	Inhalt von Variablen in hexadezimalem Format anzeigen.
	NOCALL	Statistiken anzeigen zu Natural-Objekten, die während der Ausführung einer Operation nicht aufgerufen worden sind, siehe auch ***Nicht aufgerufene Objekte anzeigen***.
	NOEXEC	Statistiken anzeigen zu nicht ausgeführten Statement-Zeilen von aufgerufenen Natural-Objekten, siehe auch ***Nur die nicht ausgeführten Statement-Zeilen anzeigen***.
	OBJECT	Statistiken anzeigen zur Aufrufhäufigkeit von Objekten, siehe auch ***Alle Objekte anzeigen***.
	STATEMENT	Statistiken anzeigen zu ausgeführten und nicht ausgeführten Statement-Zeilen von aufgerufenen Natural-Objekten, siehe ***Alle Statement-Zeilen anzeigen*** .
	VARIABLE	**Variablen anzeigen** für unterbrochene Natural-Objekte, siehe ***Variable Maintenance (Anzeigen und Ändern von Variablen)***.
	XSTATISTICS	Statistik-Zusammenfassung von Ausführungsstatistiken anzeigen, siehe auch ***Statement-Ausführungsstatistiken anzeigen***.
EM		Menü **Debug Environment Maintenance** aufrufen, siehe *Debug Environment Maintenance (Verwaltung der Debug-Umgebung)*.
ESCAPE	BOTTOM	Verarbeitung einer Schleife stoppen und Verarbeitung mit dem ersten Statement nach der Schleife fortsetzen, siehe `ESCAPE BOTTOM` in *Kommandos zur Ausführungssteuerung*.
	ROUTINE	Verarbeitung eines unterbrochenen Natural-Objekts stoppen und, falls vorhanden, mit einem anderen Objekt fortsetzen, siehe `ESCAPE ROUTINE` in *Kommandos zur Ausführungssteuerung*.
EXIT		Aktuellen Bildschirm verlassen, siehe `EXIT` in *Kommandos zur Ausführungssteuerung*.
FLIP		Anzeige der beiden PF-Tastenreihen umschalten (PF1 bis PF12 und PF13 bis PF24).
GO		Rückgabe der Steuerung an die Ausführung des unterbrochenen Natural-Objekts, siehe `GO` in *Kommandos zur Ausführungssteuerung*.
LAST		Zuletzt eingegebenes Kommando anzeige. Die drei letzten Kommandos werden gespeichert und können abgerufen werden.
LEFT		Zur linken Seite einer Sourcecode-Auflistung verschieben.
LIST		Sourcecode eines Objekts anzeigen.

Kommando	Unterkommando(s)	Erklärung
(**Syntax** siehe unten)	BREAK	Sourcecode eines Objekts mit der aktuellen Unterbrechung anzeigen. Die relevante Statement-Zeile wird hervorgehoben.
	LASTLINE	Sourcecode eines Objekts mit der letzten Zeile anzeigen, die vor der aktuellen Unterbrechung ausgeführt wurde.
LOAD (**Syntax** siehe unten)	ENVIRONMENT	Angegebene Debug-Umgebung laden, siehe ***Debug-Umgebung laden***.
MENU		Hauptmenü **Debug Main Menu** aufrufen.
MODIFY (**Syntax** siehe unten)	BREAKPOINT oder BP	**Breakpoint ändern**, siehe *Breakpoint Maintenance (Verwaltung der Breakpoints)*.
	SPY	Bildschirm **Modify Breakpoint** oder **Modify Watchpoint** aufrufen, siehe auch ***Debug-Einträge (Spies) ändern*** in *Spy Maintenance (Verwaltung der Debug-Einträge)*.
	WATCHPOINT oder WP	**Watchpoint ändern**, siehe *Watchpoint Maintenance (Verwaltung der Watchpoints)*.
	HEXADECIMAL	Inhalt von Variablen in hexadezimalem Format ändern.
	VARIABLE	Bildschirm **Display Variable** zwecks Änderung aufrufen, siehe ***Variable ändern***.
NEXT		Nächstes, für einen Breakpoint oder Watchpoint angegebenes Kommando ausführen.
OBJCHAIN		Ausgeführte Objekte auf verschiedenen Programmebenen anzeigen, siehe OBJCHAIN in *Kommandos zum Navigieren und Anzeigen von Informationen*.
ON oder OFF		Test-Modus ein- und ausschalten, siehe auch ***Test-Modus ein- und ausschalten***.
PRINT (**Syntax** siehe unten)	CALL	Statistiken drucken zu Natural-Objekten, die während der Ausführung einer Anwendung aufgerufen wurden, siehe auch ***Aufgerufene Objekte anzeigen***.
	EXEC	Statistiken drucken zu ausgeführten Statement-Zeilen von aufgerufenen Natural-Objekten, siehe auch ***Nur die ausgeführten Statement-Zeilen anzeigen***.
	NOCALL	Statistiken drucken zu Natural-Objekten, die während der Ausführung einer Anwendung nicht aufgerufen wurden, siehe auch ***Nicht aufgerufene Objekte anzeigen***.

Kommando	Unterkommando(s)	Erklärung
	NOEXEC	Statistiken drucken zu nicht ausgeführten Statement-Zeilen von aufgerufenen Natural-Objekten, siehe auch ***Nur die nicht ausgeführten Statement-Zeilen anzeigen***.
	OBJECT	Statistiken drucken zur Aufrufhäufigkeit (*Call Frequency*) von Objekten, siehe auch ***Aufgerufene Objekte anzeigen***.
	STATEMENT	Statistiken drucken zu ausgeführten und nicht ausgeführten Statement-Zeilen von aufgerufenen Natural-Objekten, siehe auch ***Alle Statement-Zeilen anzeigen***.
	XSTATISTICS	Statistiken drucken zu ausgeführten Statement-Zeilen, siehe auch ***Statement-Ausführungsstatistiken anzeigen***.
PROFILE		Bildschirm **User Profile** anzeigen. Dort können Sie die Profileinstellungen des Debuggers ändern, siehe ***Kommandos zum Navigieren und Anzeigen von Informationen***.
RESET (**Syntax** siehe unten)	ENVIRONMENT	Aktuelle Debug-Umgebung zurücksetzen, siehe ***Debug-Umgebung zurücksetzen***.
RIGHT		Zur rechten Seite einer Sourcecode-Auflistung verschieben.
RUN		Test-Modus ausschalten und Programmausführung fortsetzen.
SAVE (**Syntax** siehe unten)	ENVIRONMENT	Aktuelle Umgebung zurücksetzen und die Debug-Angaben speichern, siehe auch ***Debug-Umgebung speichern***.
SCAN	ABS	Nur anwendbar bei der Funktion **List object source** (siehe *List Object Source (Objekt-Sourcecode anzeigen)*). Suche nach einem Wert im Sourcecode eines Objekts, siehe SCAN in *Kommandos zum Navigieren und Anzeigen von Informationen* and ***Syntax-Diagramme*** below.
SCREEN		Wenn bei Unterbrechung eines Objekts eingegeben, wird die aktuelle Bildschirmausgabe des unterbrochenen Natural-Objekts angezeigt. Rückkehr in den Debug-Modus nach Drücken von ENTER.
SET (**Syntax** siehe unten)	BREAKPOINT oder BP	Bildschirm **Set Breakpoint** aufrufen, siehe *Breakpoint Maintenance (Verwaltung der Breakpoints)*.
	CALL ON oder CALL OFF	Call-Statistik aktivieren oder deaktivieren, siehe *Call Statistics Maintenance (Statistiken über gerufene Objekte)*.
	OBJECT	Das für den Debugger angegebene Standard-Objekt ändern, siehe auch SET OBJECT in *Kommandos zum Navigieren und Anzeigen von Informationen*.

Kommando	Unterkommando(s)	Erklärung
	WATCHPOINT oder WP	Bildschirm **Watchpoint setzen** aufrufen, siehe ***Watchpoint Maintenance (Verwaltung der Watchpoints)***.
	XSTATISTICS ON oder XSTATISTICS COUNT oder XSTATISTICS OFF	Statement-Ausführungsstatistik aktivieren (ON oder COUNT) deaktivieren (OFF), siehe ***Funktion Statement Execution Statistics auf ON/OFF/COUNT setzen***.
SM		Menü **Spy Maintenance** aufrufen, siehe *Spy Maintenance (Verwaltung der Debug-Einträge)*.
STACK		Inhalt des obersten Eintrags im Natural-Stack anzeigen, siehe STACK in *Kommandos zum Navigieren und Anzeigen von Informationen*.
STEP	[*n*]	Unterbrochenes Natural-Objekt für eine mit dem Kommando angegebene Anzahl (*n*) ausführbarer Statements fortsetzen. Wenn Sie *n* nicht angeben, wird standardmäßig ein ausführbares Statement übersprungen. Siehe auch STEP in *Kommandos zur Ausführungssteuerung*.
	SKIPSUBLEVEL [*n*]	Schrittweise Bearbeitung von Natural-Objekten fortsetzen, ohne Programme auf Unterebenen zu erfassen. Sie können eine Ebenennummer (*n*) angeben. Siehe auch SKIPSUBLEVEL in *Kommandos zur Ausführungssteuerung*.
STOP		Debugger und unterbrochenes Natural-Objekt beenden. Die Eingabeaufforderung NEXT erscheint.
SYSVARS		Aktuelle Werte eines begrenzten Satzes an Systemvariablen anzeigen (außer datenbankbezogene Systemvariablen). Siehe auch ***Systemvariablen anzeigen***.
TEST ON oder TEST OFF		**Test-Modus ein- und ausschalten**.
WM		Ruft das Menü **Watchpoint Maintenance** auf. Beschreibung siehe *Watchpoint Maintenance (Verwaltung der Watchpoints)*.

Syntax-Diagramme

Die im Folgenden aufgeführten Syntax-Diagramme beziehen sich auf komplexere Kommandosequenzen.

Ausführliche Erläuterungen zu den Symbolen, die innerhalb der Syntax-Beschreibungen verwendet werden, siehe Abschnitt *Systemkommando-Syntax* in der *Systemkommandos*-Dokumentation.

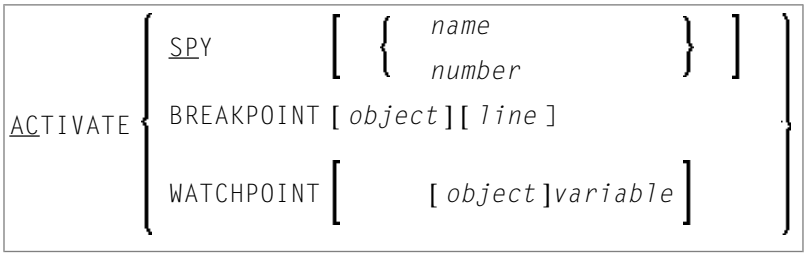
Zur besseren Lesbarkeit sind synonyme Schlüsselwörter in den unten aufgeführten Syntax-Diagrammen weggelassen worden. Eine Unterstreichung eines Schlüsselwortteils bedeutet, dass Sie das Schlüsselwort auch in entsprechend abgekürzter Form als eingeben können.

Gültige synonyme Schlüsselwörter sind:

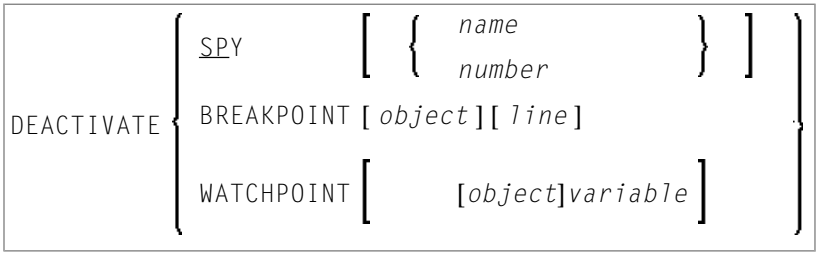
Schlüsselwort	Synonym
BREAKPOINT	BP
DEACTIVATE	DA
WATCHPOINT	WP

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

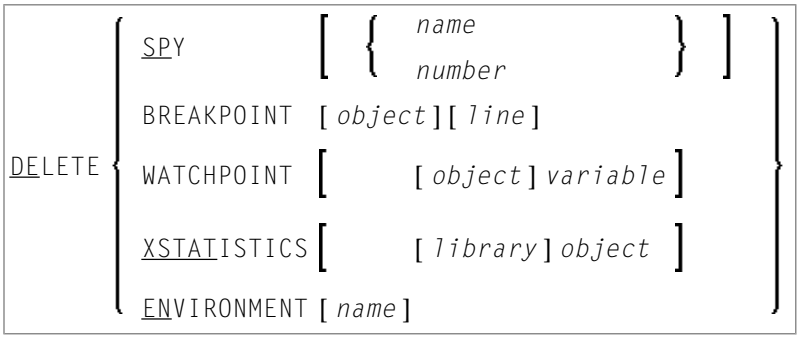
ACTIVATE



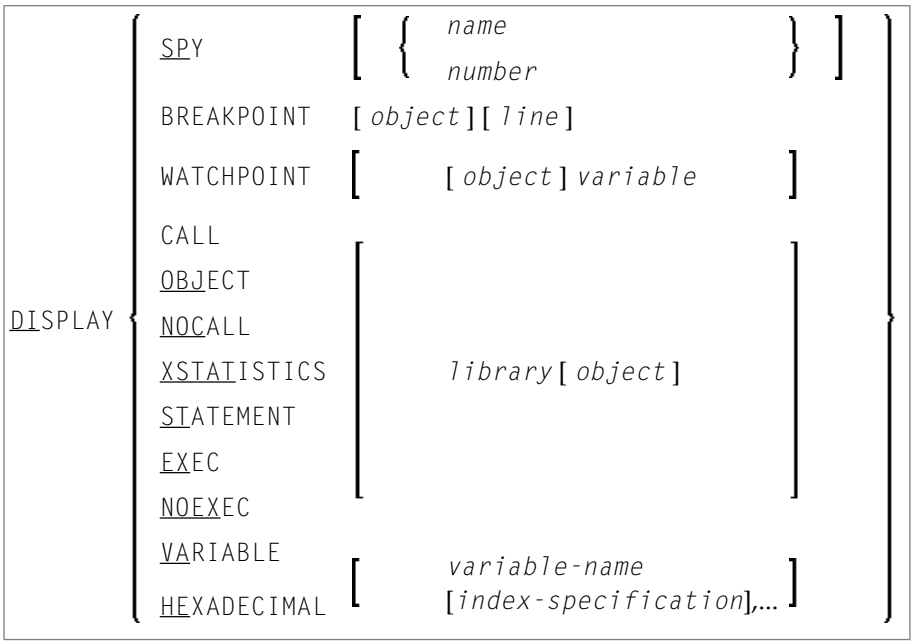
DEACTIVATE



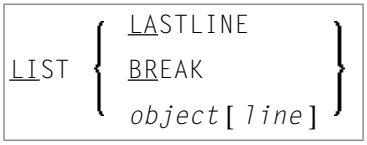
DELETE



DISPLAY

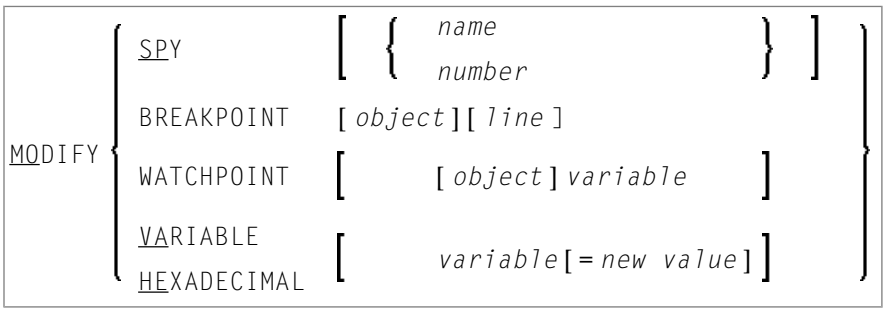


LIST

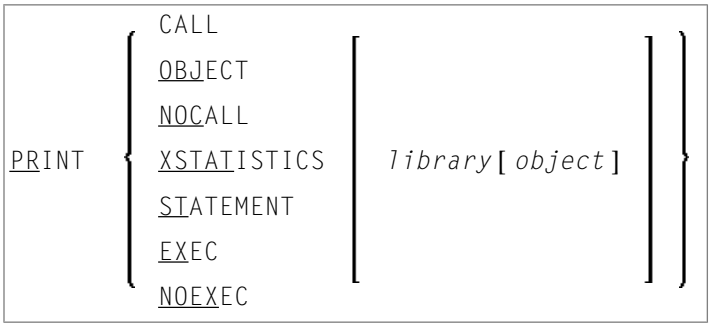


LOAD

MODIFY



PRINT

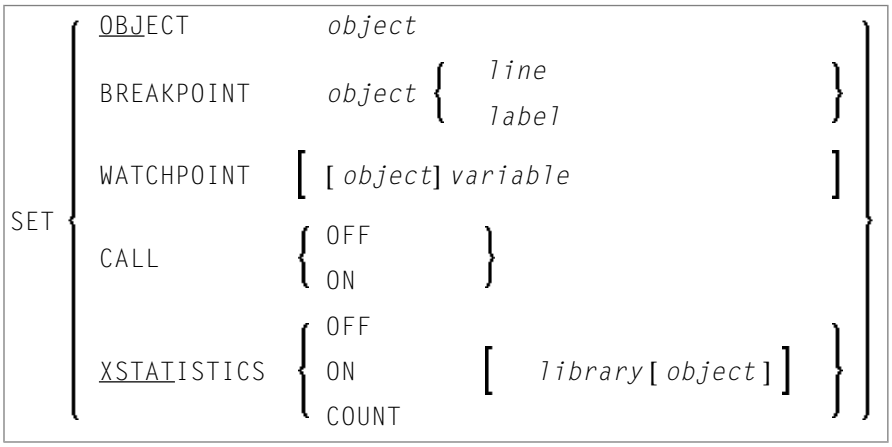


RESET

SAVE

```
SAVE ENVIRONMENT[ name ]
```

SET



18 Natural für Attached-Debugging vorbereiten

Einleitung

Dieses Dokument liefert Informationen, wie Sie den Debug Attach Server (DAS) aktivieren können, um das Debugging einer externe Natural-Anwendung mittels NaturalONE durchzuführen.

Eine externe Natural-Anwendung läuft in einer Natural-Umgebung, speichert aber ihren Sourcecode in einem NaturalONE-Project. Der Debug Attach Server (DAS) wird benutzt, um auf ein NaturalONE-Project zuzugreifen.

Weitere Informationen zur Benutzung des Debug Attach Server (DAS) siehe *NaturalONE*-Dokumentation.

Voraussetzungen für Attached Debugging

Damit Sie aus einer Natural-Sitzung heraus auf den NaturalONE-Debugger zugreifen können, müssen folgende Voraussetzungen erfüllt sein:

- Die Natural-Sitzung läuft in einer z/OS-, z/VSE- oder BS2000-Umgebung.
- NaturalONE ist installiert.
- Der Natural Development Server ist installiert und die installierte Version muss verbundenes Debugging unterstützen.
- Das Modul `NATAD`*`vrs`* (oder `NCIAD`*`vrs`* für eine CICS-Session auf z/OS) wird aus der Natural Development Server Library generiert und die Natural-Sitzung kann darauf zugreifen.
- Der Profilparameter `DBGAT` ist angegeben.
- Der Profilparameter `RCA` ist auf `NATATDBG` gesetzt.
- Der Profilparameter `RCALIAS` ist auf `(NATATDBG,NATAD`*`vrs`*`)` gesetzt, für CICS auf z/OS auf `(NATATDBG,NCIAD`*`vrs`*`)`.
- Der Debug Attach Server (DAS) läuft und kann über TCP/IP adressiert werden. Der DAS wird mit NaturalONE als `NATDAS.EXE`-Datei ausgeliefert.

Ausführliche Informationen zu den oben erwähnten Natural-Profilparametern, siehe *Parameter-Referenz*-Dokumentation.

Beispiel für z/OS Batch

Eine Natural-Batch-Anwendung soll mit NaturalONE auf Programmfehler untersucht werden. Der DAS-Server ist unter dem TCP/IP-Namen `DASSERV` verfügbar und empfängt am Port `50882`. Der NaturalONE-Debugger hat sich beim Debug Attach Server mit der Client ID `FRED` identifiziert. Die angebundene Debug-Schnittstelle befindet sich in der Library `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)
/*
```

Beispiel für z/VSE Batch

Eine Natural-Batch-Anwendung soll mit NaturalONE auf Programmfehler untersucht werden. Der DAS-Server ist unter dem TCP/IP-Namen `DASSERV` verfügbar und empfängt am Port `50882`. Der NaturalONE-Debugger hat sich beim Debug Attach Server mit der Client ID `FRED` identifiziert. Die angebundene Debug-Schnittstelle befindet sich in der Library `PRD.NAT`*`vrs`*`.LIBRARY`:

```
// DLBL NATvrs,'PRD.NATvrs.LIBRARY'
// LIBDEF PHASE,SEARCH=(NATvrs.NATvrs,NATvrs.NDVvrs,...)
// EXEC  NATBATvr,SIZE=(NATBATvr,120K),PARM='SYSRDR'
RCA=NATATDBG,RCALIAS=(NATATDBG,NATADvrs)
DBGAT=(ACTIVE=ON,HOST=DASSERV,PORT=50882,CLID=FRED)
/*
```

Beispiel für BS2000

Eine Natural-Batch-Anwendung soll mit NaturalONE auf Programmfehler untersucht werden. Der DAS-Server ist unter dem TCP/IP-Namen `DASSERV` verfügbar und empfängt am Port `50882`. Der NaturalONE-Debugger hat sich beim Debug Attach Server mit der Client ID `FRED` identifiziert. Die angebundene Debug-Schnittstelle befindet sich in der Library `NDV`*`vrs`*`.MOD`:

```
/LOGON
/SYSFILE SYSOUT=ATDEBUG.OUT
/SYSFILE SYSLST=ATDEBUG.LST
/FILE ADAPARM,DDLNKPAR
/FILE NATvrs.MOD,LINK=BLSLIB01
/FILE NDVvrs.MOD,LINK=BLSLIB02
/FILE CMPRMIN.RMDBG,LINK=CMPRMIN
/FILE DBGTRACE.NATBATCH,LINK=DBGTRACE
/START-EXE-PROG F-F=*LI-E(L=NATvrs.MOD,EL=NATBATvr,TYPE=L)
...
```

Die dynamische Parameterdatei `CMPRMIN.RMDBG` enthält folgende Natural-Parametereinstellungen:

```
RCA=NATATDBG,RCALIAS=(NATATDBG,NATADvrs),
DBGAT=(ACTIVE=ON,CLID=FRED,HOST=DASSERV,PORT=50882)
```