Natural for Mainframes

デバッガ

バージョン 4.2.5

October 2009

Natural

This document applies to Natural バージョン 4.2.5 and to all subsequent releases.

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

目次

1 デバッガ

デバッガを使用して、プログラムエラーの検出、エラーの場所の特定、エラーの修正、プログラム実行のテストや最適化、またはプログラムの実行を妨げるNaturalエラーの分析を行います。

チュートリアル	デバッガの最初の手順。
デバッガの概念	デバッガの基本概念。
デバッガの開始	デバッガを起動するための動作要件および手順。
テストモードのオンとオフの切り替え	デバッグを有効または無効にするためのテストモードの設定。
デバッグ環境のメンテナンス	定義済みデバッグ環境の保存と使用。
スパイのメンテナンス	ブレイクポイントとウォッチポイントの両方の設定、変更、削除、および有効化。
ブレイクポイントのメンテナンス	ブレイクポイントの設定、変更、削除、および有効化。 ブレイクポイント画面の内容の説明。
ウォッチポイントのメンテナンス	ウォッチポイントの設定、変更、削除、および有効化。 ウォッチポイント画面の内容の説明。
コール統計のメンテナンス	呼び出されるオブジェクトに関する統計の取得。
ステートメント実行統計のメンテナンス	実行されるステートメント行に関する統計の取得。
変数のメンテナンス	変数の表示と変更。
オブジェクトソースのリスト	オブジェクトソースの表示。
エラー処理	アプリケーションまたはデバッガの実行中に発生するエラーの処理。
実行制御コマンド	プログラムフローを制御するデバッガコマンド。

	ナビゲーションと情報コマンド	画面での移動、オブジェクト情報の表示、およびデバッガのプロファイル設定を行うデバッガコマンド。
	コマンドの概要と構文	すべてのデバッガコマンドおよび適切なコマンド構文。

2 デバッガのチュートリアル

このチュートリアルでは、デバッガの基本機能を紹介し、さまざまなデバッグ方式を説明します。 ここでは、簡単なシナリオを使用して、ランタイムエラーの分析やプログラム実行の制御を行うためのデバッガの使用方法を示します。

セッション 1～5 は、順番どおりに実行することが重要です。

注意:

1. 理解しやすいように、チュートリアルでは主に、デバッガ機能を実行するダイレクトコマンドを紹介します。代わりに使用できるメニュー機能は紹介しません。
2. このチュートリアルに含まれるすべてのデバッガ機能の詳細な説明については、『デバッガ』ドキュメントの残りの部分の該当セクションを参照してください。

前提条件

- Natural でのプログラミングに慣れている必要があります。
- セッション 1 を開始する前に、すべてのサンプルプログラム（DEBUG1P および DEBUG2P）およびサンプルサブプログラム（DEBUG1N、DEBUG2N、DEBUG3N、および DEBUG4N）を作成する必要があります。これらのサンプルは、このチュートリアルの後にある「サンプルソース」セクションに記載されています。 システムコマンド `STOW` で、これらのオブジェクトを保存し、カタログします。

デバッグの基本

デバッガを使用して、Natural オブジェクトの実行フローを特定のデバッグイベントで中断し、中断したオブジェクトの現在のステータスの情報を取得できます。例えば、次に実行されるステートメント、変数の値、および呼び出されるオブジェクトの階層構造（プログラムレベル）などを取得できます。

プログラムを中断するには、基本的に次の 2 つの主な手順を実行して、デバッガに制御を渡す必要があります。

1. システムコマンド `TEST ON` でデバッガを有効にします。

   これによりデバッガは、Natural ランタイムシステムによって実行される各ステートメントの制御を受け取ることができます。
2. 実行される Natural オブジェクトに、1 つ以上のデバッグエントリ（ブレイクポイントとウォッチポイント）を設定します。

   これによりデバッガは、Natural ランタイムシステムから制御を受け取り、プログラムの実行を中断するタイミングを判断できます。

Natural エラーが発生した場合は、プログラムの実行が必ず中断されます。 この場合は、デバッグエントリが不要となり、自動的に制御がデバッガに移動します。

考えられるすべてのプログラム中断の概要は、次のとおりです。

プログラム中断	説明
ブレイクポイント	Natural オブジェクトのステートメント行でプログラムの中断を引き起こします。 デバッガは、ブレイクポイントが設定されたステートメント行が実行されるたびに、プログラム実行を中断します。つまり、この行に含まれるステートメントが処理される*前*に中断します。
ウォッチポイント	Natural オブジェクトの変数でプログラムの中断を引き起こします。 デバッガは、ウォッチポイントが設定された変数の内容が変化するたびに、プログラム実行を中断します。つまり、この変数を参照するステートメントが処理された*後*に中断します。
ステップモード	プログラム実行中にオブジェクトをステップ実行します。 ステップモードはデバッガコマンドにより開始されます。デバッガはその前に、ブレイクポイントまたはウォッチポイントにより制御を受け取っている必要があります。ステップモードでは、このオブジェクトに含まれる各実行ステートメントが処理される*前*に、デバッガがプログラム実行を中断します。
Natural エラー	自動的にプログラムの中断を引き起こします。

セッション 1 - Natural エラーの分析

このセッションでは、プログラムの実行中に発生する Natural エラーの調査方法を説明します。

手順 2.1. Natural エラーをシミュレートするには

- NEXT プロンプトから DEBUG1P を実行します。

  次の Natural エラーメッセージが表示されます。「`DEBUG1N 0180 NAT0954 プログラムの実行中に異常終了 SOC7 が発生しました。`」

  このメッセージは、サブプログラム DEBUG1N の行 180（`BONUS := SALARY * PERCENT / 100`）を指しています。 これは、参照された 1 つ以上の変数に対して、誤った値が返されたことを示します。 ただし、この時点では、これが問題の本当の原因を示しているのではありません。また、変数値がデータベースから取得されている場合（典型的な例は従業員レコードです）は、原因の特定が困難である可能性があります。

手順 2.2. デバッガを有効にして、さらに問題を調査するには

1 NEXT プロンプトで次のように入力します。

```
TEST ON
```

メッセージ「Test mode started.」は、デバッガが有効であることを示します。

注意: TEST ON は、現在のセッション中、または「TEST OFF」を入力してデバッガを無効にするまでは、有効なままです。

2 再び、NEXT プロンプトから DEBUG1P を実行します。

次の例のような［**Debug Break**］ウィンドウが表示されます。

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

Natural エラーが発生するため、自動的に制御がデバッガに移動し、［**Debug Break**］ウィンドウが表示されます。

エラーの発生場所に関する詳細情報が、ウィンドウの最上部に表示されます。詳細情報とは、Natural ニュークリアスのモジュール（NATARI2）（Software AG 技術サポートに連絡する際に役立ちます）、オブジェクトのタイプ（subprogram）、ライブラリ（DEBUG）とデータベース ID、およびシステムファイルのファイル番号（10,32）です。

［**Debug Break**］ウィンドウには、使用可能なデバッガの機能も表示されます。例えば、プログラム実行の継続（［**Go**］または［**Run**］）、デバッガの管理メニューの呼び出し（デバッグメインメニュー）、またはステップモードの有効化などの機能を使用できます。 機能を実行するには、該当するファンクションコードまたは PF キーを使用します。

手順 2.3. 誤ったステートメント行を調べるには

■ ［**Code**］フィールドで、デフォルトエントリ「G」の代わりに「L」を入力し、［**Listbreak**］機能を実行します。

DEBUG1N のソースが表示されます。

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

Command ===>



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

last line は、行170に含まれるステートメントが、正常に実行された最後の行であることを示しています。

問題を引き起こした行180のステートメントが強調表示され、* NAT0954 * という注記が付いています。

これは、エラーの原因が変数 SALARY または PERCENT の内容であることを示します。 PERCENT は正常に初期化されているため、原因はおそらく、SALARY です。

手順 2.4. SALARY の内容をチェックするには

1 コマンド行で、次のように入力します。

```
DIS VAR SALARY
```

次の例のような［**Display Variable**］画面が、変数 SALARY について表示されます。

```
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
```

メッセージ「Variable contains invalid data.」は、変数の内容が空白で、変数のフォーマットに該当しないことを示します。 これは、次の手順で説明するように、変数の内容を 16 進表現で表示すると、明らかになります。

2 PF11 キー（Hex）を押して、変数の内容を 16 進数で表示します。

次の例のような画面が表示されます。

```
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
```

16 進値は、変数がパック型数値フォーマットではなく、そのために、プログラム実行中に計算エラーが発生することを示しています。 DEBUG1P が DEBUG1N に誤った SALARY の値を提供していることは明らかです。

ヒント: 表示を英数字表現に戻すには、PF10 キー（Alpha）を押します。

3 コマンド行で、次のように入力します。

```
GO
```

コマンド GO を使用すると、制御がデバッガから Natural ランタイムシステムに戻り、プログラムの終了または次のデバッグイベントまでプログラム実行が継続されます。 この場合は、他にデバッグイベントは発生せず、すでに知られている Natural エラーメッセージを示す NEXT プロンプトが表示されます。

手順 2.5. オブジェクトソースで SALARY を修正するには

1 プログラムエディタで DEBUG1P を開き、SALARY := 99000 に入力されているコメント記号（*）を削除します。

2 システムコマンド STOW で、プログラムを保存し、カタログします。

3 DEBUG1P を実行します。

TEST ON が設定されたままですが、デバッガはプログラムを中断しません。 プログラムが正常に実行され、次のレポートを出力します。

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

EMPLOYEE RECEIVES:   100800.00
    PLUS BONUS OF:     3465.00




NEXT                                                                LIB=DEBUG
```

セッション 2 - ブレイクポイントの使用

特定のステートメント行にブレイクポイントを設定して、このステートメント行でプログラム実行を中断することができます。

手順 2.6. DEBUG1N のステートメント行にブレイクポイントを設定するには

1 NEXT プロンプトで次のように入力します。

```
TEST SET BP DEBUG1N 170
```

メッセージ「Breakpoint DEBUG1N0170 set at line 170 of object DEBUG1N.」は、DEBUG1N0170 という名前のブレイクポイントが、DEBUG1N サブプログラムのステートメント行 170 に設定されていることを示しています。

注意:

1. ブレイクポイントは実行ステートメントにのみ設定できます。 非実行ステートメントに設定しようとすると、所定のエラーメッセージが表示されます。
2. ブレイクポイントは通常、現在の Natural セッション中にのみ有効です。必要な場合は、以降のセッションのためにブレイクポイントを保存できます。「*デバッガの使用に関するその他のヒント*」の「*ブレイクポイントおよびウォッチポイントの保存*」を参照してください。

2 DEBUG1P を実行します。

これで、新しいブレイクポイントが設定されたステートメント行で、デバッガがプログラム実行を中断します。 ［**Debug Break**］ウィンドウが表示されます。

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

このウィンドウには、ブレイクポイントの名前、対応するステートメント行とオブジェクト、およびこのオブジェクトを含むライブラリが表示されます。 また、サブプログラム DEBUG1N の動作レベルも表示されます。

手順 2.7. ［Debug Break］ウィンドウが示すステートメントを表示するには

■ ［**List break**］機能を実行します。

DEBUG1N のソースが［**List Object Source**］画面に表示されます。

```
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
```

［**Debug Break**］ウィンドウに示されたステートメント行 170 が強調表示されています。［**Message**］列には、このステートメント行に設定されたブレイクポイントの名前（DEBUG1N0170）と、最後に実行されたステートメント行（last line が示す行 160）が表示されます。
注意：ブレイクポイントは、ブレイクポイントが設定されたステートメントが処理される*前*にプログラム実行を中断します。

［**List Object Source**］画面には、現在のオブジェクトに関する詳細情報を表示するためにいくつかのダイレクトコマンドを入力できます。 例えば、次の手順で説明するように、すべての変数を表示できます。

手順 2.8. DEBUG1N に含まれる変数のリストを表示するには

■ コマンド行で、次のように入力します。

```
DIS VAR
```

次のような［**Display Variables**］画面が表示されます。

```
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
```

この画面には、DEBUG1N で定義されたすべての変数のリストが表示されます。 BONUS の invalid data（無効データ）の注釈は無視できます。 この場合は、BONUS がターゲットオペランドとしてのみ使用されているため、正しく初期化されている必要はありません。 ただし、別のデバッガコマンドを実行するには、次の手順で BONUS の内容を変更してください。

手順 2.9. BONUS の内容をチェックして変更するには

1 BONUS の隣の［**Co**］列に、次のように入力します。

```
MO
```

または:

コマンド行で、次のように入力します。

```
MOD VAR BONUS
```

次のような［**Modify Variable**］画面が表示されます。

```
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 16 進表示を使用すると、変数がパック型数値のフォーマットでないことを確認できます。表示を英数字表現に戻すには、PF10 キー（Alpha）を押します。

3 ［**Contents**］フィールドに、12345.00 などのパック型数値フォーマットの値を入力し、PF5 キー（Save）を押します。

次の例のような画面が表示されます。

```
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
```

［**Contents**］の変更を確認するメッセージが表示されます。

4 PF9 キー（Li Br）または PF3 キー（Exit）を押します。

［**List Object Source**］画面が表示されます。

5 コマンド行で、次のように入力します。

```
GO
```

デバッガがNaturalランタイムシステムに制御を戻し、これ以上、デバッグイベントは発生しないため、DEBUG1P の実行が終了します。 プログラムが生成したレポートが出力されます。

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

EMPLOYEE RECEIVES:   100800.00
    PLUS BONUS OF:     3465.00




NEXT                                                                LIB=DEBUG
```

6 次のセッションへ進む前に、NEXTプロンプトで次のように入力して、現在のすべてのブレイクポイントを削除します。

```
TEST DEL BP * *
```

すべてのブレイクポイント（この場合は1つのブレイクポイントのみ）が削除されることを確認するメッセージが表示されます。

セッション 3 - ウォッチポイントの使用

DEBUG1P および DEBUG1N は、1 名の従業員に支払われるボーナスと給与の計算を実行します。 複数の従業員レコードが処理される場合は、変数 BONUS が正しく更新されたかどうかをテストすることをお勧めします。 そのためには、この変数にウォッチポイントを設定します。 ウォッチポイントを使用すると、指定した変数の内容が変更されたときに、デバッガがプログラム実行を中断できます。

手順 2.10. 変数 BONUS にウォッチポイントを設定するには

1 NEXT プロンプトで次のように入力します。

```
TEST SET WP DEBUG1N BONUS
```

メッセージ「Watchpoint BONUS set for variable EMPLOYEE.BONUS.」は、DEBUG1N サンプルサブプログラムの変数 BONUS にウォッチポイントが設定されたことを示しています。

注意:

1. デバッガ画面のコマンド行にデバッガのダイレクトコマンドを入力する場合は、キーワード TEST を省略する必要があります。 例えば、「TEST SET WP DEBUG1N BONUS」ではなく、「SET WP DEBUG1N BONUS」のみを入力します。
2. ウォッチポイントは通常、現在の Natural セッション中にのみ有効です。必要な場合は、以降のセッションのためにウォッチポイントを保存できます。「*デバッガの使用に関するその他のヒント*」の「*ブレイクポイントおよびウォッチポイントの保存*」を参照してください。

2 NEXT プロンプトから DEBUG1P を実行します。

デバッガは、新しいウォッチポイントでプログラム実行を中断し、［**Debug Break**］ウィンドウを呼び出します。

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

このウィンドウは、行180でウォッチポイントが検出されたことを示します。この行には、変数 BONUS を処理するステートメントが含まれています。

デバッガは、BONUSのステートメントが処理された*後*にプログラム実行を中断しています。デバッガが、変数の内容が変更されたことを認識できたのは、処理された後のみであるためです。

3 ［**List break**］機能を実行します。

［**List Object Source**］画面は、次の例のように表示されます。

```
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
```

変数BONUSを参照するステートメントが強調表示され、［**Message**］列には、この変数に設定されたウォッチポイントの名前が表示されます。

▶手順 2.11. BONUS の変更をチェックするには

1 コマンド行で、次のように入力します。

```
DIS VAR BONUS
```

［**Display Variable**］画面が表示され、［**Contents**］フィールドに 3465.00 という値が示されます。 これは、変数 BONUS の内容が変更されたことを表します。

2 PF3 キー（Exit）を押して、［**List Object Source**］画面に戻ります。

手順 2.12. SALARY の変更をチェックするには

1 この後の手順で変数 SALARY の内容をテストするには、行 200 の［**Co**］列に次のように入力して、SALARY にブレイクポイントを設定します。

```
SE
```

［**List Object Source**］画面では、SET BP ダイレクトコマンドの代わりに、SE などの便利な行コマンドも使用できます。

［**Message**］列は、ブレイクポイント（BP）が行 200 に設定されていることを示しています。

```
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 コマンド行で、次のように入力します。

```
GO
```

［**Debug Break**］ウィンドウが表示されます。

```
+------------------- 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 ［**List break**］機能を実行します。

［**List Object Source**］画面は、次の例のように表示されます。

```
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
```

これはブレイクポイントであるため、SALARY を参照（および更新）するステートメントは、まだ実行されていません。 したがって、変数の内容は変更されていません。

4 コマンド行に、DIS VAR SALARY を入力して、SALARY の内容が変更されていないことを確認します。

変数画面を見ると、SALARY の値が、DEBUG1P で割り当てられた初期値である 99000 のままであることがわかります。

5 更新された変数の内容を表示するには、次のいずれかの方法を選んで、次のステートメントに進みます。

コマンド行で、次のように入力します。

```
STEP
```

または:

PF2 キー（Step）を押します。

次の例のような画面が表示されます。

```
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
```

1 行スキップして、行 200 にある次の実行ステートメントを処理し、SALARY を更新しました。 ［**Message**］列は、ステップモードが設定されていることを示しています。 ステップモードでは、次の実行ステートメントまで、デバッガがプログラム実行を継続します。

6 コマンド行に DIS VAR SALARY を入力して、変数の内容をチェックします。

［**Display Variable**］画面が表示され、［**Contents**］フィールドに 100800.00 という値が示されます。 これは、変数 SALARY の内容が変更されたことを示しています。

7 コマンド行で、次のように入力します。

```
GO
```

デバッガがNaturalランタイムシステムに制御を戻し、これ以上、デバッグイベントは発生しないため、DEBUG1P の実行が終了します。 プログラムが生成したレポートが出力されます。

セッション 4 - プログラムの論理的な流れの追跡

このセッションでは、さまざまなデバッグ方法について説明します。これらの方法を使用すると、多数のオブジェクトを含む複雑なNaturalアプリケーションへの理解を深め、全体の把握と制御を向上させることができます。

このセッションでは最初に、アプリケーションの論理的な流れをステートメントレベルで分析する手順を説明します。 そして、ブレイクポイントを使用して、プログラムの実行順序を調べる方法を示します。

このセッションでの手順の説明は、簡単な（しかも、十分に実行できる）サンプルアプリケーションに基づいています。このアプリケーションは、1 つのプログラム（DEBUG2P）と 3 つのサブプログラム（DEBUG2N、DEBUG3N、DEBUG4N）で構成されています。

手順 2.13. プログラムの先頭または末尾にブレイクポイントを設定するには

1 NEXT プロンプトに次のように入力して、DEBUG2P にブレイクポイントを設定します。

```
TEST SET BP DEBUG2P BEG
```

メッセージ「Breakpoint DEBUG2P-BEG set at line BEG of object DEBUG2P.」は、DEBUG1N にブレイクポイントが設定されていることを示しています。

特定の行番号でなく、キーワード BEG を使用すると、プログラムの先頭、つまり、プログラムで実行される最初のステートメントにブレイクポイントが設定されます。 例えば、INIT 節が使用される場合の DEFINE DATA ステートメントにも、ブレイクポイントを設定できます。こうすると、プログラムがカタログされるときに、実行ステートメントが生成されます。

ヒント:

また、キーワード END を指定して、実行される最後のステートメントにブレイクポイントを設定することもできます。 最後のステートメントは、END ステートメントだけでなく、FETCH または CALLNAT ステートメントである場合があります。

2 DEBUG2P を実行します。

［**Debug Break**］ウィンドウが表示されます。

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

これで、プログラムに設定された最初のブレイクポイントで、制御がデバッガに移動します。

3 ［**List break**］機能を実行して、ソースをチェックし、最初の実行ステートメント NAME := 'MEIER' でデバッガに制御が移動することを確認します。

手順 2.14. アプリケーションをステップ実行するには

1 ［**List Object Source**］画面で、PF2 キー（Step）を押すか、コマンド行に STEP を入力して、ステップモードを設定します。

実行される最後の行には、last line という注記が付いています。 実行される次のステートメントは強調表示され、step mode という注記が付いています。

ヒント:

デバッガを各ステートメントで停止させずに、アプリケーションをより短時間でステップ実行するには、STEP コマンドで、STEP 2 または STEP 10 のように、スキップするステートメントの数を指定します。

2 CALLNAT ステートメントに step mode という注記が付けられるまで、PF2 キー（Step）を繰り返し押します。

3 さらに PF2 キー（Step）を押して、CALLNAT を実行します。

呼び出されたサブプログラムDEBUG2Nが表示されます。ここでは、実行される次のステートメントが強調表示されています。

```
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
```

この方法を使用せずに、コマンド行にSTEP SKIPを入力して、CALLNATをスキップすることもできます。

この場合は、呼び出すプログラム DEBUG2 のステートメントだけがステップ実行され、呼び出されるサブプログラムの各ステートメントは実行されません。

▶手順 2.15. オブジェクトの実行レベルを表示するには

1 DEBUG2N の［**List Object Source**］画面で、コマンド行に次のように入力します。

```
OBJCHAIN
```

次のような［**Break Information**］画面が表示されます。

```
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).
```

すでに知られているオブジェクト情報の他に、プログラムが GDA（グローバルデータエリア）を参照するかどうかがこの画面に示されます。

2 Enter キーを押して、1 ページ下方にスクロールします。

次の例のような画面が表示されます。

```
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
```

この画面は、オブジェクトが実行される動作レベルを示します。サブプログラム DEBUG2N はレベル 2 で、プログラム DEBUG2P（これがサブプログラムを呼び出します）はその上位のレベル 1 で実行されます。

3 Enter キーを押します。

［**List Object Source**］画面が表示されます。

4 コマンド行で、次のように入力します。

```
GO
```

デバッガが Natural ランタイムシステムに制御を戻し、これ以上、デバッグイベントは発生しないため、DEBUG2P の実行が終了します。 プログラムが生成したレポートが出力されます。

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

EMPLOYEE RECEIVES:    99300.00
    PLUS BONUS OF:     3565.00




NEXT                                                                LIB=DEBUG
```

5 NEXTプロンプトで次のように入力して、現在設定されているすべてのブレイクポイントを削除します。

```
TEST DEL BP * *
```

すべてのブレイクポイントが削除されることを確認するメッセージが表示されます。

▶手順 2.16. プログラム実行に続くブレイクポイントを設定するには

1 NEXT プロンプトで次のように入力します。

```
TEST SET BP ALL BEG
```

メッセージ「Breakpoint ALL-BEG set at line BEG of object ALL.」が表示されます。

これは、実行される各オブジェクトの最初の実行ステートメントにブレイクポイントを設定したことを示します。

2 DEBUG2P を実行します。

DEBUG2P の［**Debug Break**］ウィンドウが表示されます。

3 ［**Go**］関数を繰り返し実行します。

［**Go**］を実行するたびに、次に呼び出されるオブジェクトが［**Debug Break**］ウィンドウに表示されます（最初に DEBUG2N、次に DEBUG3N と DEBUG4N）。 したがって、どのオブジェクトが、プログラム実行中のどの時点で呼び出されるかを簡単に判定できます。 また、各オブジェクトに対して、［**Debug Break**］ウィンドウのメニュー機能を適用できます。

4 NEXTプロンプトが表示されたら、次のように入力して、現在設定されているすべてのブレイクポイントを削除します。

```
TEST DEL BP * *
```

すべてのブレイクポイントが削除されることを確認するメッセージが表示されます。

セッション 5 - プログラム実行に関する統計の使用

デバッガを使用して、呼び出されるオブジェクトと、それらが呼び出される頻度の統計情報を表示できます。 さらに、実行されるステートメントと、その実行頻度を表示できます。

▶手順 2.17. プログラム実行中に呼び出されるオブジェクトがどれかをチェックするには

1 NEXT プロンプトで次のように入力します。

```
TEST SET CALL ON
```

メッセージ「Call statistics started.」は、統計機能が有効であることを示しています。

2 DEBUG2P を実行します。

デバッガは、実行されるすべてのオブジェクト呼び出しをログに記録します。その後、プログラムで生成されたレポートが出力されます。

3 NEXT プロンプトで次のように入力します。

```
TEST DIS CALL
```

次のような［**Display Called Objects**］画面が表示されます。

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



Command ===>


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

画面には、実行されるすべてのオブジェクトのリストが表示されます。ここでは、呼び出しプログラム（DEBUG2P）および、呼び出されたその他のすべてのオブジェクト（DEBUG2N、DEBUG3N、DEBUG4N）が表示されています。また、各オブジェクトが呼び出される頻度（CALLS）、呼び出されるオブジェクトのタイプ、オブジェクトが格納される場所とNaturalバージョン、ソースオブジェクトとカタログ化オブジェクトがあるかどうか、オブジェクトがいつカタログされたかも表示されます。

4 NEXT プロンプトが表示されるまで、PF3 キー（Exit）または PF12 キー（Canc）を押します。

手順 2.18. プログラム実行中に実行されるステートメントがどれかをチェックするには

1 NEXT プロンプトで次のように入力します。

```
TEST SET XSTAT COUNT
```

メッセージ「Statement execution counting started for library/object */*.」は、現在のライブラリに含まれるすべてのオブジェクト、およびこのライブラリに連結されたすべての steplib に対して統計機能が有効であることを示しています。

2 DEBUG2P を実行します。

デバッガは、プログラムが処理するすべてのステートメントをログに記録します。その後、プログラムで生成されたレポートが出力されます。

3 NEXT プロンプトで次のように入力します。

```
TEST DIS XSTAT
```

次の例のような［**List Statement Execution Statistics**］画面が表示されます。

```
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
```

この画面には、呼び出しの数（Obj. Called n Times）、実行ステートメントの数（Executable）、実行されたステートメントの数（Executed）、実行ステートメントの総数に対する、実行されたステートメントのパーセンテージ（%）、および実行されたステートメントの総数（Total No. Executions）のリストが含まれます。

4 DEBUG4N の隣の［**Co**］列に、次のように入力します。

```
DS
```

次のような統計画面が表示されます。

```
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
```

この画面は、ステートメントが実行された頻度と、処理されなかった実行ステートメントを示します。

デバッガの使用に関するその他のヒント

このセクションでは、デバッガの使用に関するその他のヒントを説明します。

- オブジェクトのタイムスタンプ
- ブレイクポイントおよびウォッチポイントの保存
- メンテナンス機能のデバッグメインメニュー
- メンテナンス画面のコマンドのヘルプ
- プログラム中断中に使用できる主な機能
- プログラム中断中の追加コマンドの次のオプション
- ラージ変数および配列の表示
- デバッガ統計の出力

- バッチモードでのデバッガの使用

オブジェクトのタイムスタンプ

カタログ化オブジェクトがソースオブジェクトと正確には対応しない場合は、デバッグエラーが発生する可能性があります。 ソースオブジェクトとカタログ化オブジェクトが確実に対応するようにするには、これらをシステムコマンド STOW で保存し、カタログします。

詳細については、「*動作要件*」セクションを参照してください。

ブレイクポイントおよびウォッチポイントの保存

現在のセッションで設定されているブレイクポイントとウォッチポイントをデバッグ環境として保存できます。今後のセッションでは、この環境をロードして使用できます。 これは、同じデバッグエントリを含むアプリケーションを繰り返しテストする場合に便利です。

詳細については、「*デバッグ環境のメンテナンス*」を参照してください。

メンテナンス機能のデバッグメインメニュー

ブレイクポイントの設定または統計の作成などのデバッガメンテナンス機能はすべて、ダイレクトコマンドか、またはデバッグメインメニューで提供されるメンテナンス機能を使用して実行できます。 このメニューは、次のいずれかを入力すると開きます。

- TEST
  （コマンドプロンプト）
- MENU
  （デバッガ画面のコマンド行）
- M
  （［**Debug Break**］ウィンドウの［**Code**］フィールド）

メンテナンス画面のコマンドのヘルプ

デバッガメンテナンス画面で使用できるダイレクトコマンドのリストを表示するには、PF1 キー（Help）を押すか、コマンド行に疑問符（?）を入力します。

リスト項目を含むデバッガメンテナンス画面には通常、項目をさらに処理するために使用できる行コマンドも表示されます。行コマンドは、目的の項目の横の［**Co**］列に入力します。 この列に疑問符（?）を入力すると、有効な行コマンドがリストされます。

プログラム中断中に使用できる主な機能

プログラム中断中に使用できる主な機能のリストは、次のセクションにあります。 これらの機能は［**Debug Break**］ウィンドウから、またはデバッガメンテナンス画面のコマンド行から実行できます。

［Debug］ウィンドウのコード	代替ダイレクトコマンド	機能
G	GO	次のデバッグイベントが発生するまで、プログラム実行を継続します。
L	LIST BREAK	デバッグイベントが発生したステートメント行に、オブジェクトソースのリストを表示します。
N	NEXT	ブレイクポイントまたはウォッチポイントに指定されている場合は、次の BREAK コマンドを実行します。 「*プログラム中断中の追加コマンドの次のオプション*」も参照してください。
R	RUN	テストモードをオフに切り替え、プログラム実行を継続します。
S	STEP	実行ステートメントを 1 行ずつ処理します。
V	DIS VAR	中断されたオブジェクトに定義されている変数のリストを表示します。

プログラム中断中の追加コマンドの次のオプション

ブレイクポイントまたはウォッチポイントを表示または変更しているときには、そのそれぞれにデバッガコマンド BREAK が付加されています。 このコマンドは［**Debug Break**］ウィンドウを呼び出します。また、このコマンドは削除しないでください。 ただし、プログラム中断中に実行される追加デバッガコマンドを、BREAK コマンドの後に指定できます。 追加コマンドは、コマンド行にコマンド NEXT を入力するか、［**Debug Break**］ウィンドウにファンクションコード「N」を入力すると、実行されます。

デバッガコマンドは、次の例に示すように、適切なブレイクポイントまたはウォッチポイントのメンテナンス画面の［**Commands**］フィールドに入力します。

```
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
```

上の例では、コマンド STACK の指示によりデバッガが Natural スタックを表示します。 コマンド DIS VAR BONUS の指示により、デバッガが指定された変数を表示します。 これは、ループにブレイクポイントを設定し、特定の変数の値のみを常に表示させる場合などに便利です。 こうすると、DIS VAR コマンドを繰り返し入力する必要がなくなります。

詳細については、「*ブレイクポイント画面のフィールドと列*」および「*ウォッチポイント画面のフィールドと列*」の各セクションにある［**Commands**］フィールドの説明を参照してください。

ラージ変数および配列の表示

［**Display Variable**］画面には、変数のすべての定義と、英数字または 16 進形式でのその内容が表示されます。 内容が現在の画面に収まらないラージ変数、または配列定義を持つ変数に使用できる表示機能の詳細については、「*［Display Variable］- 詳細*」セクションを参照してください。

デバッガ統計の出力

デバッガが生成した統計レポートを出力したり、PC にダウンロードしたりすることができます。

詳細については、「*コール統計のメンテナンス*」セクションの「*オブジェクトの出力*」および「*ステートメント実行統計のメンテナンス*」セクションの「*出力ステートメント*」を参照してください。

バッチモードでのデバッガの使用

デバッガは主に、オンラインモードでの対話式動作を目的として設計されています。 原則として、すべてのデバッガ機能はバッチモードで実行できますが、バッチでのオンライン動作の処理（PF キーの使用など）には、複雑なバッチプログラミングが必要な場合があります。 ただし、バッチ処理を行うと便利なデバッガ機能もあります。その1つの例は、「*バッチ処理*」セクションの「*バッチでの統計の生成と出力の例*」で説明されているように、アプリケーションに関する統計データの収集と出力です。

サンプルソース

このセクションには、セッション 1〜5 に必要なサンプルのプログラムとサブプログラムのソースコードを掲載しています。

プログラム **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
```

サブプログラム **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
```

プログラム **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
```

サブプログラム **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
```

サブプログラム **DEBUG3N**

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

サブプログラム **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 デバッガの概念

デバッガは、Natural オブジェクトの実行中に、デバッグを行うために Natural セッションの制御を受け取ります。 このため、プログラムの処理フローに従って、さまざまなプログラム調査を実行できます。

プログラムにデバッグエントリ（ブレイクポイントまたはウォッチポイント）を設定すると、デバッガがプログラムを中断する場所を指定できます。

プログラム実行が中断すると、プログラムで使用されている変数またはパラメータの内容を調べて、プログラムロジックを分析したり、Natural エラーの原因を特定したりできます。

このセクションでは、デバッガの機能に関する一般的な情報を説明します。

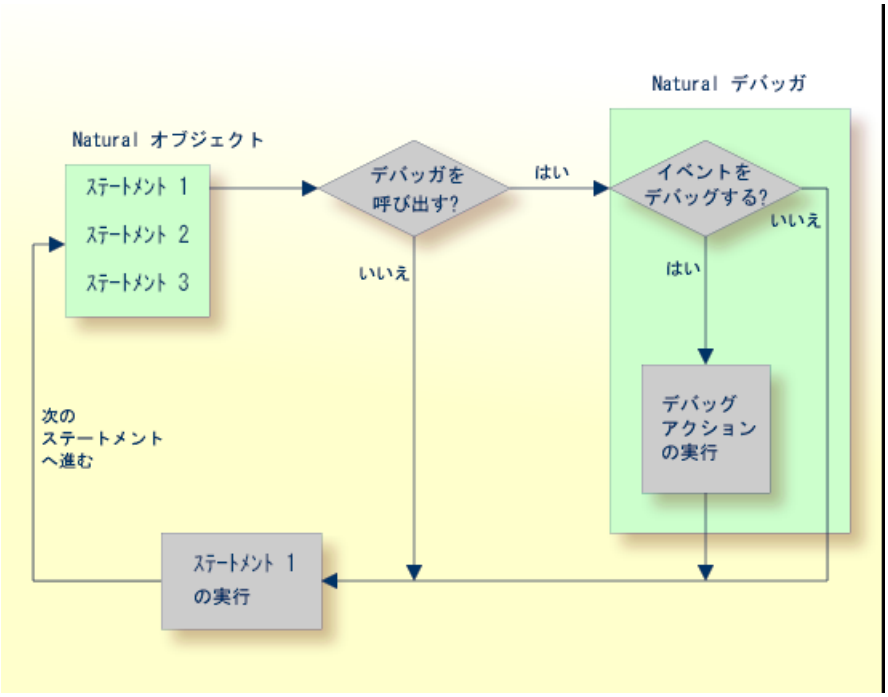
セッション制御および制御機能

テストモードが ON に設定されると、Natural セッションの制御がデバッガに移動します（「*テストモードのオンとオフの切り替え*」を参照）。 プロファイルパラメータ DBGERR が ON に設定されている（『パラメータリファレンス』ドキュメントを参照）場合は、デバッグエントリやテストモードの設定（ON または OFF）にかかわらず、Natural エラーが発生すると、デバッガが呼び出されます。

デバッガは、セッションを制御しているときに、次の 1 つ以上の機能を実行します。

- デバッグエントリをチェックします。
- ブレイクポイントが設定されたステートメント行で、Natural オブジェクトを中断します。
- ウォッチポイントが設定された変数の値が変更されると、Natural オブジェクトを中断します。
- 検出したデバッグエントリ（ウォッチポイントとブレイクポイントのいずれかまたは両方）の情報を表示します。
- 呼び出される Natural オブジェクトの統計を提供します。
- Natural オブジェクトで実行されるステートメントの統計を提供します。
- Natural エラーが発生すると、Natural オブジェクトを中断します。 「*エラー処理*」セクションも参照してください。

次の図は、デバッガで Natural オブジェクトが実行されるときの処理フローの例を示しています。



デバッグエントリ／スパイ

デバッグエントリは、デバッガ環境ではスパイとも呼ばれます。 使用できるデバッグエントリ（スパイ）は、ブレイクポイントとウォッチポイントの 2 タイプです。

以下では次のトピックについて説明します。

- メンテナンスと検証
- デバッグエントリの名前
- 初期状態または現在の状態
- デバッグイベントのカウンタ

- デバッグエントリのコマンド

メンテナンスと検証

現在のデバッグセッションのデバッグエントリは、該当するデバッガメンテナンス機能を使用して、設定、変更、一覧表示、表示、有効化、無効化、および削除が可能です。これらの機能については、デバッガのドキュメントの該当セクションで説明しています。 デバッグエントリは、将来使用するために保存することができます。これについては、「*デバッグ環境のメンテナンス*」で説明しています。

デバッグエントリの有効性チェックは、該当するメンテナンス画面でブレイクポイントまたはウォッチポイントが定義された直後、または、プログラム実行中に実行されます。

プログラム実行中に有効性チェックに失敗した場合は、「`Check for invalid spy definition`」という注記が［**Debug Break**］ウィンドウに表示されます（「*［Debug Break］ウィンドウ*」を参照）。 さらに、無効なブレイクポイントまたはウォッチポイントには、該当するブレイクポイントまたはウォッチポイントのメンテナンス画面でマークが付けられます。

デバッグエントリが設定または変更されると、Naturalは内部的に、オブジェクトが存在するライブラリ、データベースID、およびファイル番号を保存します。 オブジェクトは、現在のライブラリ、またはそのsteplibのいずれかに存在します。 その後に同じ名前のオブジェクトが別のライブラリから実行された場合、対応するデバッグエントリは実行されません。

デバッグエントリの名前

デバッガは各デバッグエントリに、名前と一意の番号（スパイ番号）を割り当てます。 デバッグエントリに割り当てられる名前（スパイ名とも呼ばれます）は、ユーザーが指定した名前、またはデバッガが作成したデフォルト名です。 デバッグエントリは、対応するデバッガコマンドで番号を指定して選択できます。 特定のステートメント行で複数のデバッグエントリを実行する必要がある場合は、その番号の昇順で実行されます。

初期状態または現在の状態

各デバッグエントリには、初期状態と現在の状態があります。設定可能な値は`A`（アクティブ）および`I`（非アクティブ）です。初期状態の値は、ブレイクポイントまたはウォッチポイントが設定または変更されるときに指定され、これにより、環境の起動時またはリセット後の各デバッグエントリの状態が決定されます。デバッグセッション中には、デバッグコマンド`ACTIVATE`および`DEACTIVATE`で状態を変更できます（「*コマンドの概要と構文*」の構文図も参照）。

デバッグイベントのカウンタ

各デバッグエントリにはイベントカウントがあり、デバッグエントリが実行されるたびに増えていきます。 デバッグエントリは、現在の状態が非アクティブの場合は実行されません。 ブレイクポイントまたはウォッチポイントのイベントカウントも増えません。

デバッグエントリの実行回数は 2 通りの方法で制限できます。

- デバッグエントリが実行される前にスキップ回数を指定できます。 その後、イベントカウントが指定されているスキップの数より多くなるまで、デバッグエントリは無視されます。
- 実行回数の上限を指定して、イベントカウントが指定の実行回数を上回ったら直ちにデバッグエントリが無視されるようにすることができます。

デバッグエントリのコマンド

各デバッグエントリ（ブレイクポイントまたはウォッチポイント）には、6個までのデバッグコマンドを指定できます。 これらのコマンドは、ブレイクポイントまたはウォッチポイントの実行時に実行されます。デバッグ中断中に適用可能なすべてのデバッガコマンドを使用できます。デフォルトのコマンドは BREAK コマンドで、これにより、次のセクションで説明する［**Debug Break**］ウィンドウが表示されます。

注意: デバッグエントリの設定時に BREAK コマンドを削除し、ダイアログを表示するコマンドを何も指定しないと、プログラムの中断時に制御を受け取る方法がなくなります。

［Debug Break］ウィンドウ

デバッガがセッションの制御を受け取ると、次の例のような［**Debug Break**］ウィンドウが表示されます。

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

［**Debug Break**］ウィンドウには、ブレイクを発生させたデバッグエントリのタイプと名前（つまり、対応するブレイクポイントまたはウォッチポイントの名前）、そのソースコード行番号、および中断された Natural オブジェクトの名前が示されます。

また、［**Debug Break**］ウィンドウの一番下には、Natural エラー（「エラー*処理*」の「*アプリケーション実行時のエラー*」も参照）、または、考えられる無効なデバッグエントリを示すメッセージが表示される場合があります。

［**Debug Break**］ウィンドウに用意されている機能は、次の表で説明しています。 詳細については、「*実行制御コマンド*」を参照してください。

機能	コード	説明
Go	**G**	指定された次のデバッグエントリまで、Naturalオブジェクトの実行を継続します。
List break	**L**	現在アクティブな Natural オブジェクトのコードのリストを表示します。 最後に実行されたステートメントが強調表示されます。
Debug Main Menu	**M**	デバッグメインメニューを呼び出します。このメニューでは、制御が渡されるデバッグエントリのメンテナンスに必要なすべての機能が提供されます。
Next break command	**N**	現在のブレイクポイントまたはウォッチポイントに指定された次のコマンドを実行します。
Run（テストモードを **OFF** に設定）	**R**	テストモードを OFF に設定して、Natural オブジェクトの実行を継続します。
Step mode	**S**	ステップモードで、Natural オブジェクトの実行を継続します。
Variable maintenance	**V**	現在アクティブな Natural オブジェクトの変数を表示し、これらの変数の内容を変更します。

4 デバッガの開始

このセクションでは、基本動作要件を説明し、デバッガの適用計画の作成方法についての大まかなガイドラインを示します。

Natural Security 環境でのデバッガ

デバッガの使用は、Natural Security によって制御されます。

- TEST システムコマンドを禁止することによって、未認可の使用に対してデバッガを保護できます。このシステムコマンドは、デバッガを起動します。『*Natural Security*』ドキュメントの「*Library Maintenance*」セクションの「*Command Restrictions*」を参照してください。
- デバッガの使用を禁止または制限できます。『*Natural Security*』ドキュメントの「*Components of an Environment Profile*」に説明があります。

動作要件

デバッガは、現在のNaturalシステムファイルの現在のライブラリに格納されている、カタログ化オブジェクトの実行時にのみ呼び出されます。 RUN コマンドを使用して、ワークエリアに含まれるソースコードを実行したときには、デバッガは*呼び出されません*。

効率的かつ正確にデバッグを行うには、ソースオブジェクトのソースコードが、カタログ化オブジェクトのコンパイル済みソースコードに対応している必要があります。これは、システムコマンド STOW を使用すると保証できます。 ソースオブジェクトを、カタログした*後*に変更した場合は、参照先のステートメントまたは変数が変更または削除されたために、デバッグエントリ（ブレイクポイントまたはウォッチポイント）が正しく機能しない可能性があります。 ソースオブジェクトのタイムスタンプが、対応するカタログ化オブジェクトのタイムスタンプより古いことをデバッガが検出すると、次の警告が表示されます。「Time stamps of source and cataloged object do not match.」

デバッガは、現在のライブラリまたはその steplib の 1 つに含まれる Natural オブジェクトをすべて調査します。 Natural システムライブラリ SYSLIB または SYSLIBS に格納された Natural オブジェクトは、デバッガにより調査されません。

デバッガは Natural バージョン 2.3 以上のオブジェクトにのみ適用できます。それより前のバージョンでカタログされた他の Natural オブジェクトには適用できません。

バッチ処理

デバッガは、主にオンラインモードでの対話式動作を目的として設計されていますが、デバッガコマンドは、ブレイクポイントまたはウォッチポイントの設定などのバッチ実行に使用することもできます。

注意: バッチ処理には制限があり、そのためにデバッガコマンドが拒否される場合があります。 例えば、デバッガは ++ コマンドおよび +4 コマンドをサポートしていません。

バッチでの統計の生成と出力の例

次に示すのは、デバッガのダイレクトコマンドをバッチモードで使用して、コール統計に関するレポートの生成と出力を行う例です。

```
//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
/*
```

デバッガの起動

手順 4.1. デバッガを起動するには

1 Natural オブジェクトまたはアプリケーションのデバッグ環境を設定します。

- Natural システムコマンド TEST を入力して、デバッグメインメニューを呼び出します。

  または
  実行中のアプリケーション内で、端末コマンド %<TEST を入力します。
- デバッグメインメニューの機能を使用して、Naturalオブジェクトまたはアプリケーションのデバッグエントリを指定します。

Debug environment maintenance
Spy maintenance
Breakpoint maintenance
Watchpoint maintenance
Call statistics maintenance
Statement execution statistics maintenance
Variable maintenance
List object source

2 デバッガを有効にします。

- コマンドプロンプトでコマンド TEST ON を入力します。

  または
  デバッグメインメニューでファンクションコード「T」を入力します。

3 Natural オブジェクトまたはアプリケーションを実行します。

デバッガは、指定したデバッグエントリでプログラム実行を中断し、［**Debug Break**］ウィンドウを呼び出します。

手順 4.2. エラー処理のためにデバッガを起動するには

- セッション開始時に、プロファイルパラメータ DBGERR を ON に設定します。

  『パラメータリファレンス』ドキュメントの「*DBGERR - ランタイムエラー発生時のデバッガの自動起動*」も参照してください。

  または:

  セッション中に、コマンドプロンプトでコマンド TEST ON を入力するか、デバッグメンテナンスのメインメニューでファンクションコード「T」を入力します。

  Natural エラーが発生すると、デバッガが［**Debug Break**］ウィンドウを呼び出します。

「*エラー処理*」セクションも参照してください。

デフォルトオブジェクト

該当する各セクションで説明したように、デバッガのメンテナンス機能は、対応するメニューの名前フィールドまたはダイレクトコマンドで指定したオブジェクトを参照します。 オブジェクト名を指定しない場合、デバッガはデフォルトで、現在のオブジェクトの名前を使用します。これは、デバッグメインメニューの右上隅にある［**Object**］フィールドに表示されています。 デフォルトのオブジェクトを指定すると、ダイレクトコマンドおよびメニューオプションを使用してブレイクポイントまたはウォッチポイントを指定するときに、オブジェクト名を指定する必要がなくなります。デフォルトオブジェクトを変更するには、「*コマンドの概要と構文*」セクションにあるコマンド SET の構文を参照してください。

5 テストモードのオンとオフの切り替え

以前に設定したデバッグ環境を有効にするには、テストモードを ON に設定する必要があります。

手順 5.1. テストモードを ON または OFF に設定するには

- デバッグメンテナンスのメインメニューで、ファンクションコード「T」を入力して、テストモードをオンまたはオフに切り替えます。

  または:

  次のいずれかのダイレクトコマンドを入力します。

  ```
  TEST ON
  ```

  または

  ```
  TEST OFF
  ```

テストモードを ON に設定して Natural オブジェクトを実行すると、デバッガは、すべてのデバッグエントリで必要なすべての動作を連続的にチェックしていきます。

テストモードを OFF に設定して Natural オブジェクトを実行すると、すべてのデバッグエントリが無視されます。

コマンド TEST、およびアプリケーション全体を Natural Security で保護できます。『*Natural Security*』ドキュメントの「*Library Maintenance*」セクションの「*Command Restrictions*」に説明があります。

6 デバッグ環境のメンテナンス

デバッグ環境は主にデバッグエントリで構成されているため、該当する各セクションで説明しているように、ブレイクポイントおよびウォッチポイントの設定により確立されます。

デバッグ環境を確立すると、後で使用できるように保存できます。デバッグ環境を保存するファイルは、デバッガコマンド PROFILE で指定できます（「*ナビゲーションと情報コマンド*」を参照）。 また、デバッグ環境を削除したり、そのカウンタを初期値にリセットしたりできます。

以下の項目もデバッグ環境の一部であり、したがって、デバッグ環境の保存やロードのたびに、これらの項目も保存またはロードされます。

- テストモードの設定（ON または OFF）
- デバッガコマンド PROFILE で設定可能なすべてのオプション（デバッグ環境のロードまたは保存に使用するファイルを除く）
- ［**Statement execution statistics maintenance**］機能の設定（ON、OFF、または COUNT）

注意: デバッガは、バージョン 2.3 より上の Natural バージョンで作成されたデバッグ環境のみをサポートします。それ以前のバージョンで作成されたデバッグ環境は無視されます。

手順 6.1. Debug Environment Maintenance 機能を呼び出すには

- デバッグメインメニューでファンクションコード「E」を入力します。

  または:

  次のダイレクトコマンドを入力します。

  ```
  EM
  ```

  ［**Debug Environment Maintenance**］メニューが表示されます。

このセクションでは、［**Debug Environment Maintenance**］メニューに用意されている機能と、さまざまなライブラリでメンテナンス機能を実行する手順を説明します。

各機能を選択したら、メンテナンスを行うデバッグ環境の名前を入力する必要があります。

テストモードを ON/OFF に設定

「*テストモードのオンとオフの切り替え*」を参照してください。

デバッグ環境のロード

手順 6.2. ユーザーシステムファイル（FUSER）からデバッグ環境をロードするには

- ［**Debug Environment Maintenance**］メニューで、ファンクションコード「L」および環境名を入力します。

  または:

  次のダイレクトコマンドを入力します。

  ```
  LOAD ENVIRONMENT name
  ```

  指定したデバッグ環境がロードされます。

環境名を指定しない場合は、「Noname」という名前のデフォルトの環境がロードされます。

アスタリスク（*）を入力すると、使用可能なすべてのデバッグ環境のリストが表示されます。リストでは、使用する環境に行コマンド LO でマークを付けて、それをデバッグバッファにロードできます。または、環境を削除するには、行コマンド DE を使用します。

デバッグ環境の保存

手順 6.3. デバッグ環境を保存するには

- ［**Debug Environment Maintenance**］メニューで、ファンクションコード「S」および環境名を入力します。

  または:

  次のダイレクトコマンドを入力します。

  ```
  SAVE ENVIRONMENT name
  ```

  指定した環境がリセットされ（次のセクションを参照）、デバッガコマンド PROFILE（「*ナビゲーションと情報コマンド*」を参照）で指定されたファイルの場所に保存されます。

環境名を指定しない場合は、「Noname」という名前で環境が保存されます。

指定した名前のデバッグ環境がすでに存在する場合は、その既存の環境を上書きするかどうかの確認を求められます。

デバッグ環境のリセット

デバッグ環境は、テスト実行を行う前に、毎回リセットする必要があります。 環境をリセットすると、次の結果となります。

- すべてのデバッグエントリの現在の状態が、初期状態に設定されます。
- すべてのイベントカウントが 0 に設定されます。
- 「*コール統計のメンテナンス*」セクションで説明しているように、デバッグバッファのコール統計がクリアされます。

手順 6.4. デバッグ環境をリセットするには

- ［**Debug Environment Maintenance**］メニューで、ファンクションコード「R」および環境名を入力します。

  または:

  次のダイレクトコマンドを入力します。

  ```
  RESET ENVIRONMENT name
  ```

  指定したデバッグ環境がリセットされます。

環境名を指定しない場合は、現在のデバッグ環境がリセットされます。

デバッグ環境の削除

手順 6.5. デバッグ環境を削除するには

1 ［**Debug Environment Maintenance**］メニューで、ファンクションコード「D」および環境名を入力します。

  または:

  次のダイレクトコマンドを入力します。

  ```
  DELETE ENVIRONMENT name
  ```

  確認ウィンドウが表示されます。

2 確認ウィンドウで「Y」（Yes）を入力して、削除を確認します。

  指定したデバッグ環境が削除されます。

環境名を指定しない場合は、現在のデバッグ環境が削除されます。

さまざまなライブラリのデバッグ環境のメンテナンス

SYSMAINユーティリティには、さまざまなライブラリやシステムファイルの間でデバッグ環境をコピーまたは移動する機能、およびデバッグ環境の削除、リスト表示、または名前変更を行う機能があります。

デバッグ環境が1つのライブラリから他のライブラリに移動またはコピーされても、そのブレイクポイントとウォッチポイントは、以前の（ソース）ライブラリを参照しています。 デバッグ環境を新しい（ターゲット）ライブラリに合わせて調整するには、対応するブレイクポイント（「*ブレイクポイントのメンテナンス*」の「*ブレイクポイントの変更*」も参照）またはウォッチポイント（「*ウォッチポイントのメンテナンス*」の「*ウォッチポイントの変更*」も参照）を変更します。 変更機能を実行するときに、既存の定義を変更する必要はありません。保存コマンド（PF5）の実行時に、ライブラリの参照は自動的に新しいライブラリに変更されます。これは、［**Modify Breakpoint**］画面または［**Modify Watchpoint**］画面にある［**Library**］フィールドのエントリで確認できます。

関連トピック：

- *デバッグ環境処理 - SYSMAIN ユーティリティ*、『ユーティリティ』ドキュメント

7 スパイのメンテナンス

この機能を使用して、すべてのデバッグエントリ（スパイ）、つまり、ブレイクポイントおよびウォッチポイントの有効化、無効化、リスト表示、または削除を行います。 また、ブレイクポイントまたはウォッチポイントの各メンテナンス画面にアクセスするために、［**Spy maintenance**］を使用することもできます。 これらの画面については、「*ブレイクポイントのメンテナンス*」および「*ウォッチポイントのメンテナンス*」で説明しています。

手順 7.1. Spy maintenance 機能を呼び出すには

- デバッグメインメニューでファンクションコード「S」を入力します。

  または:

  次のダイレクトコマンドを入力します。

  ```
  SM
  ```

  ［**Spy Maintenance**］メニューが表示されます。

［**Spy Maintenance**］メニューで提供される機能については、次のセクションで説明します。

テストモードを ON/OFF に設定

「*テストモードのオンとオフの切り替え*」を参照してください。

スパイの有効化

手順 7.2. 指定したスパイの現在の状態をアクティブに設定するには

- ［**Spy Maintenance**］メニューで、ファンクションコード「A」および、スパイ番号またはスパイ名を入力します。

  または:

  ダイレクトコマンド ACTIVATE を使用します。その構文については、「*コマンドの概要と構文*」セクションで説明しています。

スパイ番号もスパイ名も指定しない場合は、すべてのスパイ（ブレイクポイントおよびウォッチポイント）が有効になります。

スパイの無効化

手順 7.3. 指定したスパイの現在の状態を非アクティブに設定するには

- ［**Spy Maintenance**］メニューで、ファンクションコード「B」および、スパイ番号またはスパイ名を入力します。

  または:

  ダイレクトコマンド DEACTIVATE を使用します。その構文については、「*コマンドの概要と構文*」セクションで説明しています。

スパイ番号もスパイ名も指定しない場合は、すべてのスパイ（ブレイクポイントおよびウォッチポイント）が無効になります。

スパイの削除

手順 7.4. 指定したスパイを削除するには

- ［**Spy Maintenance**］メニューで、ファンクションコード「C」および、スパイ番号またはスパイ名を入力します。

  または:

  ダイレクトコマンド DELETE を使用します。その構文については、「*コマンドの概要と構文*」セクションで説明しています。

スパイ番号もスパイ名も指定しない場合は、すべてのスパイ（ブレイクポイントおよびウォッチポイント）が削除されます。

スパイの表示

手順 7.5. 指定したスパイを表示するには

- ［**Spy Maintenance**］メニューで、ファンクションコード「D」および、スパイ番号またはスパイ名を入力します。

  または:

ダイレクトコマンド DISPLAY を使用します。その構文については、「*コマンドの概要と構文*」セクションで説明しています。

指定したスパイが一意である場合は、［**Display Breakpoint**］画面または［**Display Watchpoint**］画面がスパイのタイプに応じて表示され、指定されているブレイクポイントまたはウォッチポイントがすべて表示されます。

指定したスパイが一意でない場合は、関連するスパイのリストが表示されます。 このリストで、スパイを行コマンド AC、DA、DI、MO、または DE でマークして、それぞれ、有効化、無効化、表示、変更、削除を実行できます。

スパイ番号もスパイ名も指定しない場合は、すべてのスパイ（ブレイクポイントおよびウォッチポイント）が表示されます。

スパイの変更

手順 7.6. 指定したスパイを変更するには

- ［**Spy Maintenance**］メニューで、ファンクションコード「M」および、スパイ番号または スパイ名を入力します。

  または:

  ダイレクトコマンド MODIFY を使用します。その構文については、「*コマンドの概要と構文*」セクションで説明しています。

  指定したスパイが一意の場合は、［**Modify Breakpoint**］画面または［**Modify Watchpoint**］画面がスパイのタイプに応じて表示され、ブレイクポイントまたはウォッチポイントを変更できます。

  指定したスパイが一意でない場合は、関連するスパイのリストが表示されます。 このリストで、スパイを行コマンド AC、DA、DI、MO、または DE でマークして、それぞれ、有効化、無効化、表示、変更、削除を実行できます。

スパイ番号もスパイ名も指定しない場合は、すべてのスパイ（ブレイクポイントおよびウォッチポイント）が表示され、選択したり、変更したりできます。

8 ブレイクポイントのメンテナンス

ブレイクポイントにより、Naturalオブジェクトの実行が特定のステートメント行で中断されます。このセクションでは、ブレイクポイントを設定する方法とタイミングについて説明します。ここで説明するメンテナンス機能は、オブジェクトソースから、［**List object source**］機能を使用して呼び出せます。

手順 8.1. Breakpoint Maintenance を呼び出すには

- デバッグメインメニューでファンクションコード「B」を入力します。

  または:

  次のダイレクトコマンドを入力します。

  ```
  BM
  ```

  ［**Breakpoint Maintenance**］メニューが表示されます。

このセクションでは、ブレイクポイントのメンテナンスの使用条件、［**Breakpoint Maintenance**］メニューに用意されている機能、およびブレイクポイント画面に含まれるフィールドと列について説明します。

使用条件

ブレイクポイントは、処理するNaturalオブジェクトの名前、およびそのオブジェクトのソースコード中でブレイクポイントが実行される行番号を指定して設定します。

一度指定されたブレイクポイントは、削除されない限り、その Natural セッション全体にわたり、設定されたままとなります。

ブレイクポイントは、ソースコード中の特定の行番号を参照します。 したがって、ブレイクポイントの設定後にソースコードを変更すると、ブレイクポイントが誤ったステートメントに適用される結果となり、そのために、Naturalオブジェクトが目的の位置で中断されなくなる可能性があります。 このプログラムループの問題は、これらのループ内にラベルを設定すると回避できます。 これらのラベルに設定されたブレイクポイントは、ステートメント行が挿入または削除されると、正しい行番号に調整されます。

ブレイクポイントの一意の識別子は、デバッガにより割り当てられたスパイ番号です。

ブレイクポイントは、コメント行、最初のステートメント行以外のステートメント行（1つのステートメントが複数のプログラム行にわたる場合）、以下のステートメントのいずれかのみを含む行には設定できません。

- AT BREAK OF
- AT END OF DATA

- AT END OF PAGE
- AT START OF DATA
- AT TOP OF PAGE
- BEFORE BREAK
- DECIDE
  バージョン 4.1 より前の Natural バージョン でカタログされたオブジェクトの場合は、WHEN 節および VALUE 節を含む行にのみブレイクポイントを設定できます。 これらの制限は、Natural バージョン 4.1 以上でカタログされたオブジェクトには適用されません。
- DEFINE SUBROUTINE
- DEFINE WINDOW
- FORMAT
- IF NO RECORDS FOUND
- ON ERROR
- OPTIONS

Natural Optimizer Compiler でコンパイルされた行にブレイクポイントを設定できるかどうかは、OPTIONS ステートメントの NODBG オプションにより異なります。これについては、『*Natural Optimizer Compiler*』ドキュメントの「*Switching on the Optimizer Compiler*」で説明しています。

テストモードを ON/OFF に設定

「*テストモードのオンとオフの切り替え*」を参照してください。

ブレイクポイントの有効化

手順 8.2. 指定したブレイクポイントの現在の状態をアクティブに設定するには

- ［**Breakpoint Maintenance**］メニューでファンクションコード「A」および、オブジェクト名と行番号のいずれかまたは両方を入力します。

  または:

  ダイレクトコマンド ACTIVATE を使用します。その構文については、「*コマンドの概要と構文*」セクションで説明しています。

  オブジェクト名も行番号も指定しない場合は、すべてのブレイクポイントが有効になります。

ブレイクポイントの無効化

手順 8.3. 指定したブレイクポイントの現在の状態を非アクティブに設定するには

- ［**Breakpoint Maintenance**］メニューでファンクションコード「B」および、オブジェクト名と行番号のいずれかまたは両方を入力します。

  または:

  ダイレクトコマンド DEACTIVATE を使用します。その構文については、「*コマンドの概要と構文*」セクションで説明しています。

  オブジェクト名も行番号も指定しない場合は、すべてのブレイクポイントが無効になります。

ブレイクポイントの削除

手順 8.4. 指定したブレイクポイントを削除するには

- ［**Breakpoint Maintenance**］メニューでファンクションコード「C」および、オブジェクト名と行番号のいずれかまたは両方を入力します。

  または:

  ダイレクトコマンド DELETE を使用します。その構文については、「*コマンドの概要と構文*」セクションで説明しています。

  オブジェクト名も行番号も指定しない場合は、すべてのブレイクポイントが削除されます。

ブレイクポイントの表示

手順 8.5. ブレイクポイントを表示するには

- ［**Breakpoint Maintenance**］メニューでファンクションコード「D」、オブジェクト名、および行番号を入力します。

  オブジェクト名を入力しない場合は、デフォルトオブジェクト（指定されている場合）が使用されます。

  または:

ダイレクトコマンド DISPLAY を使用します。その構文については、「*コマンドの概要と構文*」セクションで説明しています。

指定されたオブジェクトと行番号にブレイクポイントが設定されている場合は、下の例のような、すべてのブレイクポイントの定義を示す［**Display Breakpoint**］画面が表示されます。

```
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
```

一意のブレイクポイントがまったく見つからない場合は、次に説明する［**List Breakpoints**］画面が表示されます。

［**Display Breakpoint**］画面のフィールドについては、「*ブレイクポイント画面のフィールドと列*」で説明しています。

手順 8.6. ブレイクポイントのリストを表示するには

- ［**Breakpoint Maintenance**］メニューでファンクションコード「D」、オブジェクト名、または行番号を入力します。ABC* のように、アスタリスク（*）記号を使用すると、一定の範囲のオブジェクト名を指定できます。アスタリスク（*）のみを入力すると、すべてのオブジェクト名が選択されます。 オブジェクト名を入力しない場合は、デフォルトオブジェクト（指定されている場合）が使用されます。

  または:

ダイレクトコマンド DISPLAY を使用します。その構文については、「*コマンドの概要と構文*」セクションで説明しています。

下の例のような、指定されたオブジェクトまたは行番号に設定されたすべてのブレイクポイントのリストを含む［**List Breakpoints**］画面が表示されます。

```
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
```

このリストは、［**No.**］列のスパイ番号によって昇順で ソートされます。

［**List Breakpoints**］画面に含まれる列、およびリストの項目に対して実行できる行コマンドの詳細については、「*ブレイクポイント画面のフィールドと列*」を参照してください。

ブレイクポイントの変更

▶手順 8.7. ブレイクポイントを変更するには

1 ［**Breakpoint Maintenance**］メニューでファンクションコード「M」、オブジェクト名、および行番号を入力します。 オブジェクト名を入力しない場合は、デフォルトオブジェクト（指定されている場合）が使用されます。

または:

ダイレクトコマンド MODIFY を使用します。その構文については、「*コマンドの概要と構文*」セクションで説明しています。

一意のブレイクポイントが指定されている場合は［**Modify Breakpoint**］画面が表示され、ここでフィールドのエントリを変更できます。［**Modify Breakpoint**］画面のフィールドについては、「*ブレイクポイント画面のフィールドと列*」で説明しています。

一意のブレイクポイントがまったく見つからない場合は、［**List Breakpoints**］画面（「*ブレイクポイントの表示*」を参照）が表示されます。

2 ブレイクポイントを定義し終わったら、PF3 キー（Exit）または PF5 キー（Save）を押して、変更を保存します。デバッグエントリの有効性チェックの詳細については、「*メンテナンスと検証*」も参照してください。PF12 キー（Canc）を押した場合、ブレイクポイント定義は変更されないままとなります。

ブレイクポイントの設定

手順 8.8. セッションのブレイクポイントを追加するには

■ ［**Breakpoint Maintenance**］メニューでファンクションコード「S」および、オブジェクト名と行番号のいずれかまたは両方を入力します。

または:

ダイレクトコマンド SET を使用します。その構文については、「*コマンドの概要と構文*」セクションで説明しています。

オブジェクト名ではなく、有効な行番号を指定した場合は、デフォルトオブジェクト（「*デバッガの開始*」セクションを参照）の名前が使用されます。デフォルトオブジェクトが指定されていない場合は、現在のライブラリで使用可能なすべてのオブジェクトを示す選択ウィンドウが表示されます。

オブジェクト名と行番号を正しく指定した場合は通常、ブレイクポイントが設定され、直ちに確認されます。

ただし、コピーコードに設定されたブレイクポイントは、そのコピーコードを含むプログラムの実行時にのみ有効となります。デバッグエントリの有効性チェックの詳細については、「*メンテナンスと検証*」も参照してください。

ブレイクポイントがデフォルトコマンド（BREAK）を受け取るときは、その初期状態と現在の状態がアクティブに設定されており、実行に関する制限は指定されていません。ブレイクポイントの設定時にコマンド BREAK を削除し、ダイアログを表示するコマンドを何も指定しないと、プログラムの中断時にデバッガが制御を受け取れなくなります。

ブレイクポイント画面のフィールドと列

次の表では、［**Display Breakpoint**］画面または［**Modify Breakpoint**］画面のフィールドと、［**List Breakpoints**］画面の列について説明します。

フィールド	列	説明
Test Mode		テストモードが ON と OFF のいずれに設定されているかを示します。
Object		指定されている場合、デフォルトオブジェクトの名前を示します（「*デバッガの開始*」を参照）。
	Co	これは入力フィールドであり、次のいずれかの行コマンドを指定できます。 AC ブレイクポイントの有効化 DA ブレイクポイントの無効化 DI ブレイクポイントの表示 MO ブレイクポイントの変更 DE ブレイクポイントの削除 ? 有効な行コマンドのリストの表示 . ブレイクポイント画面の終了
Spy number	**No.**	ブレイクポイントの設定時にデバッガによって割り当てられる一意の数値。
Initial state	**Stat I**	ブレイクポイントの初期の状態と現在の状態として、アクティブ（A）または非アクティブ（I）を指定します。
Current state	**Stat C**	
Breakpoint name	**BP Name**	ブレイクポイントの名前。 有効な値は 1～12 文字です。 ブレイクポイントのデフォルト名は、オブジェクト名と行番号で構成されます。
DBID/FNR	**DBID**	Natural オブジェクトが保存されているシステムファイルのデータベース ID（DBID）とファイル番号（FNR）。
	FNR	
Library	**Library**	オブジェクトを含むライブラリの名前。
Object name	**Object**	現在のライブラリまたは STEPLIB のいずれかで使用できるオブジェクトの名前。
Line number	**Line**	オブジェクトのソースコード中のステートメントの行番号。上記の「*使用条件*」も参照してください。 BEG、END、または ALL も行番号として指定できます。 BEG オブジェクトで実行される最初のステートメントで、プログラム実行を中断するブレイクポイントを指定します。 BEG ブレイクポイントは、コピーコードには指定できません。

フィールド	列	説明
		END END または FETCH ステートメントなど、オブジェクトで実行される最後のステートメントで、プログラム実行を中断するブレイクポイントを指定します。 END ブレイクポイントは、コピーコードには指定できません。 ALL 実行ステートメントを含む各プログラム行で、プログラム実行を中断するブレイクポイントを指定します。
Label		オブジェクトのソースコード中で、処理ループを定義するステートメントに対して以前に設定されたラベルを参照します。上記の「*使用条件*」も参照してください。 有効な値は 1～32 文字です。
Skips before execution	**Skips**	ブレイクポイントが、対応するステートメント行が特定の回数実行されるまでは実行されないようにします。 有効な値は 0（デフォルト）～32767 です。
Max number executions	**Execs**	ゼロ（0）より大きい任意の数値で、ブレイクポイントの最大実行回数を指定します。 有効な値は 0（デフォルト）～32767 です。
Number of activations	**Count**	関連するステートメント行でブレイクポイントが有効化された回数を示します。 このカウンタは、プログラムがレベル 1 で起動されるとリセットされます。
Error in definition	**E**	プログラム実行中に、ブレイクポイント定義にあるステートメント行が、カタログ化オブジェクトに見つからないことを示します。 このエラーは、デバッグ中にオブジェクトのソースが変更され、再度カタログされると発生する場合があります。
Commands		6 個までのデバッグコマンド。 各行に 1 個のコマンドを入力します。 使用できるすべてのコマンドの概要については、「*コマンドの概要と構文*」を参照してください。 **注意:** ブレイクポイントの変更時にコマンド BREAK を削除し、ダイアログを表示するコマンドを何も指定しないと、プログラムの中断時にデバッガが制御を受け取れなくなります。

9 ウォッチポイントのメンテナンス

変数値が変わると常に、ウォッチポイントによって Natural オブジェクトの実行が中断されます。 また下記の「ウォッチポイント演算子」で説明しているように（「*ウォッチポイントの設定*」も参照）、特定の変数値に関連付けた条件に応じて中断させることもできます。

ウォッチポイントを使用すると、エラーがあるオブジェクトによって予期せず変更された変数を検出できます。

変数が変更されたと見なされるのは、変数の現在の値が、ウォッチポイントが前回トリガされたときに記録された値または初期値と異なる場合です。 ウォッチポイント値の比較検証は、253 バイトのフィールド長に制限されています。 最大長を超えるラージ変数の場合は、最初の 253 バイトのみが比較に使用されます。

ウォッチポイントは、Naturalオブジェクトの名前と適切な変数の名前を指定して定義します。

ウォッチポイントの一意の識別子は、デバッガによって割り当てられるスパイ番号です。

ウォッチポイントを指定すると、そのウォッチポイントを削除するまでNaturalセッション全体で有効になります。

手順 9.1. Watchpoint maintenance 機能を呼び出すには

- デバッグメインメニューでファンクションコード「W」を入力します。

  または:

  次のダイレクトコマンドを入力します。

  ```
  WM
  ```

  ［**Watchpoint Maintenance**］メニューが表示されます。

このセクションでは、［**Watchpoint Maintenance**］メニューの機能、およびウォッチポイント画面のフィールドと列について説明します。

テストモードを ON/OFF に設定

「*テストモードのオンとオフの切り替え*」を参照してください。

ウォッチポイントの有効化

手順 9.2. 指定したウォッチポイントの現在の状態をアクティブに設定するには

- ［**Watchpoint Maintenance**］メニューで、ファンクションコード「A」、オブジェクト名または変数名、あるいはその両方を入力します。

  または:

  ダイレクトコマンド `ACTIVATE` を使用します。その構文については、「*コマンドの概要とコマンド構文*」セクションで説明しています。

オブジェクトまたは変数を指定しない場合（または［**Variable**］フィールドでデフォルトのアスタリスクをそのまま使用する場合）は、すべてのウォッチポイントが有効になります。

ウォッチポイントの無効化

手順 9.3. 指定したウォッチポイントの現在の状態を非アクティブに設定するには

- ［**Watchpoint Maintenance**］メニューで、ファンクションコード「B」、オブジェクト名または変数名、あるいはその両方を入力します。

  または:

  ダイレクトコマンド `DEACTIVATE` を使用します。その構文については、「*コマンドの概要とコマンド構文*」セクションで説明しています。

オブジェクト名または変数を指定しない場合（または［**Variable**］フィールドでデフォルトのアスタリスクをそのまま使用する場合）は、すべてのウォッチポイントが無効になります。

ウォッチポイントの削除

手順 9.4. 指定したウォッチポイントを削除するには

- ［**Watchpoint Maintenance**］メニューで、ファンクションコード「C」、オブジェクト名または変数名、あるいはその両方を入力します。

  または:

ダイレクトコマンド DELETE を使用します。その構文については、「*コマンドの概要とコマンド構文*」セクションで説明しています。

オブジェクト名または変数を指定しない場合（または［**Variable**］フィールドでデフォルトのアスタリスクをそのまま使用する場合）は、すべてのウォッチポイントが削除されます。

ウォッチポイントの表示

手順 9.5. ウォッチポイントを表示するには

1 ［**Watchpoint Maintenance**］メニューで、ファンクションコード「D」、オブジェクト名または変数名、あるいはその両方を入力します。オブジェクト名を入力しない場合は、デフォルトオブジェクト（指定されている場合）が使用されます。

または:

ダイレクトコマンド DISPLAY を使用します。その構文については、「*コマンドの概要とコマンド構文*」セクションで説明しています。

指定されたオブジェクト名と変数名にウォッチポイントが設定されている場合は、下の例のような、すべてのウォッチポイントの定義を示す［**Display Watchpoint**］画面が表示されます。

```
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
```

［**Display Watchpoint**］画面のフィールドについては、「*ウォッチポイント画面のフィールドと列*」で説明します。

一意のウォッチポイントが見つからない場合は、［**List Watchpoints**］画面（下記を参照）が表示されます。

2 ［**Display Watchpoint**］画面では、ウォッチポイント演算子で指定したウォッチポイントの有効化の条件を表示できます（「*ウォッチポイント演算子*」も参照）。

演算子またはオペランド値、あるいはその両方を英数字形式で表示するには、PF10（Alpha）キーを押します。 .

または:

演算子またはオペランド値、あるいはその両方を 16 進形式で表示するには、PF11（Hex）キーを押します。

［**Commands**］フィールドがある［**Display Watchpoint**］画面のデフォルト表示に戻るには、PF22（Cmds）キーを押します。

手順 9.6. ウォッチポイントをリストするには

■ ［**Watchpoint Maintenance**］メニューで、ファンクションコード「D」、オブジェクト名または変数名を入力します。ABC*のように、アスタリスク（*）記号を使用すると、一定の範囲のオブジェクト名や変数名を指定できます。アスタリスク（*）のみを入力すると、すべての名前が選択されます。 オブジェクト名を入力しない場合は、デフォルトオブジェクト（指定されている場合）が使用されます。

または:

ダイレクトコマンド DISPLAY を使用します。その構文については、「*コマンドの概要とコマンド構文*」セクションで説明しています。

下の例のような［**List Watchpoints**］画面が表示されます。この画面には、指定したオブジェクト名または変数名に設定されたすべてのウォッチポイントがリストされます。

```
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
```

このリストは、［**No.**］ 列のスパイ番号によって昇順でソートされます。

［**List Watchpoints**］画面の列や、リスト項目に対して実行できる行コマンドの詳細については、「*ウォッチポイント画面のフィールドと列*」を参照してください。

ウォッチポイントの変更

手順 9.7. ウォッチポイントを変更するには

1 ［**Watchpoint Maintenance**］メニューで、ファンクションコード「M」、オブジェクト名および変数名を入力します。 オブジェクト名を入力しない場合は、デフォルトオブジェクト（指定されている場合）が使用されます。

または:

ダイレクトコマンド MODIFY を使用します。その構文については、「*コマンドの概要とコマンド構文*」セクションで説明しています。

一意のウォッチポイントが指定されている場合は、［**Modify Watchpoint**］画面が表示され、ここでフィールドのエントリを変更できます。 ［**Modify Watchpoint**］画面のフィールドについては、「*ウォッチポイント画面のフィールドと列*」で説明しています。

一意のウォッチポイントが見つからない場合は、［**List Watchpoints**］画面（「ウォッチポイントの表示」を参照）が表示されます。

2 ［**Modify Watchpoint**］画面では、ウォッチポイント演算子で指定したウォッチポイントの有効化の条件を変更できます（「*ウォッチポイント演算子*」も参照）。

演算子またはオペランド値、あるいはその両方を英数字形式で変更するには、PF10（Alpha）キーを押します。 .

または:

演算子またはオペランド値、あるいはその両方を 16 進形式で変更するには、PF11（Hex）キーを押します。

［**Commands**］フィールドがある［**Modify Watchpoint**］画面のデフォルト表示に戻るには、PF22（Cmds）キーを押します。

3 ウォッチポイント定義の編集を完了したら、PF3（Exit）または PF5（Save）キーを押して変更を保存します。 PF12（Canc）キーを押すと、ウォッチポイントは変更されないままになります。

ウォッチポイントの設定

手順 9.8. セッションのウォッチポイントを追加するには

■ ［**Watchpoint Maintenance**］メニューで、ファンクションコード「S」、オブジェクト名および変数名を入力します。

または:

ダイレクトコマンド SET を使用します。その構文については、「*コマンドの概要とコマンド構文*」セクションで説明しています。

または:

Natural オブジェクトの実行*前*：

- ［**List Object Source**］画面を表示します（「*オブジェクトソースのリスト*」を参照）。
- ［**Source**］列で、変数名にカーソルを置いて PF18（Se Wp）キーを押します。

オブジェクト名ではなく、有効な変数名を指定すると、デフォルトオブジェクトの名前が使用されます（「*デバッガの開始*」を参照）。 デフォルトオブジェクトが指定されていない場合は、現在のライブラリで使用可能なすべてのオブジェクトを示す選択ウィンドウが表示されます。

オブジェクト名と変数名が適切に指定されている場合、ウォッチポイントがすぐに設定され、対応する確認メッセージが画面に表示されます。 ダイナミック変数または X-array に設定したウォッチポイントは、プログラムの実行時にのみチェックされます。 デバッグエントリの有効性チェックの詳細については、「*メンテナンスと検証*」も参照してください。

ウォッチポイントはデフォルトコマンド（BREAK）を受信し、初期の状態と現在の状態がアクティブに設定され、実行に関する制限は指定されません。 ウォッチポイントの設定時にデフォルトコマンド BREAK を削除し、ダイアログを表示するコマンドを何も指定しないと、プログラムの中断時にデバッガが制御を受け取れなくなります。

このセクションでは、次のトピックについて説明します。

ウォッチポイント演算子

［Watchpoint Maintenance］画面で演算子と適切なオペランド（対応するオペランドがある場合）を指定することで、ウォッチポイントの有効化に対して条件を指定することができます。

手順 9.9. ウォッチポイント演算子を指定するには

1 英数字形式で演算子オペランドを指定する場合は、選択したウォッチポイントの［**Set Watchpoint**］または［**Modify Watchpoint**］画面で PF10（Alpha）キーを押します。

または:

16 進形式で演算子オペランドを指定する場合は、選択したウォッチポイントの［**Set Watchpoint**］または［**Modify Watchpoint**］画面で PF11（Hex）キーを押します。

2 つの入力フィールドが画面の下半分に表示されます。

2 左の入力フィールドには、次の表に示すウォッチポイント演算子のいずれかを入力します。

右の入力フィールドには、変数と比較するオペランド値（対応するオペランドがある場合）を入力します。 ダイナミック変数（英数字またはバイナリ）に対して指定した演算子があるウォッチポイントの場合、オペランド値は左から右へ比較されます。 ダイナミック変数のフィールド長は変化するため、最大253バイトを比較値として入力できます。ダイナミック変数の現在の長さが253バイトの最大比較長よりも短い場合、比較はダイナミック変数の現在の長さの範囲内でのみ実行されます。

演算子	説明
MOD	変更。 変数の内容が変更されるたびに、ウォッチポイントが有効になります。 これはデフォルト設定です。
EQ	等しい。 変数が変更され、変数の現在値が指定されたオペランド値と等しい場合、ウォッチポイントが有効になります。
NE	等しくない。 変数が変更され、変数の現在値が指定されたオペランド値と等しくない場合、ウォッチポイントが有効になります。
GT	より大きい。 変数が変更され、変数の現在値が指定されたオペランド値よりも大きい場合、ウォッチポイントが有効になります。
GE	より大きいまたは等しい。

演算子	説明
	変数が変更され、変数の現在値が指定されたオペランド値以上の場合、ウォッチポイントが有効になります。
LT	より小さい。 変数が変更され、変数の現在値が指定されたオペランド値よりも小さい場合、ウォッチポイントが有効になります。
LE	より小さいまたは等しい。 変数が変更され、変数の現在値が指定されたオペランド値以下の場合、ウォッチポイントが有効になります。
INV	無効な内容です。 タイプ N、P、D または T の変数に割り当てられた値が次の条件に一致しないとき、ウォッチポイントが有効になります。 N アンパック型数値。 P パック型数値。 D 1582-01-01 から 2700-12-31 までの日付。 T 1582-01-01 00:00:00.0 から 2700-12-31 23:59:59.9 までの時刻。

PF22（Cmds）キーを押すと、［**Commands**］入力フィールドのある［**Set Watchpoint**］または［**Modify Watchpoint**］画面のデフォルト表示に戻ることができます。

3 演算子の定義を保存するには、PF5（Save）キーを押します。

または:

演算子の定義を変更しないままで［**Modify Watchpoint**］画面を終了するには、PF12（Canc）キーを押します。

ウォッチポイント画面のフィールドと列

次の表では、［**Display Watchpoint**］または［**Modify Watchpoint**］画面のフィールドと、［**List Watchpoints**］画面の列について説明します。

フィールド	列	説明
Test Mode		テストモードが ON と OFF のいずれに設定されているかを示します。
Object		指定されている場合、デフォルトオブジェクトの名前を示します（「*デバッガの開始*」を参照）。
	Co	これは入力フィールドであり、次のいずれかの行コマンドを指定できます。 AC ウォッチポイントの有効化 DA ウォッチポイントの無効化

フィールド	列	説明
		DI ウォッチポイントの表示 MO ウォッチポイントの変更 DE ウォッチポイントの削除 ? 有効な行コマンドのリストの表示 . ウォッチポイント画面の終了
Spy number	**No.**	ウォッチポイントの設定時にデバッガによって割り当てられる一意の数値。
Initial state	**Stat I**	ウォッチポイントの初期の状態と現在の状態として、アクティブ（A）または非アクティブ（I）を指定します。
Current state	**Stat C**	
Watchpoint name	**WP Name**	ウォッチポイントの名前。 ウォッチポイントのデフォルト名は、関連する変数の名前です。 有効な値は 1～12 文字です。 フィールドサイズを超えた名前は、12 文字よりも後が切り捨てられます。 ［**List Watchpoints**］画面では、ウォッチポイント名が変数名の上の第 1 行に表示されます。
DBID/FNR	**DBID**	Natural オブジェクトが保存されているシステムファイルのデータベース ID（DBID）とファイル番号（FNR）。
	FNR	
Library	**Library**	オブジェクトを含むライブラリの名前。
Object name	**Object**	現在のライブラリまたは STEPLIB のいずれかで使用できるオブジェクトの名前。 システム変数をウォッチポイントとして指定する場合は、［**Object name**］フィールドにアスタリスク（*）を入力します。
Variable name		ユーザー定義のグローバル変数またはシステム変数の名前。 変数がグループの一部である場合、その変数には接頭辞としてグループ名が追加されます。 システム変数を指定する場合は、［**Object name**］フィールドにアスタリスク（*）を入力します。 配列に対しては、インデックスの記述を指定する必要があります。ウォッチポイントは、1 つの要素に対してのみ定義できます。 ［**List Watchpoints**］画面では、変数名がウォッチポイント名の下の第 2 行に表示されます。 詳細については、「変数のメンテナンス」も参照してください。
Skips before execution	**Skips**	ウォッチポイントに対して設定した条件を満たすまで、ウォッチポイントは実行しないと指定します（「ウォッチポイント演算子」も参照）。 有効な値は 0（デフォルト）～32767 です。

フィールド	列	説明
Max number executions	**Execs**	ゼロ（0）より大きい任意の数値で、ウォッチポイントの最大実行回数を指定します。 有効な値は 0（デフォルト）～32767 です。
Number of activations	**Count**	ウォッチポイント演算子を使用して変数に対して設定したウォッチポイント条件が満たされた回数を示します。 このカウンタは、プログラムがレベル 1 で起動されるとリセットされます。
Format/length		例えば A10 など、変数の Natural データフォーマットと長さ。
Persistent	**P**	ウォッチポイントをパーシスタントとしてマークします。パーシスタントウォッチポイントは、それらが定義された Natural オブジェクトに限定されず、下位のすべてのプログラムレベルに追加的に適用されます。 パーシスタントウォッチポイントは、`BY VALUE RESULT` ではなく、参照によってサブプログラムに渡される変数に対してのみ意味があります。『ステートメント』ドキュメントで、`CALLNAT` ステートメントの関連するパラメータについての説明（パラメータを示す「*operand2*」の項）を参照してください。 制限事項 パーシスタントウォッチポイントは、パラメータまたはコンテキスト節で定義された変数には使用できません。 有効な値は `Y`（Yes）または `N`（No）です。 `N` はデフォルトです。
Act. level		パーシスタントへの参照。 パーシスタントウォッチポイントが自動的に有効にされたプログラムレベルを示します。
Error in definition	**E**	無効なウォッチポイント定義を示します。 それぞれの変数定義が変更された後、デバッグ時に実行中のプログラムが再カタログされると、このエラーが発生する可能性があります。 ダイナミック変数または X-array（eXtensible 配列）に設定したウォッチポイントは、プログラムの実行時にのみチェックされます。
Commands		6 個までのデバッグコマンド。 各行に 1 個のコマンドを入力します。 使用できるすべてのコマンドの概要については、「*コマンドの概要と構文*」を参照してください。 **注意:** コマンド `BREAK` を削除し、ダイアログを表示するコマンドを何も指定しないと、プログラムの中断時にデバッガが制御を受け取れなくなります。

10 コール統計のメンテナンス

この機能は、アプリケーションの実行時に呼び出されたNaturalオブジェクトの統計情報や、オブジェクトが呼び出された回数の情報を取得するために使用します。 コール統計は、デバッグ環境のリセット後に削除されます。

手順 10.1. Call statistics maintenance 機能を呼び出すには

- デバッグメインメニューでファンクションコード「C」を入力します。

  または:

  次のダイレクトコマンドを入力します。

  ```
  CS
  ```

  ［**Call Statistics Maintenance**］メニューが表示されます。

［**Call Statistics Maintenance**］メニューの機能については、次のセクションで説明します。すべての出力機能については「*オブジェクトの出力*」で説明します。

テストモードを ON/OFF に設定

「*テストモードのオンとオフの切り替え*」を参照してください。

コール統計の ON/OFF の設定

［**Call statistics**］を ON に設定して Natural オブジェクトを実行すると、特定のオブジェクトに対するすべての呼び出しがカウントされて、処理結果の統計を後で表示または出力することができます。

手順 10.2. コール統計を ON または OFF に設定するには

- ［**Call Statistics Maintenance**］メニューで、ファンクションコード「C」を入力してコール統計を有効または無効に切り替えます。

  または:

  次のいずれかのダイレクトコマンドを入力します。

  ```
  SET CALL ON
  ```

  または

  ```
  SET CALL OFF
  ```

注意: ［**Call statistics**］機能をオフに切り替えてコール統計が作成されなかった場合、または、デバッグ環境のリセットによってコール統計が削除された場合、ステートメントの実行統計用に保存された情報（「*ステートメント実行統計のメンテナンス*」を参照）が表示に使用されます。 この情報により、アプリケーションの実行時に呼び出されなかったNatural オブジェクトを検出できます。

すべてのオブジェクトの表示

この機能では、ライブラリに含まれているすべてのオブジェクトの呼び出しの頻度が概要として表示されます。

手順 10.3. ライブラリにあるすべてのオブジェクトの呼び出しの頻度を表示するには

- ［**Call Statistics Maintenance**］メニューで、ファンクションコード「1」とライブラリ名を入力します。

  または:

  次のダイレクトコマンドを入力します。

  ```
  DISPLAY OBJECT library
  ```

  「*コマンドの概要と構文*」で DISPLAY の構文も参照してください。

  ライブラリ名を指定していない場合は、現在ログオンしているライブラリがデフォルトで使用されます。

  ［**Display Called Objects**］のサンプル画面に似ている［**Display Call Statistics**］画面が表示されます。

  ［**Display Call Statistics**］画面には、指定したライブラリ内のすべてのオブジェクトがリストされ、右側の［**Calls**］列には呼び出しの頻度が示されます。 FETCH または CALLNAT などの呼び出しステートメントごとに、オブジェクト名の付いたエントリとカウンタ変数がデバッグバッファに書き込まれます。 次に、対応するオブジェクトの呼び出しごとに、このカウンタが加算されていきます。

呼び出されたオブジェクトの表示

この機能で表示される画面は［**Display Call Statistics**］画面に対応していますが、呼び出されたオブジェクトのみが表示されます。

手順 10.4. ライブラリの呼び出されたオブジェクトを表示するには

- ［**Call Statistics Maintenance**］メニューで、ファンクションコード「2」とライブラリ名を入力します。

  または:

  次のダイレクトコマンドを入力します。

```
DISPLAY CALL library
```

  「*コマンドの概要と構文*」で DISPLAY の構文も参照してください。

  ［**Display Called Objects**］画面が表示されます。

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

ライブラリ名を指定していない場合は、現在ログオンしているライブラリがデフォルトで使用されます。

呼び出されなかったオブジェクトの表示

この機能で表示される画面は［**Display Call Statistics**］画面に対応していますが、*呼び出されなかった*オブジェクトのみが表示されます。

手順 10.5. 呼び出されなかったオブジェクトを表示するには

- ［**Call Statistics Maintenance**］メニューで、ファンクションコード「3」とライブラリ名を入力します。

または:

次のダイレクトコマンドを入力します。

```
DISPLAY NOCALL library
```

「*コマンドの概要と構文*」で DISPLAY の構文も参照してください。

ライブラリ名を指定していない場合は、現在ログオンしているライブラリがデフォルトで使用されます。

サンプル画面については、上記の「*呼び出されたオブジェクトの表示*」を参照してください。

オブジェクトの出力

出力機能では、コール統計の生成リストをプリンタに直接送信したり、リストを PC にダウンロードしたりすることができます。 デバッガの［**User Profile**］画面で出力デバイスとしてプリンタを指定します。 この画面を表示するには、デバッガコマンド PROFILE を使用します（「*ナビゲーションと情報コマンド*」を参照）。

ライブラリ名を指定していない場合は、現在ログオンしているライブラリがデフォルトで使用されます。

下記の「*出力オプション*」に示すように、いずれかの出力機能を実行するには、［**Call Statistics Maintenance**］メニューでファンクションコードを入力するか、ダイレクトコマンドを入力します。

出力オプション

出力機能	ファンクションコード	ダイレクトコマンド
すべてのオブジェクト	**4**	PRINT OBJECT *library*
呼び出されたオブジェクト	**5**	PRINT CALL *library*
呼び出されなかったオブジェクト	**6**	PRINT NOCALL *library*

「*コマンドの概要と構文*」で PRINT の構文も参照してください。

関連トピック：

- 「*バッチ処理*」の「*バッチでの統計の生成と出力の例*」

PC ダウンロードの例

サイトに Entire Connection と Natural Connection がインストールされている場合は、次の手順で説明するように、統計リストを PC にダウンロードできます。

▶手順 10.6. リストを PC にダウンロードするには

1 セッションの開始時に、次のようにプロファイルパラメータ PRINT を指定します。

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

2 セッションの開始後、次の端末コマンドを使用して PC 接続をアクティブにします

```
%+
```

3 デバッガを呼び出して有効にします。

4 デバッガコマンド PROFILE を入力して、［**User Profile**］画面を表示します（「*ナビゲーションと情報コマンド*」を参照）。

5 ［**User Profile**］画面の［**Output device**］フィールドで、現在のエントリを PCPRNT01 に変更し、PF3（Exit）キーを押してこの設定を保存します。

6 ［**Call Statistics**］機能を有効にして、デバッガで統計データを収集するアプリケーションを実行します。

7 統計画面で、出力機能を選択します。

表示された［Entire Connection］ウィンドウで、出力ファイルと PC ディレクトリを指定できます。

11 ステートメント実行統計のメンテナンス

この機能は、呼び出されたNaturalオブジェクトのどのステートメント行が実行されたかについての統計情報を取得します。 また、オブジェクトが呼び出された回数と、ステートメント行が実行された回数も示します。

ステートメントの実行統計は、以下の目的に使用することができます。

- アプリケーション内の無駄な（まったく実行されていない）プログラミングコードを検出する。
- アプリケーションテストの適用範囲を評価する（テストで1回も実行されなかったステートメント行がいくつあるか）。
- アプリケーションのパフォーマンスに影響を与える可能性がある、高い頻度で実行されるコードセグメントを特定する。

手順 11.1. Statement execution statistics maintenance 機能を呼び出すには

- デバッグメインメニューで、ファンクションコード「X」を入力します。

  または:

  次のダイレクトコマンドを入力します。

  ```
  XS
  ```

  ［**Statement Execution Statistics Maintenance**］メニューが表示されます。

［**Statement Execution Statistics Maintenance**］メニューの機能については、次のセクションで説明します。すべての出力機能については「*出力ステートメント*」で説明します。

テストモードを ON/OFF に設定

「*テストモードのオンとオフの切り替え*」を参照してください。

ステートメント実行統計の設定 ON/OFF/COUNT

この機能は、Natural オブジェクトの実行されたステートメント行に関する統計を有効にします。

このセクションでは、次のトピックについて説明します。

- 統計の有効化と無効化

セットアップオプション

［**Statement execution statistics**］を ON または COUNT に設定して、Natural オブジェクトを実行すると、特定のオブジェクト内で実行されたすべてのステートメント行が統計レポートにリストされます。

オプション ON を指定すると特定のステートメント行が実行されたかどうかのみが記録され、オプション COUNT を指定するとステートメント行が実行された回数がカウントされます。 ライブラリとオブジェクト名を指定して、ステートメントの実行統計を目的の Natural オブジェクトに制限することができます。 デフォルトでは、現在のライブラリにあるすべてのオブジェクトの統計が収集されます。 アスタリスク（*）表記を使用することもできます。

ステートメント実行統計を ON から COUNT に、またはその逆に切り替えても、既存の統計は影響を受けません。つまり、ON または COUNT のステータスは維持されます。

収集された統計データは、デバッグバッファに格納されます。 Natural オブジェクトの統計情報の保存に必要な容量は、次の式で計算します。

［**Statement execution statistics**］を ON に設定した場合：ソースの行数 / 8 + 100 バイト
［**Statement execution statistics**］を COUNT に設定した場合：ソースの行数 * 4 + 100 バイト

行の挿入または削除で Natural オブジェクトを変更し、STOW の実行前にオブジェクト行の行番号を再設定しなかった場合、オブジェクトの統計に必要な容量が大きくなる可能性があります。この問題を回避するには、エディタプロファイルで［**Auto Renumber**］を Y（Yes）に設定するか（『エディタ』ドキュメントの「*エディタプロファイル*」を参照）、［**Renumber source-codes lines**］オプションを有効（これがデフォルトです）にしてシステムコマンド CATALL を使用します（『システムコマンド』ドキュメントを参照）。

デバッガコマンド PROFILE（「*ナビゲーションと情報コマンド*」を参照）を使用して、デバッグバッファのサイズを制限できます。 ［**Statement execution statistics**］を COUNT に設定した場合、8000 ステートメント行を超えるオブジェクトに関しては、ステートメント実行統計が収集されません。

ステートメント実行統計はデバッグ環境の一部であり、したがって、ダイレクトコマンドの SAVE ENVIRONMENT と LOAD ENVIRONMENT の影響を受けます（「*デバッグ環境のメンテナンス*」も参照）。

統計の有効化と無効化

このセクションでは、ステートメント実行統計を有効または無効にする手順を説明します。

ライブラリまたはオブジェクト名、あるいはその両方を指定して、ステートメントの実行統計を目的の Natural オブジェクトに制限することができます。 デフォルトでは、現在のライブラリにあるすべてのオブジェクトの統計が収集されます。 アスタリスク（*）表記を使用することもできます。

手順 11.2. ステートメント実行統計を有効にするには

- ［**Statement Execution Statistics Maintenance**］メニューで、ファンクションコード「S」、ライブラリの名前またはオブジェクトの名前、あるいはその両方を入力します。 ［**State**］フィールドで値を ON に変更します。

  または:

  次のいずれかのダイレクトコマンドを入力します。

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

  または

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

  「*コマンドの概要と構文*」で SET の構文も参照してください。

ライブラリまたはオブジェクト、あるいはその両方を指定しない場合は、現在のライブラリにあるすべてのオブジェクトの統計データが有効になります。

手順 11.3. ステートメント実行統計を無効にするには

- ［**Statement Execution Statistics Maintenance**］メニューで、ファンクションコード「S」、ライブラリの名前またはオブジェクトの名前、あるいはその両方を入力します。 ［**State**］フィールドで値を OFF に変更します。

  または:

  次のダイレクトコマンドを入力します。

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

  「*コマンドの概要と構文*」で SET の構文も参照してください。

ライブラリまたはオブジェクト、あるいはその両方を指定しない場合は、現在のライブラリにあるすべてのオブジェクトの統計データが無効になります。

ステートメント実行統計の削除

手順 11.4. ステートメント実行統計を削除するには

■ ［**Statement Execution Statistics Maintenance**］メニューで、ファンクションコード「C」、ライブラリの名前またはオブジェクトの名前、あるいはその両方を入力します。

または:

次のダイレクトコマンドを入力します。

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

「*コマンドの概要と構文*」で DELETE の構文も参照してください。

ライブラリまたはオブジェクト、あるいはその両方を指定しない場合は、現在のライブラリにあるすべてのオブジェクトの統計データが削除されます。

ステートメント実行統計の表示

この機能では、指定したステートメント実行統計のリストの画面を表示します。

手順 11.5. ［List Statement Execution Statistics］画面を表示するには

1 ［**Statement Execution Statistics Maintenance**］メニューでファンクションコード「D」を入力します。

または:

次のダイレクトコマンドを入力します。

```
DISPLAY XSTATISTICS
```

［**List Statement Execution Statistics**］画面が表示されます。

```
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
```

各オブジェクトについて、以下の情報が表示されます。

- 呼び出しの頻度
- 実行ステートメントの数
- 実行されたステートメントの数
- 実行ステートメントの合計に対する実行されたステートメントの比率
- 実行されたステートメントの総数

データが欠けている場合、または整合性を失った可能性がある場合は、リストエントリが強調表示されます。

2 統計リストでは、さらに処理するために行コマンドで項目をマークすることができます。

行コマンド	説明
DE	上記で説明したように、ステートメント実行統計を削除します。
DS	すべてのステートメント行を表示します。
DX	実行されたステートメント行のみを表示します。
DN	実行されなかったステートメント行のみを表示します。
I	カタログ化オブジェクトとエラーの情報を表示します。
PS	すべてのステートメント行を出力します。
PX	実行されたステートメント行のみを出力します。
PN	実行されなかったステートメント行のみを出力します。

出力機能の詳細については、「*出力ステートメント*」を参照してください。

次のセクションでは、表示コマンドで表示できる画面について説明します。

- **すべてのステートメント行の表示**
- **実行されたステートメント行の表示**

- 実行されなかったステートメント行の表示

すべてのステートメント行の表示

［**Display Statement Lines**］画面には、オブジェクトソースが表示され、ステートメント行が実行されたかどうかが示されます。

手順 11.6. ［Display Statement Lines］画面を表示するには

■ ［**List Statement Execution Statistics**］画面で、行コマンド DS でエントリをマークします。

または:

次のダイレクトコマンドを入力します。

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

「*コマンドの概要と構文*」で DISPLAY の構文も参照してください。

［**Display Statement Lines**］画面が表示されます。［**Statement execution statistics**］を COUNT に設定した場合、下のサンプル画面で示すように、ステートメント行の実行頻度が表示されます。

```
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
```

一意のオブジェクトが指定されていない場合は、［**List Statement Execution Statistics**］画面が表示されます。

実行されたステートメント行の表示

［**Display Executed Statement Lines**］画面は［**Display Statement Lines**］画面に対応していますが、実行されたステートメント行のみが表示されます。

▶手順 11.7. ［Display Executed Statement Lines］画面を表示するには

■ ［**List Statement Execution Statistics**］画面で、行コマンド DX でエントリをマークします。

または:

次のダイレクトコマンドを入力します。

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

「*コマンドの概要と構文*」で DISPLAY の構文も参照してください。

一意のオブジェクトが指定されていない場合は、［**List Statement Execution Statistics**］画面が表示されます。

実行されなかったステートメント行の表示

［**Non-Executed Statement Lines**］画面は［**Display Statement Lines**］画面に対応していますが、実行されなかったステートメント行のみが表示されます。

▶手順 11.8. ［Display Non-Executed Statement Lines］画面を表示するには

■ ［**List Statement Execution Statistics**］画面で、行コマンド DN でエントリをマークします。

または:

次のダイレクトコマンドを入力します。

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

「*コマンドの概要と構文*」で DISPLAY の構文も参照してください。

一意のオブジェクトが指定されていない場合は、［**List Statement Execution Statistics**］画面が表示されます。

出力ステートメント

出力機能では、ステートメント実行統計の生成リストをプリンタに直接送信したり、リストをPC にダウンロードしたりすることができます。 デバッガの［**User Profile**］画面で出力デバイスとしてプリンタを定義します。 この画面を表示するには、デバッガコマンド `PROFILE` を使用します（「*ナビゲーションと情報コマンド*」を参照）。

ライブラリ名を指定していない場合は、現在ログオンしているライブラリがデフォルトで使用されます。

下記の「*出力オプション*」に示すように、いずれかの出力機能を実行するには、［**Statement Execution Statistics Maintenance**］メニューでファンクションコードを入力するか、［**Display Statement Lines**］画面で行コマンドを入力するか、あるいはダイレクトコマンドを入力します。

出力オプション

出力機能	ファンクションコード	行コマンド	ダイレクトコマンド
ステートメント実行統計の出力	**1**		`PRINT XSTATISTICS library (object)`
すべてのステートメントの出力	**2**	`PS`	`PRINT STATEMENT library (object)`
実行されたステートメントの出力	**3**	`PX`	`PRINT EXEC library (object)`
実行されなかったステートメントの出力	**4**	`PN`	`PRINT NOEXEC library (object)`

「*コマンドの概要と構文*」で `PRINT` の構文も参照してください。

関連トピック：

- 「*コール統計のメンテナンス*」の「*オブジェクトの出力*」にある「***PC ダウンロードの例***」
- 「*バッチ処理*」の「*バッチでの統計の生成と出力の例*」

12 変数のメンテナンス

この機能は、Naturalオブジェクトが中断されたときにデバッガ内で変数を表示および変更するために使用します。

［**Variable maintenance**］機能では、中断されたNaturalオブジェクトに関して、ユーザー定義変数、グローバル変数、データベース関連システム変数の*COUNTER、*ISN、*NUMBERに加えて、Naturalのデータフォーマット、データ長、内容が表示されます。

ユーザー定義変数、グローバル変数、およびデータベース関連システム変数の表示

このセクションでは、すべての変数のリストを表示する［**Display Variables**］（概要）画面や、特定の変数の詳細をすべて表示する［**Display Variable**］（詳細）画面を表示する手順を説明します。

- ［Display Variables］- 概要
- ［Display Variable］- 詳細

［Display Variables］- 概要

手順12.1.ユーザー定義変数、グローバル変数、およびデータベース関連システム変数の概要を表示するには

- デバッグメインメニューまたは［**Debug Break**］ウィンドウで、ファンクションコード「V」を入力します。

  または:

  次のダイレクトコマンドを入力します。

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

［**Display Variables**］（概要）画面には、中断されたNaturalオブジェクトに指定された変数のリストが表示されます。 長い値は、切り捨てた状態で画面に表示される場合があります。 配列の場合、最初のオカレンスの内容のみが表示されます。

変数の内容の表示形式を英数字と16進表記で切り替えるには、PF10（Alpha）キーを押すか、PF11（Hex）キーを押します。

切り捨てられた変数が表示される形式と、グループ名、変数名、インデックス（存在する場合）で構成された完全な名前が表示される形式で、表示を切り替えるには、PF5（Zoom）キーを押します。

［Display Variable］- 詳細

手順 12.2. 個々の変数を完全に表示するには

■ ［**Display Variables**］（概要）画面で、行コマンド DI で変数をマークして選択します。

または:

次のダイレクトコマンドを入力します。

```
DISPLAY VARIABLE variable
```

または:

［**List Object Source**］画面の［**Source**］列で、変数名にカーソルを置いて PF18（Di Va）キーを押します。

■ PF18（Di Va）キーを押す場合、次の制限が適用されます。

変数名（配列のオカレンスを含む）が複数行にまたがる場合は、最初の行の内容のみが評価されます。
array (3,2,6) のように、配列のインデックスが定数である場合は、このオカレンスのみが表示されます。
array (i,j) または array (3:i) のように、配列のインデックスが変数である場合、配列全体が表示されます。

［**Display Variable**］（詳細）画面が開いて、特定の変数に関連するすべての指定が表示されます。

256 バイトの長さを超えるラージ変数の場合、画面にはデフォルトで最初の 256 バイトが表示されます。

手順 12.3. 変数の内容全体を表示するには、または値の中を移動するには

■ 前のページに戻るには PF22 キーを、次のページに進むには PF23 キーを押します。

または:

特定の位置で表示を開始するには、［**Position**］フィールドに数値を入力します。

PF10（Alpha）キーと PF11（Hex）キーを押すと、変数の内容の表示形式を、英数字と 16 進表記の間で切り替えることができます。

手順 12.4. 画面の機能を使用して配列のすべてのオカレンスを表示するには

■ ［**Display Variables**］画面で、行コマンド DI で変数をマークして選択します。

または:

PF7（-）および PF8（+）キーを押して、個々のオカレンスの間でページを移動します。

手順 12.5. ダイレクトコマンドを使用して配列の 1 つ以上のオカレンスを表示するには

- 次のダイレクトコマンドを使用します。

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

variable-name は変数の名前です。*index-specification* は、次元のすべてのオカレンスに対して、インデックス表記、インデックス範囲、アスタリスク（*）のいずれかを示します。

例：

`DISPLAY VARIABLE ARRAY1(*)`	1 次元配列： 1 次元配列 ARRAY1 のすべてのオカレンスを表示します。

`DISPLAY VARIABLE ARRAY1(1)`
 	
または
 	
`DISPLAY VARIABLE ARRAY1`
	1 次元配列：
	
	1 次元配列 ARRAY1 の第 1 オカレンスを表示します。

`DISPLAY VARIABLE ARRAY2(2,3:4)`	2 次元配列： 2 次元配列 ARRAY2 で、1 次元の第 2 オカレンスと、2 次元のインデックス表記を表示します。
`DISPLAY VARIABLE ARRAY3(1,3:4,*)`	3 次元配列： 3 次元配列 ARRAY3 で、1 次元の第 1 オカレンス、2 次元のインデックス表記、3次元のすべてのオカレンスを表示します。

システム変数の表示

手順 12.6. システム変数（データベース関連システム変数を除く）を表示するには

- 次のダイレクトコマンドを入力します。

```
SYSVARS
```

［**System Variables**］画面が開いて、システム変数の制限セットが表示されます。

Handle タイプの変数の場合、Handle が参照するインスタンスのクラスの名前が、英数字表記で表示されます。 クラス名が使用できない場合、グローバルユニーク ID（GUID）が代わりに表示されます。 クラスがNatural内で定義された場合、クラス名またはGUIDに接尾辞の（NAT）が追加されます。

クラスインスタンスのプロパティの内容は、デバッガでは表示できません。

変数の変更

この機能は、システム変数には適用されません。

この機能は、ユーザー定義変数、グローバル変数、およびデータベース関連システム変数の値を変更するために使用します。

手順 12.7. ［Modify Variable］画面から変数の内容を変更するには

1 変数を行コマンド MO でマークして、［**Modify Variable**］画面を表示します。

または:

［**Display Variable**］画面で、PF5（Mod）キーを押します。

2 ［**Modify Variable**］画面の［**Contents**］フィールドで、変数の値を変更します。

デバッガ内では変数のフォーマットを変更できないので、新しい内容が、変更する変数のNatural データフォーマットに対して有効である必要があります。

［**Modify Variable**］画面では、PF10（Alpha）と PF11（Hex）キーを使用して、英数字と16進表記の間で、変数値の表示を切り替えることができます。

手順 12.8. ダイレクトコマンドで変数の内容を変更するには

■ 次のダイレクトコマンドを入力します。

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

変数値の変更を確認するメッセージが表示されます。

注意: ［**Modify Variables**］機能または MODIFY VARIABLE コマンドを Natural Security で禁止できます。『*Natural Security*』ドキュメントの「*Components of an Environment Profile*」に説明があります。

13 オブジェクトソースのリスト

この機能は、オブジェクトのソースコードを表示して、ブレイクポイントをメンテナンスするために使用します。［**List Object Source**］機能を使用するには、対応するソースが現在のライブラリまたは STEPLIB の 1 つに存在している必要があります。

手順 13.1. オブジェクトのソースコードをリストするには

■ デバッグメインメニューで、ファンクションコード「L」とオブジェクト名を入力します。

または:

次のダイレクトコマンドを入力します。

```
LIST object
```

「*コマンドの概要と構文*」で LIST の構文も参照してください。

［**List Object Source**］画面が開いてオブジェクトソースが表示され、画面の右側の［**Message**］列に、現在のブレイクポイントがすべてリストされます。

1 ページのスクロールアップまたはスクロールダウンを行うには、PF7（-）または PF8（+）キーを押します。

Natural オブジェクトを実行すると、デバッガは、設定された各ブレイクポイントまたはウォッチポイントで実行を中断し、［**Debug Break**］ウィンドウを表示します（「*デバッガの概念*」の「*［Debug Break］ウィンドウ*」を参照）。

手順 13.2. 中断された Natural オブジェクトのソースコードをリストするには

■ ［**Debug Break**］ウィンドウで、［**List break**］のファンクションコード L を選択します。

または:

必要な場合は、デバッガ画面で PF9（Li Br）キーを押すか次のダイレクトコマンドを入力します。

```
LIST BREAK
```

［**List Object Source**］画面が開いて、ブレイク（ブレイクポイントまたはウォッチポイント）が発生した位置で、オブジェクトのソースコードが表示されます。 ブレイクポイントまたはウォッチポイントの名前が、画面の右側の［**Message**］列に表示されます。 対応するソースコード行が強調表示されます。

ブレイクポイントのメンテナンス

［**List Object Source**］機能は、オブジェクトソース内からブレイクポイントのメンテナンス機能を呼び出したり、直接実行したりするために使用します。ブレイクポイントの設定の手順と、ブレイクポイントの全般的な情報については、「*ブレイクポイントのメンテナンス*」の「使用条件」を参照してください。

手順 13.3. オブジェクトソースからブレイクポイントのメンテナンス機能を呼び出すには

1 デバッグメインメニューで、ファンクションコード「L」とオブジェクト名を入力します。

または:

次のダイレクトコマンドを入力します。

```
LIST object
```

「*コマンドの概要と構文*」で LIST の構文も参照してください。

指定したオブジェクトのソースコードが表示されます。

すでに設定されたブレイクポイントの名前が、画面の右側の［**Message**］列に表示されます。

■ ソースリスト内で移動するには、コマンド行に次のいずれかのコマンドを入力します。

1 ページスクロールダウンまたはスクロールアップするには、+（プラス記号）または -（マイナス記号）

先頭にスクロールするには TOP

末尾にスクロールするには BOTTOM

左にスクロールするには LEFT

右にスクロールするには RIGHT

2 オブジェクトソース内で、目的の行を次に示すコマンドのいずれかでマークします。

行コマンド	説明
AC	ブレイクポイントを有効にします。
DA	ブレイクポイントを無効にします。
DE	ブレイクポイントを削除します。
DI	ブレイクポイントを表示します。
MO	［**Modify Breakpoint**］メンテナンス画面を表示します。
SE	ブレイクポイントを設定します。
SM	［**Set Breakpoint**］メンテナンス画面を表示します。

コマンドを正常に実行すると、対応するメッセージが画面の右側にある［**Message**］列に表示されます。

14 エラー処理

このセクションでは、デバッガの使用時にエラーを処理する方法について説明します。

アプリケーション実行時のエラー

デバッガを使用して、プログラムの実行を中断させたNaturalシステムエラーを分析できます。テストモードをONに設定した場合（「*テストモードのオンとオフの切り替え*」を参照）、または、DBGERRをONに設定した場合（『パラメータリファレンス』ドキュメントを参照）、エラーが発生するとデバッガが制御を取得します。 この場合、次の例のような［**Debug Break**］ウィンドウが表示されます。

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

［**List break**］機能を使用すると、最後のステートメントを実行した位置で、プログラムのソースコードを表示できます。Naturalエラー番号が画面の右側にある［**Message**］列に表示され、対応するソースコード行が強調表示されます。

この時点で、例えばプログラム内で変数の内容を確認し、エラーの原因を特定できます。

デバッガ実行時のエラー

アプリケーションのデバッグ時にエラーが検出されると、デバッガは実行を停止して、次に示す例のようなエラーメッセージのウィンドウを表示します。

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

N（No - これがデフォルト設定です）でこのエラーメッセージを確認すると、次のような操作が行われます。

- デバッガはデバッグを停止して、テストモードを OFF に設定します。
- Natural ランタイムシステムはエラーを無視して、アプリケーションの実行を続けます。

Y（Yes）でこのメッセージを確認すると、次のような操作が行われます。

- デバッガはデバッグを停止して、テストモードを OFF に設定します。
- Natural ランタイムシステムはエラーに対処して、エラーをアプリケーションに渡します。

  ON ERROR ステートメントが使用されている場合（『ステートメント』ドキュメントを参照）、アプリケーションは実行時のエラーが発生した後の対処方法を決定します。 例えば、トランザクションがデータベースからバックアウトされる NAT3009 の場合、アプリケーションは適切なアクションを実行できます。

  ON ERROR ステートメントが使用されていない場合、Natural ランタイムシステムは、アプリケーションの実行を停止し、Natural コマンドプロンプトを表示します。

15 実行制御コマンド

このセクションでは、デバッグセッションでプログラムフローを制御するためにデバッガで提供されているダイレクトコマンドについて説明します。 デバッガで使用できるすべてのコマンドの概要については、「*コマンドの概要と構文*」を参照してください。

次に挙げるコマンドは、デバッガがプログラムの実行を中断する場合にのみ適用されます。

ESCAPE BOTTOM

このコマンドを使用できるのは、処理ループ内でNaturalオブジェクトが中断された場合のみです。

このコマンドを入力すると、中断された Natural オブジェクトが、処理ループに続く最初のステートメントで続行されます。

注意: このコマンドを Natural Security で禁止できます。『*Natural Security*』ドキュメントの「*Components of an Environment Profile*」に説明があります。

ESCAPE ROUTINE

このコマンドを入力すると、中断されたNaturalオブジェクトの処理が停止されて、中断されたNaturalオブジェクトを呼び出したオブジェクトで処理が続行されます。この処理は、対応する `CALLNAT`、`PERFORM`、または `FETCH RETURN` ステートメントに続くステートメントで続行されます。

コマンド `ESCAPE ROUTINE` をメインプログラムに適用すると、Natural はプログラムを終了してコマンドモードに戻ります。

注意: このコマンドを Natural Security で禁止できます。『*Natural Security*』ドキュメントの「*Components of an Environment Profile*」に説明があります。

EXIT

デバッグメインメニューを表示しているときに終了機能を実行するには、PF3（Exit）キーを押すか、実行制御コマンド `EXIT` を入力します。デバッガをダイレクトコマンド `TEST` で呼び出した場合、デバッガは呼び出し元のプログラム（つまり、次に続行される中断された Natural オブジェクト）に戻るか、コマンドプロンプトに戻ります。また、端末コマンド `%<TEST` で呼び出した場合は、対応する入力フィールドに戻ります。 ただし、ブレイクポイントまたはウォッチポイントが現在アクティブである場合、このブレイクポイントまたはウォッチポイントの次のコマンドが実行されます。

デバッグメインメニューが表示されていないときに、ダイレクトコマンド EXIT を入力したり、PF3（Exit）キーを押したりすると、現在の機能が終了されて、デバッグセッションの前の手順に戻ります。

GO

ダイレクトコマンド GO を入力すると（または PF14 キーを押すと）、デバッガは中断された Natural オブジェクトの実行に制御を戻します。 Natural オブジェクトが中断された時点でブレイクポイントまたはウォッチポイントがアクティブであった場合、このブレイクまたはウォッチポイントの残りのコマンドは*実行されません*。

NEXT

ダイレクトコマンド NEXT を入力すると（または PF13 キーを押すと）、ブレイクポイントまたはウォッチポイントに指定した次のコマンドが実行されます。 その他にコマンドを指定していない場合、プログラムの実行が続行されます。

RUN

ダイレクトコマンド RUN を入力すると、テストモードがオフに切り替えられ、プログラムの実行が続行されます。ブレイクポイントとウォッチポイントがそれ以上調べられることはありません。

STEP

ダイレクトコマンド STEP を入力すると、*n* 実行ステートメントに対して、中断された Natural オブジェクトが続行されます。 *n* のデフォルト値は 1 です。

STEP SKIPSUBLEVEL

別のオブジェクトを呼び出すステートメント（例えば、CALLNAT）に対してダイレクトコマンド STEP SKIPSUBLEVEL を入力すると、呼び出されたオブジェクト内で最初に実行されたステートメントではなく、現在のオブジェクト内にある次の実行ステートメントで処理が続行されます。

このコマンドが別のオブジェクトを呼び出さないステートメントに適用されると、デバッガは、コマンド STEP が入力された場合と同様に動作します。

STEP SKIPSUBLEVEL n

コマンド STEP SKIPSUBLEVEL では、上位レベルの数値 *n* を指定できます。 ステップモードは、指定されたレベルの次のオブジェクト内で続行されます。 例：レベル 4 のオブジェクトに STEP SKIPSUBLEVEL 2 を入力すると、レベル 2 のオブジェクトでステップモードが続行されます。

「*ナビゲーションと情報コマンド*」で説明しているように、オブジェクトレベル情報は、コマンド OBJCHAIN で取得することができます。

STOP

ダイレクトコマンド STOP を入力すると、デバッガと中断された Natural オブジェクトが両方とも終了されます。

注意: このコマンドを Natural Security で禁止できます。『*Natural Security*』ドキュメントの「*Components of an Environment Profile*」に説明があります。

16 ナビゲーションと情報コマンド

このセクションでは、デバッグエリアでの移動、画面表示のスクロール、オブジェクトや変数に関するさまざまな情報の取得、プロファイルの指定を行うために、デバッガで使用できるダイレクトコマンドについて説明します。デバッガで使用できるすべてのコマンドの概要については、「*コマンドの概要と構文*」を参照してください。

BREAK

コマンド BREAK は、新しいデバッグエントリの作成時に自動的に設定されるデフォルトコマンドです。 このコマンドは［**Debug Break**］ウィンドウを表示します。このウィンドウについては、「*デバッガの概念*」の「*［Debug Break］ウィンドウ*」で説明しています。

コマンド BREAK が対応するデバッグエントリの変更で削除されると、［**Debug Break**］ウィンドウが表示されなくなります。 ただし、指定されたその他のコマンドは実行され、イベントのカウントが加算されます。

FLIP

コマンド FLIP は、2 つの PF キー行（PF1～PF12 および PF13～PF24）の表示を相互に切り替えます。

LAST

コマンド LAST は、最後に入力されたコマンドを表示します。 最後の 3 つのコマンドは保存されており、再度呼び出すことができます。

OBJCHAIN

コマンド OBJCHAIN を使用できるのは、Natural オブジェクトが中断された場合のみです。

このコマンドは、現在のレベルと上位の全レベルでオブジェクトを表示し、可能な場合は、現在の GDA（グローバルデータエリア）も表示して、中断に関する情報を提供します。

ON/OFF

コマンド ON または OFF をデバッガに入力すると、テストモードがオンまたはオフに切り替わります。 「*TEST ON/OFF*」も参照してください。

PROFILE

コマンド PROFILE は、デバッガのプロファイルを変更できる［**User Profile**］画面を表示します。

［User Profile］画面

［**User Profile**］画面には、次のオプションがあります。

オプション	説明
Reset debug environment automatically on exit	デバッガの終了時に現在のデバッグ環境を自動的にリセットすると指定します。デフォルトは N（No）です。
File for loading/saving debug environments	保存先、またはロード元のシステムファイルデバッグ環境を指定します。値は、FUSER（デフォルト）、FNAT、SPAD（スクラッチパッドファイル）のいずれかです。
Confirm EXIT/CANCEL before execution	実行前に EXIT または CANCEL コマンドの確認を指定します。 デフォルトは N（No）です。
Stack unknown commands	入力された不明なデバッグコマンド（例えば、呼び出されたプログラムの名前）をスタックするかどうかを指定します。 スタックする場合、不明なデバッグコマンドを入力すると、すぐにデバッガが終了されて、コマンドが実行されます。このオプションを指定していない場合、不明なデバッグコマンドが検出されると、対応するエラーメッセージが表示されます。 デフォルトは Y（Yes）です。
Output device	［**Call statistics maintenance**］機能（「*オブジェクトの出力*」を参照）と［**Statement execution statistics maintenance**］機能（「*出力ステートメント*」を参照）のためにプリンタを指定します。 デフォルト値は HARDCOPY です。 出力を別のプリンタに転送する場合は、HARDCOPY を Natural システム管理者が提供する有効なプリンタ名に変更します。
Maximum debug buffer size in KB	デバッグバッファの最大サイズ（単位はKB）を指定します。デバッグバッファは必要に応じて自動的に拡張されますが、拡張は指定された最大サイズに限定されます。 無制限を表す 0 を指定するか、4～16384 の値を入力します（4 の倍数にする必要があります）。 この制限を超えた場合、それ以上のデバッグエン

オプション	説明
	トリを定義できなくなり、その他のコールまたはステートメント実行統計のエントリも生成されなくなります。

SCAN

［List object source］機能にのみ適用されます（「*オブジェクトソースのリスト*」を参照）。

このコマンドは、オブジェクトソース内で文字列を検索します。

- SCAN は、空白または英数字以外の文字で区切られた指定値を検索します。
- SCAN ABS は、値が他のどのような文字に囲まれている場合でも、ソースコード内で、指定された値の完全スキャンを実行します。

「*コマンドの概要と構文*」で構文図も参照してください。

SCREEN

Natural オブジェクトの中断に対してコマンド SCREEN を入力すると、中断された Natural オブジェクトの現在の画面出力が表示されます。 Enter キーを押すと、デバッグモードに戻ります。

SET OBJECT

「*デバッガの開始*」にある関連セクションで説明しているように、コマンド SET OBJECT は、デフォルトオブジェクトの名前を変更します。 「*コマンドの概要と構文*」で SET の構文も参照してください。

STACK

コマンド STACK を入力すると、Natural スタックの最上位にあるエントリの内容が表示されます。 個々の最上位エントリ要素を最大 15 個まで表示できます。 55 文字を超える要素は切り捨てられ、アスタリスク（*）でマークされます。

注意: 要素が 1 つでも 249 文字を超えた場合は、エラーメッセージが表示されます。

SYSVARS

このコマンドを入力すると、システム変数の制限セットの現在値が表示されます。

TEST ON/OFF

コマンド `TEST ON` または `TEST OFF` は、テストモードをそれぞれオンまたはオフに切り替えます。 上記で説明したように、デバッガでは単に `ON` または `OFF` を入力します。

注意: `TEST` コマンドを Natural Security で禁止できます。『*Natural Security*』ドキュメントの「*Library Maintenance*」セクションの「*Command Restrictions*」に説明があります。

17 コマンドの概要と構文

このセクションでは、デバッグ機能を直接実行したり、デバッガ画面を操作したりするすべてのデバッガコマンドについて説明します。

ユーザー定義オペランドを指定した複雑なコマンド構造の詳細については、下記の「*構文図*」を参照してください。

すべてのデバッグコマンド

次の表に示すデバッグコマンドは、任意のデバッガ画面のコマンド行に入力することができます。 デバッグコマンドまたはサブコマンドの下線部分は最短の省略形を表します。

コマンド	サブコマンド	説明
-		リストを 1 ページ下方にスクロールします。
--		リストの先頭までスクロールします。
TOP		
+		リストを 1 ページ下方にスクロールします。
++		リストの末尾までスクロールします。
BOTTOM		
ACTIVATE （下記の構文）	BREAKPOINT または BP	「*ブレイクポイントのメンテナンス*」で説明しているように、ブレイクポイントを有効にします。
	SPY	ブレイクポイントおよびウォッチポイントを有効にします。「*スパイのメンテナンス*」の「スパイの有効化」も参照してください。
	WATCHPOINT または WP	「*ウォッチポイントのメンテナンス*」で説明しているように、ウォッチポイントを有効にします。
BM		「*ブレイクポイントのメンテナンス*」で説明した［**Breakpoint Maintenance**］メニューを表示します。
BREAK		［**Debug Break**］ウィンドウを表示します。「*ナビゲーションと情報コマンド*」の「BREAK」も参照してください。
CANCEL		現在の操作をキャンセルし、変更を保存しないで画面を閉じます。
DBLOG	A または Q または	デバッガ内から DBLOG ユーティリティを呼び出します（『ユーティリティ』ドキュメントを参照）。 データベース環境を指定するには、次のいずれかのサブコマンドを使用します。 ■ A = Adabas（これがデフォルトです）

コマンド	サブコマンド	説明
	D	■ Q = SQL ■ D = DL/I **注意:** デバッグの中断時には、上記に示したサブコマンドのいずれか 1 つを指定できます。
DEACTIVATE または DA （下記の構文）	BREAKPOINT または BP	「*ブレイクポイントのメンテナンス*」で説明しているように、ブレイクポイントを無効にします。
	SPY	ブレイクポイントおよびウォッチポイントを無効にします。「*スパイの無効化*」も参照してください。
	WATCHPOINT または WP	「*ウォッチポイントのメンテナンス*」で説明しているように、ウォッチポイントを無効にします。
DELETE （下記の構文）	BREAKPOINT または BP	「*ブレイクポイントのメンテナンス*」で説明しているように、ブレイクポイントを削除します。
	SPY	ブレイクポイントおよびウォッチポイントを削除します。「*スパイの削除*」も参照してください。
	WATCHPOINT または WP	「*ウォッチポイントのメンテナンス*」で説明しているように、ウォッチポイントを削除します。
	ENVIRONMENT	指定したデバッグ環境を削除します。「*デバッグ環境の削除*」も参照してください。
DISPLAY （下記の構文）	BREAKPOINT または BP	「*ブレイクポイントのメンテナンス*」で説明しているように、ブレイクポイントを表示します。
	SPY	ブレイクポイントおよびウォッチポイントを表示します。「*スパイの表示*」も参照してください。
	WATCHPOINT または WP	「*ウォッチポイントのメンテナンス*」で説明しているように、ウォッチポイントを表示します。
	CALL	アプリケーションの実行時に呼び出した Natural オブジェクトの統計を表示します。「*呼び出されたオブジェクトの表示*」も参照してください。

コマンド	サブコマンド	説明
	EXEC	呼び出された Natural オブジェクトの実行されたステートメント行の統計を表示します。「*実行されたステートメント行の表示*」も参照してください。
	HEXADECIMAL	変数の内容を 16 進形式で表示します。
	NOCALL	アプリケーションの実行時に呼び出されなかった Natural オブジェクトの統計を表示します。「*呼び出されなかったオブジェクトの表示*」も参照してください。
	NOEXEC	呼び出された Natural オブジェクトの実行されなかったステートメント行の統計を表示します。「*実行されなかったステートメント行の表示*」も参照してください。
	OBJECT	オブジェクトの呼び出しの頻度に関する統計を表示します。「*すべてのオブジェクトの表示*」も参照してください。
	STATEMENT	呼び出された Natural オブジェクトの実行されたステートメント行と実行されなかったステートメント行の統計を表示します。「*すべてのステートメント行の表示*」を参照してください。
	VARIABLE	「変数のメンテナンス」で説明しているように、中断された Natural オブジェクトの変数を表示します。
	XSTATISTICS	実行統計の概要を表示します。「*ステートメント実行統計の表示*」も参照してください。
EM		「*デバッグ環境のメンテナンス*」で説明しているように、［**Debug Environment Maintenance**］メニューを表示します。
ESCAPE	BOTTOM	ループの処理を停止し、ループ後の最初のステートメントにエスケープします。「*実行制御コマンド*」の「ESCAPE BOTTOM」を参照してください。
	ROUTINE	中断された Natural オブジェクトの処理を停止し、可能な場合は、別のオブジェクトで処理を続行します。「*実行制御コマンド*」の「ESCAPE ROUTINE」を参照してください。
EXIT		現在の画面を終了します。「*実行制御コマンド*」の「EXIT」を参照してください。
FLIP		2 つの PF キー行（PF1～PF12 および PF13～PF24）の表示を相互に切り替えます。
GO		中断された Natural オブジェクトの実行に制御を戻します。「*実行制御コマンド*」の「GO」を参照してください。
LAST		最後に入力されたコマンドを表示します。 最後の 3 つのコマンドは保存されており、再度呼び出すことができます。
LEFT		ソースコードリストの左側に移動します。
LIST （下記の構文）		オブジェクトのソースコードを表示します。
	BREAK	現在のブレイクでオブジェクトソースを表示します。 関連するステートメント行が強調表示されます。
	LASTLINE	現在のブレイクの前に最後に実行された行で、オブジェクトソースを表示します。

コマンド	サブコマンド	説明
LOAD （下記の構文）	ENVIRONMENT	指定されたデバッグ環境をロードします。「*デバッグ環境のロード*」を参照してください。
MENU		デバッグメインメニューを表示します。
MODIFY （下記の構文）	BREAKPOINT または BP	「*ブレイクポイントのメンテナンス*」で説明しているように、ブレイクポイントを変更します。
	SPY	［**Modify Breakpoint**］または［**Modify Watchpoint**］画面を表示します。「*スパイのメンテナンス*」の「*スパイの変更*」も参照してください。
	WATCHPOINT または WP	「*ウォッチポイントのメンテナンス*」で説明しているように、ウォッチポイントを変更します。
	HEXADECIMAL	16 進形式で示された変数の内容を変更します。
	VARIABLE	「*変数の変更*」で説明しているように、変更のために［**Display Variable**］画面を表示します。
NEXT		ブレイクポイントまたはウォッチポイントに指定された次のコマンドを実行します。
OBJCHAIN		さまざまなプログラムレベルで実行されたオブジェクトを表示します。「*ナビゲーションと情報コマンド*」の「OBJCHAIN」を参照してください。
ON または OFF		テストモードをオンまたはオフに切り替えます。「*テストモードのオンとオフの切り替え*」も参照してください。
PRINT （下記の構文）	CALL	アプリケーションの実行時に呼び出した Natural オブジェクトの統計を出力します。「*呼び出されたオブジェクトの表示*」も参照してください。
	EXEC	呼び出された Natural オブジェクトの実行されたステートメント行の統計を出力します。「*実行されたステートメント行の表示*」も参照してください。
	NOCALL	アプリケーションの実行時に呼び出されなかった Natural オブジェクトの統計を出力します。「*呼び出されなかったオブジェクトの表示*」も参照してください。
	NOEXEC	呼び出された Natural オブジェクトの実行されなかったステートメント行の統計を出力します。「*実行されなかったステートメント行の表示*」も参照してください。
	OBJECT	オブジェクトの呼び出しの頻度に関する統計を出力します。「*すべてのオブジェクトの表示*」も参照してください。

コマンド	サブコマンド	説明
	STATEMENT	呼び出された Natural オブジェクトの実行されたステートメント行と実行されなかったステートメント行の統計を出力します。「*すべてのステートメント行の表示*」も参照してください。
	XSTATISTICS	実行されたステートメント行の統計を出力します。「*ステートメント実行統計の表示*」も参照してください。
PROFILE		「*ナビゲーションと情報コマンド*」で説明しているように、デバッガのプロファイルを変更できる［**User Profile**］画面を表示します。
RESET （下記の構文）	ENVIRONMENT	現在のデバッグ環境をリセットします。「*デバッグ環境のリセット*」を参照してください。
RIGHT		ソースコードリストの右側に移動します。
RUN		テストモードをオフに切り替えて、プログラムの実行を続行します。
SAVE （下記の構文）	ENVIRONMENT	現在の環境をリセットして、デバッグの指定を保存します。「*デバッグ環境の保存*」も参照してください。
SCAN	ABS	［**List object source**］機能の使用時にのみ適用されます（「*オブジェクトソースのリスト*」を参照）。 オブジェクトのソースコードにある値を検索します。下記の「*ナビゲーションと情報コマンド*」と「*構文図*」の「SCAN」を参照してください。
SCREEN		オブジェクトの中断時に入力すると、中断された Natural オブジェクトの現在の画面出力が表示されます。Enter キーを押すと、デバッグモードに戻ります。
SET （下記の構文）	BREAKPOINT または BP	「*ブレイクポイントのメンテナンス*」で説明した［**Set Breakpoint**］画面を表示します。
	CALL ON または CALL OFF	「*コール統計のメンテナンス*」で説明しているように、コール統計を有効または無効にします。
	OBJECT	デバッガに対して指定されたデフォルトのオブジェクトを変更します。「*ナビゲーションと情報コマンド*」の「SET OBJECT」も参照してください。
	WATCHPOINT または WP	「*ウォッチポイントのメンテナンス*」で説明した［**Set Watchpoint**］画面を表示します。
	XSTATISTICS ON または	「*ステートメント実行統計の設定*」で説明しているように、ステートメント実行統計を有効（ON または COUNT）または無効（OFF）にします。

コマンド	サブコマンド	説明
	XSTATISTICS COUNT または XSTATISTICS OFF	
SM		「*スパイのメンテナンス*」で説明した［**Spy Maintenance**］メニューを表示します。
STACK		Natural スタックの最上位にあるエントリの内容を表示します。「*ナビゲーションと情報コマンド*」の「STACK」を参照してください。
STEP	[*n*]	コマンドで指定した実行ステートメントの数（*n*）に対して、中断されたNaturalオブジェクトの処理を続行します。 *n*を指定しない場合、1つの実行ステートメントがデフォルトでスキップされます。「*実行制御コマンド*」の「STEP」も参照してください。
	SKIPSUBLEVEL [*n*]	下位レベルのプログラムに入らずに、Natural オブジェクトのステップモード処理を続行します。 レベル番号（*n*）を指定できます。「*実行制御コマンド*」の「SKIPSUBLEVEL」も参照してください。
STOP		デバッガと中断されたNaturalオブジェクトの両方を終了します。NEXTプロンプトが表示されます。
SYSVARS		システム変数（データベース関連システム変数を除く）の制限セットの現在値を表示します。「［*Display Variables*］」も参照してください。
TEST ON または TEST OFF		テストモードをオンまたはオフに切り替えます。「*テストモードのオンとオフの切り替え*」も参照してください。
WM		「*ウォッチポイントのメンテナンス*」で説明した［**Watchpoint Maintenance**］メニューを表示します。

構文図

次に示す構文図は、複雑なコマンドシーケンスを示しています。

構文の説明で使用されている記号の詳細については、『システムコマンド』ドキュメントの「*システムコマンド構文*」を参照してください。

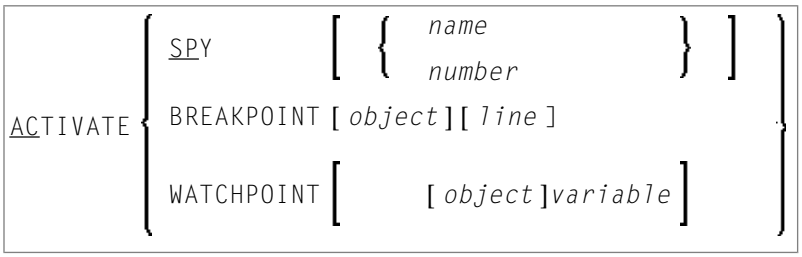
わかりやすくするために、次の構文図では同義のキーワードを省略しています。 キーワードの下線部分は、入力可能な省略形を表します。

有効なシノニム：

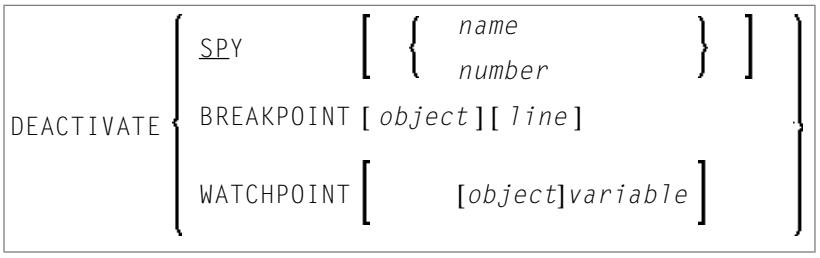
キーワード	シノニム
BREAKPOINT	BP
DEACTIVATE	DA
WATCHPOINT	WP

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

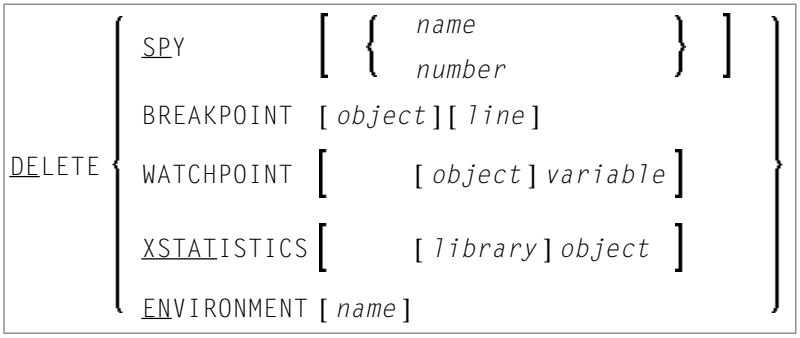
ACTIVATE



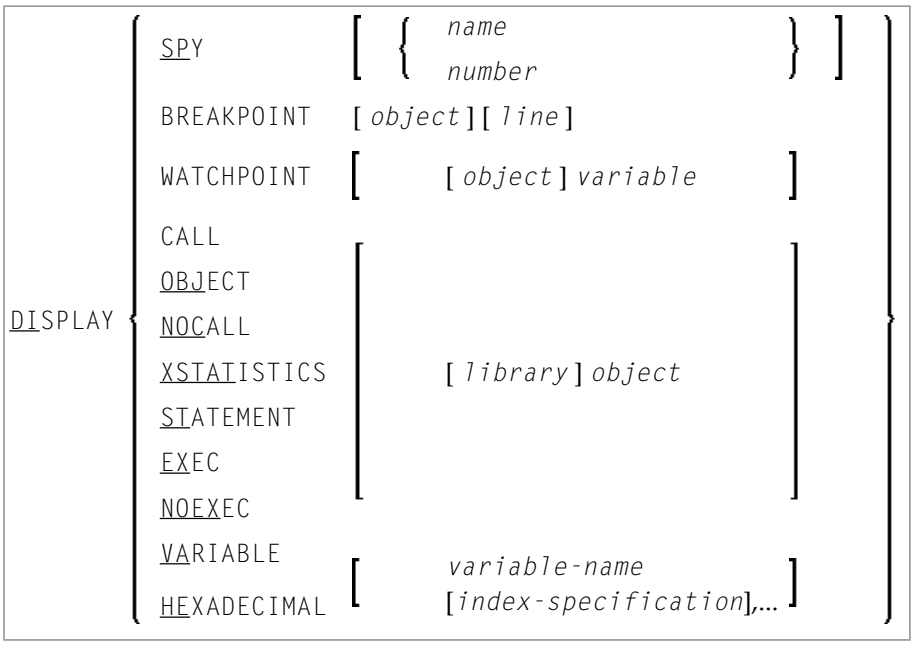
DEACTIVATE



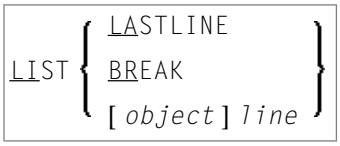
DELETE



DISPLAY



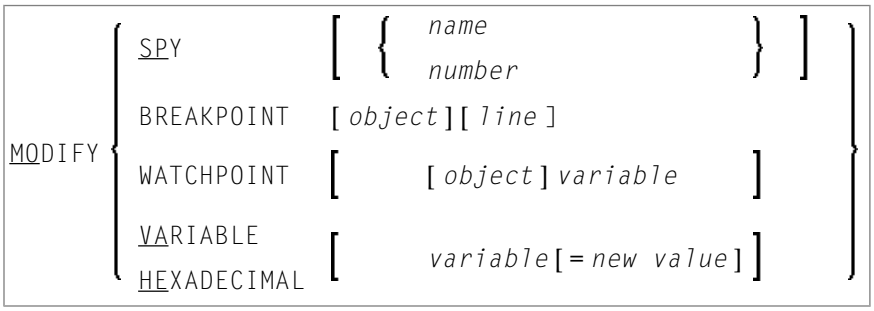
LIST



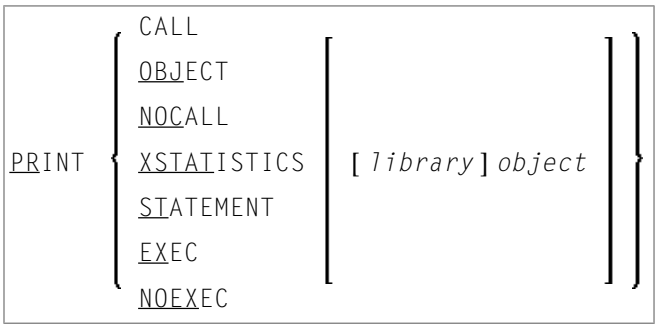
LOAD

```
LOAD ENVIRONMENT[ name]
```

MODIFY



PRINT



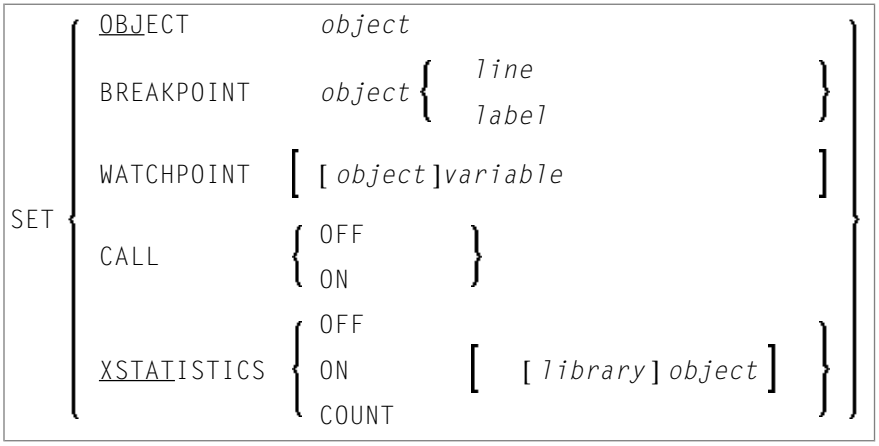
RESET

```
RESET ENVIRONMENT [ name ]
```

SAVE

```
SAVE ENVIRONMENT [ name ]
```

SET


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