

# HP OpenVMS

---

## デバッガ・コマンド・ディクショナリ

AA-QUTWD-TE

2005 年 4 月

本書は、高級言語とアセンブリ言語のプログラマのための OpenVMS デバッガ機能のコマンドについて説明します。

改訂 / 更新情報:

OpenVMS V7.3 『デバッガ・コマンド・ディクショナリ』の改訂版です。

ソフトウェア・バージョン:

OpenVMS I64 Version 8.2

OpenVMS Alpha Version 8.2

OpenVMS VAX Version 7.3

日本ヒューレット・パッカード株式会社

---

© Copyright 2005 Hewlett-Packard Development Company, L.P.

本書の著作権は日本ヒューレット・パカード株式会社が保有しており、本書中の解説および図、表はヒューレット・パカードの文書による許可なしに、その全体または一部を、いかなる場合にも再版あるいは複製することを禁じます。

また、本書に記載されている事項は、予告なく変更されることがありますので、あらかじめご承知おきください。万一、本書の記述に誤りがあった場合でも、ヒューレット・パカードは一切その責任を負いかねます。

本書で解説するソフトウェア (対象ソフトウェア) は、所定のライセンス契約が締結された場合に限り、その使用あるいは複製が許可されます。

ヒューレット・パカードは、ヒューレット・パカードまたはヒューレット・パカードの指定する会社から納入された機器以外の機器で対象ソフトウェアを使用した場合、その性能あるいは信頼性について一切責任を負いかねます。

Intel および Itanium は、米国およびその他の国における、Intel Corporation またはその関連会社の商標または登録商標です。

原典：HP OpenVMS Debugger Manual

© Copyright 2005 Hewlett-Packard Development Company, L.P.

本書は、日本語 VAX DOCUMENT V 2.1を用いて作成しています。

---

# 目次

まえがき .....	vii
1 デバッガ・コマンド・ディクショナリ概要	
1.1 はじめに .....	1-1
1.2 デバッガ・コマンド形式 .....	1-1
1.2.1 一般形式 .....	1-1
1.2.2 会話形式入力の規則 .....	1-2
1.2.3 コマンド・プロシージャでのコマンド入力 .....	1-3
1.3 デバッガの HP DECwindows Motif for OpenVMS ユーザ・インタフェースでは使 用不可能なコマンド .....	1-4
1.4 デバッガ診断メッセージ .....	1-5
1.5 デバッガ・コマンド・ディクショナリ .....	1-6
2 デバッガ・コマンド・ディクショナリ	
@ (Execute Procedure) .....	2-2
ACTIVATE BREAK .....	2-4
ACTIVATE TRACE .....	2-7
ACTIVATE WATCH .....	2-10
ANALYZE/CRASH_DUMP .....	2-12
ANALYZE/PROCESS_DUMP .....	2-14
ATTACH .....	2-16
CALL .....	2-18
CANCEL ALL .....	2-24
CANCEL BREAK .....	2-26
CANCEL DISPLAY .....	2-30
CANCEL MODE .....	2-32
CANCEL RADIX .....	2-34
CANCEL SCOPE .....	2-36
CANCEL SOURCE .....	2-37
CANCEL TRACE .....	2-41
CANCEL TYPE/OVERRIDE .....	2-44
CANCEL WATCH .....	2-45
CANCEL WINDOW .....	2-47
CONNECT .....	2-49
Ctrl/C .....	2-53
Ctrl/W .....	2-55
Ctrl/Y .....	2-56
Ctrl/Z .....	2-58

DEACTIVATE BREAK .....	2-59
DEACTIVATE TRACE .....	2-62
DEACTIVATE WATCH .....	2-65
DECLARE .....	2-67
DEFINE .....	2-70
DEFINE/KEY .....	2-73
DEFINE/PROCESS_SET .....	2-77
DELETE .....	2-80
DELETE/KEY .....	2-82
DEPOSIT .....	2-85
DISABLE AST .....	2-93
DISCONNECT .....	2-94
DISPLAY .....	2-96
DUMP .....	2-103
EDIT .....	2-106
ENABLE AST .....	2-108
EVALUATE .....	2-109
EVALUATE/ADDRESS .....	2-112
EXAMINE .....	2-115
EXIT .....	2-128
EXITLOOP .....	2-132
EXPAND .....	2-133
EXTRACT .....	2-136
FOR .....	2-138
GO .....	2-140
HELP .....	2-143
IF .....	2-145
MONITOR .....	2-146
MOVE .....	2-150
PTHREAD .....	2-153
QUIT .....	2-155
REBOOT (Alpha および I64 のみ) .....	2-158
REPEAT .....	2-159
RERUN .....	2-160
RUN .....	2-162
SAVE .....	2-165
SCROLL .....	2-167
SEARCH .....	2-170
SDA .....	2-174
SELECT .....	2-176
SET ABORT_KEY .....	2-181
SET ATSIGN .....	2-183
SET BREAK .....	2-184
SET DEFINE .....	2-196
SET EDITOR .....	2-198
SET EVENT_FACILITY .....	2-201
SET IMAGE .....	2-203

SET KEY .....	2-205
SET LANGUAGE .....	2-207
SET LANGUAGE/DYNAMIC .....	2-210
SET LOG .....	2-211
SET MARGINS .....	2-213
SET MODE .....	2-216
SET MODULE .....	2-220
SET OUTPUT .....	2-223
SET PROCESS .....	2-225
SET PROMPT .....	2-228
SET RADIX .....	2-230
SET SCOPE .....	2-233
SET SEARCH .....	2-238
SET SOURCE .....	2-240
SET STEP .....	2-246
SET TASK   THREAD .....	2-250
SET TERMINAL .....	2-254
SET TRACE .....	2-256
SET TYPE .....	2-266
SET WATCH .....	2-269
SET WINDOW .....	2-277
SHOW ABORT_KEY .....	2-279
SHOW AST .....	2-280
SHOW ATSIGN .....	2-281
SHOW BREAK .....	2-282
SHOW CALLS .....	2-284
SHOW DEFINE .....	2-288
SHOW DISPLAY .....	2-289
SHOW EDITOR .....	2-291
SHOW EVENT_FACILITY .....	2-292
SHOW EXIT_HANDLERS .....	2-293
SHOW IMAGE .....	2-294
SHOW KEY .....	2-296
SHOW LANGUAGE .....	2-299
SHOW LOG .....	2-300
SHOW MARGINS .....	2-301
SHOW MODE .....	2-303
SHOW MODULE .....	2-304
SHOW OUTPUT .....	2-308
SHOW PROCESS .....	2-309
SHOW RADIX .....	2-314
SHOW SCOPE .....	2-316
SHOW SEARCH .....	2-319
SHOW SELECT .....	2-321
SHOW SOURCE .....	2-323
SHOW STACK .....	2-325
SHOW STEP .....	2-329

SHOW SYMBOL .....	2-330
SHOW TASK   THREAD .....	2-334
SHOW TERMINAL .....	2-338
SHOW TRACE .....	2-339
SHOW TYPE .....	2-341
SHOW WATCH .....	2-343
SHOW WINDOW .....	2-344
SPAWN .....	2-346
START HEAP_ANALYZER (I64 のみ) .....	2-349
STEP .....	2-350
STOP .....	2-358
SYMBOLIZE .....	2-360
TYPE .....	2-362
WAIT .....	2-365
WHILE .....	2-366

## 索引

## 表

2-1	プロセス作成の制限事項 (デバッガのバージョン別) .....	2-51
2-2	デバッグ状態 .....	2-311

## 対象読者

本書は、デバッガを使用するすべてのプログラマを対象とします。本書には、デバッガの次の2つのユーザ・インタフェースについての内容が含まれています。

- 端末およびワークステーションで使用するコマンド・インタフェース
- ワークステーションで使用する HP DECwindows Motif for OpenVMS ユーザ・インタフェース
- Microsoft Windows PC クライアント・インタフェース

OpenVMS I64 あるいは OpenVMS Alpha システムの OpenVMS デバッガを使用すると、OpenVMS オペレーティング・システムの 64 ビット処理により使用可能になる、すべての拡張メモリにアクセスできるようになります。このため、完全な 64 ビット・アドレス空間でデータのテストと処理が行えるようになります。

OpenVMS デバッガはあらゆる地域で使えるよう設計されています。アジア地域のユーザであれば、デバッガの HP DECwindows Motif for OpenVMS、コマンド行、画面モード・ユーザ・インタフェースをマルチバイト文字で 사용할 こともできます。

デバッガを使用してコードをデバッグすることができるのは、ユーザ・モードの場合だけです。スーパーバイザ・モード、エグゼクティブ・モード、カーネル・モードではコードをデバッグすることはできません。

## 本書の構成

本書は2つの章で構成されています。

第1章：デバッガ・コマンドの概要が記述されています。

第2章：デバッガ・コマンドが辞書形式で記述されています。

デバッグの際は『OpenVMS デバッガ説明書』をご覧ください。

## 関連資料

デバッガを使用する際には、次の資料も参考になります。

### プログラミング言語

本書では、デバッガでサポートしている言語の大部分に共通する使用法について記述しています。特定の言語に固有の情報についての詳しい説明は、次の資料を参照してください。

- デバッガのオンライン・ヘルプ・システム
- 各言語に提供されている資料のうち、特にプログラムのデバッグのためのコンパイルとリンクに関する資料
- VAX アセンブリ言語命令とVAX MACROアセンブラの詳細について説明している『VAX MACRO and Instruction Set Reference Manual』または『MACRO-64 Assembler for OpenVMS AXP Systems Reference Manual』

### リンカ・ユーティリティ

プログラムや共用可能イメージのリンクについての詳しい説明は、『OpenVMS Linker Utility Manual』を参照してください。

### Delta/XDelta デバッガ

スーパーバイザ・モード、エグゼクティブ・モード、カーネル・モード（つまり、ユーザ・モード以外のモード）でのコードのデバッグについての詳しい説明は、ドキュメント・セットの『OpenVMS Delta/XDelta Debugger Manual』を参照してください。このマニュアルには、特権プロセッサ・モードで実行するプログラムや、高い割り込み優先順位で実行するプログラムのデバッグについての情報が記載されています。

### OpenVMS System-Code デバッガ

オペレーティング・システム・コードのデバッグについては、『OpenVMS System Analysis Tools Manual』を参照してください。このマニュアルには、OpenVMS デバッガから OpenVMS System-Code デバッガを起動する方法、OpenVMS System-Code デバッガ環境でのデバッグ方法についての情報が記載されています。

OpenVMS System-Code デバッガ固有のコマンドについての詳しい説明は、『デバッガ・コマンド・ディクショナリ』の CONNECT コマンドと REBOOT コマンドの項を参照してください。



HP DECwindows Motif for OpenVMS  
HP DECwindows Motif for OpenVMS ユーザ・インタフェースの一般的な情報については、『VMS DECwindows User's Guide』を参照してください。

その他の HP OpenVMS 製品やサービスについての詳細は、次の Web サイトを参照してください。

<http://www.hp.com/go/openvms>

## 本書で使用する表記法

製品名について

VMScluster システムは、OpenVMS クラスタ・システムを指します。

また、日本語 DECwindows および日本語 DECwindows Motif はすべて日本語 DECwindows Motif for OpenVMS ソフトウェアを意味します。

例について

本書には、デバッガの DECwindows Motif ユーザ・インタフェースを示す図が多数収録されています。このインタフェースの画面構成はそれぞれのユーザごとにカスタマイズできるため、ユーザのシステム上のデバッガ表示と一致しないことがあります。

OpenVMS I64 あるいは OpenVMS Alpha システムの OpenVMS デバッガは、OpenVMS オペレーティング・システムの 64 ビット処理により使用可能になるすべての拡張メモリに対してアクセスできるようになっていますが、本書のサンプルは、その事実を反映するよう更新されてはいません。そのため、16 進アドレスは、Alpha では 16 桁の数で VAX では 8 桁の数になります。つまり次の例のようになります。

```
DBG> EVALUATE/ADDRESS/HEX %hex 000004A0
000000000000004A0
DBG>
```

また、本書では、次の表記法を使用しています。

表記法	意味
Ctrl/x	Ctrl/x という表記は、Ctrl キーを押しながら別のキーまたはポインティング・デバイス・ボタンを押すことを示します。
PF1 x	PF1 x という表記は、PF1 に定義されたキーを押してから、別のキーまたはポインティング・デバイス・ボタンを押すことを示します。

表記法	意味
<code>Return</code>	例の中で、キー名が四角で囲まれている場合には、キーボード上でそのキーを押すことを示します。テキストの中では、キー名は四角で囲まれていません。 HTML 形式のドキュメントでは、キー名は四角ではなく、括弧で囲まれています。
...	例の中の水平方向の反復記号は、次のいずれかを示します。 <ul style="list-style-type: none"> <li>• 文中のオプションの引数が省略されている。</li> <li>• 前出の 1 つまたは複数の項目を繰り返すことができる。</li> <li>• パラメータや値などの情報をさらに入力できる。</li> </ul>
. . .	垂直方向の反復記号は、コードの例やコマンド形式の中の項目が省略されていることを示します。このように項目が省略されるのは、その項目が説明している内容にとって重要ではないからです。
( )	コマンドの形式の説明において、括弧は、複数のオプションを選択した場合に、選択したオプションを括弧で囲まなければならないことを示しています。
[ ]	コマンドの形式の説明において、大括弧で囲まれた要素は任意のオプションです。オプションをすべて選択しても、いずれか 1 つを選択しても、あるいは 1 つも選択しなくても構いません。ただし、OpenVMS ファイル指定のディレクトリ名の構文や、割り当て文の部分文字列指定の構文の中では、大括弧に囲まれた要素は省略できません。
[   ]	コマンド形式の説明では、括弧内の要素を分けている垂直棒線はオプションを 1 つまたは複数選択するか、または何も選択しないことを意味します。
{ }	コマンドの形式の説明において、中括弧で囲まれた要素は必須オプションです。いずれか 1 つのオプションを指定しなければなりません。
太字	太字のテキストは、新しい用語、引数、属性、条件を示しています。
<i>italic text</i>	イタリック体のテキストは、重要な情報を示します。また、システム・メッセージ (たとえば内部エラー <i>number</i> )、コマンド行 (たとえば <i>/PRODUCER=name</i> )、コマンド・パラメータ (たとえば <i>device-name</i> ) などの変数を示す場合にも使用されます。
UPPERCASE TEXT	英大文字のテキストは、コマンド、ルーチン名、ファイル名、ファイル保護コード名、システム特権の短縮形を示します。
Monospace type	モノスペース・タイプの文字は、コード例および会話型の画面表示を示します。 C プログラミング言語では、テキスト中のモノスペース・タイプの文字は、キーワード、別々にコンパイルされた外部関数およびファイルの名前、構文の要約、または例に示される変数または識別子への参照などを示します。
-	コマンド形式の記述の最後、コマンド行、コード・ラインにおいて、ハイフンは、要求に対する引数がその後の行に続くことを示します。
数字	特に明記しない限り、本文中の数字はすべて 10 進数です。10 進数以外 (2 進数、8 進数、16 進数) は、その旨を明記してあります。

---

# デバッガ・コマンド・ディクショナリ概要

---

## 1.1 はじめに

デバッガ・コマンド・ディクショナリには、すべてのデバッガ・コマンドに関する詳細情報があります。その構成は次のとおりです。

- 「第 1.2 節」では、デバッガ・コマンドを入力する方法を説明します。
- 「第 1.3 節」では、デバッガの HP DECwindows Motif for OpenVMS ユーザ・インタフェースのコマンド/メッセージ入力ビューで使用不可能なコマンドの一覧を示します。
- 「第 1.4 節」には、デバッガ診断メッセージに関する一般的な情報を示します。
- 「第 1.5 節」には、デバッガ・コマンドに関する詳しい参考情報があります。

---

## 1.2 デバッガ・コマンド形式

デバッガ・コマンドはキーボードで会話形式で入力するか、または実行プロシージャ (@) コマンドにより、あとで実行されるコマンド・プロシージャの中に保存することができます。

この節では次の情報について記述します。

- デバッガ・コマンドの一般形式
- キーボードで会話形式でコマンドを入力するための規則
- デバッガ・コマンド・プロシージャにコマンドを入力するための規則

### 1.2.1 一般形式

コマンド文字列はデバッガ・コマンドの完全な指定です。1 つのコマンドを 2 行以上に続けることもできますが、コマンド文字列という用語はデバッガに渡されるコマンド全体を定義する際に使用されます。

デバッガ・コマンド文字列は動詞で構成され、パラメータと修飾子が付くこともあります。

動詞は実行の対象となるコマンドを指定します。デバッガ・コマンド文字列の中には、次の例のように 1 つの動詞だけか、1 対の動詞だけで構成されるものもあります。

```
DBG> GO
DBG> SHOW IMAGE
```

パラメータは指定された動詞が作用する対象 (たとえばファイル指定) を指定します。修飾子は動詞による処理を記述または変更するものです。コマンド文字列によっては、パラメータまたは修飾子を 1 つまたは複数入れられるものがあります。次の例では、COUNT、I、J、K、OUT2、PROG4.COM はパラメータで (@は実行プロシージャ・コマンド)、/SCROLL と/OUTPUT が修飾子です。

```
DBG> SET WATCH COUNT
DBG> EXAMINE I,J,K
DBG> SELECT/SCROLL/OUTPUT OUT2
DBG> @PROG4.COM
```

コマンドの中には、省略可能な WHEN 句または DO 句を受け付けるものもあります。DO 句はある種の画面表示定義の中でも使用されます。

WHEN 句の構成はキーワードの WHEN を記述し、続いて現在の言語で評価結果が真か偽かになる条件式を (括弧で囲んで) 指定します。DO 句の構成はキーワードの DO を記述し、続いて 1 つまたは複数の (括弧で囲まれた) コマンド文字列を指定します。これらのコマンドは指定した順に実行されます。複数のコマンド文字列があるときは、それぞれをセミコロン(;)で区切らなければなりません。これらの点について次の例で説明します。

次のコマンド文字列は SWAP というルーチンにブレークポイントを設定します。実行時に J の値が 4 になるといつでもブレークポイントが検出されます。ブレークポイントが検出されると、デバッガは指示された順番で 2 つのコマンド、SHOW CALLS と EXAMINE I,K を実行します。

```
DBG> SET BREAK SWAP WHEN (J = 4) DO (SHOW CALLS; EXAMINE I,K)
```

デバッガは DO 句の実行時に DO 句の中のコマンドの構文を調べます。DO 句の中でコマンドをネストできます。

### 1.2.2 会話形式入力の規則

キーボードで会話形式でコマンドを入力するとき、すべてのデバッガ・キーワード群にわたって固有であるために最低必要な文字数までキーワード (動詞、修飾子、パラメータ) を短縮できます。ただし、よく使用されるコマンドのいくつか (たとえば EXAMINE、DEPOSIT、GO、STEP) は最初の文字だけに短縮できます。また、固有でない短縮形でもデバッガが状況に基づいて正しく解釈できる場合もあります。

Return キーを押すと、現在の行が終了し、デバッガはその行を処理します。長いコマンド文字列を次の行に続けるには、Return キーを押す前にハイフン(-)を入力します。こうすると、デバッガ・プロンプトにはアンダスコア文字 (\_DBG>) が接頭辞として付き、まだコマンド文字列の入力中であることを示します。

各コマンド文字列をセミコロン(;)で区切るにより、1行に複数のコマンド文字列を入力できます。

コメント(デバッガ・ログ・ファイルには記録されているが、その他の点ではデバッガから無視される説明文)を入力するには、入力するコメント・テキストの前に感嘆符(!)を入れます。コメントが次の行に続く場合には、新しい行を感嘆符で始めてください。

DCL プロンプト (\$) が表示された状態で使用できるコマンド行編集機能は、上向き矢印キーと下向き矢印キーによるコマンドの再呼び出しも含めて、デバッガ・プロンプト (DBG>) が表示された状態でも使用できます。たとえば、左向き矢印キーを押すとカーソルが左に1文字移動し、右向き矢印キーを押すと右に1文字移動します。Ctrl/H を押すとカーソルは行頭に移動し、Ctrl/E を押すと行末に移動します。Ctrl/U を押すと、カーソルの左側にある文字がすべて削除されます。

デバッガで処理中のコマンドに割り込みをかけるには、Ctrl/C を押します。Ctrl/C コマンドを参照してください。

### 1.2.3 コマンド・プロシージャでのコマンド入力

コマンドを読みやすくするために、コマンド・プロシージャの中ではコマンド・キーワードを短縮しないでください。将来のリリースでの潜在的な矛盾を避けるために、コマンド・キーワードを(否定の/NO...を数えずに)有意文字3文字以下に短縮しないでください。

デバッガ・コマンド行は左マージンから始めます。(DCL コマンド・プロシージャを記述するときのように、ドル記号(\$)でコマンド行を始めないでください。)

新しい行を開始すると以前のコマンド行が終結します(ファイルの終端文字によっても以前のコマンド行は終結します)。コマンド文字列を次の行に続けるには、新しい行を開始する前にハイフン(-)を入力してください。

セミコロン(;)で各コマンド文字列を区切るにより、1行に複数のコマンド文字列を入力できます。

コメント(コマンド・プロシージャの実行には影響しない説明文)を入力するには、入力するコメント・テキストの前に感嘆符(!)を入れます。コメントが次の行に続く場合には、新しい行を感嘆符で始めてください。

---

## 1.3 デバッガの HP DECwindows Motif for OpenVMS ユーザ・インタフェースでは使用不可能なコマンド

次のコマンドは、デバッガへの HP DECwindows Motif for OpenVMS ユーザ・インタフェースでは使用できません。このうちの多くはコマンド・インタフェースの画面モードにだけ関係があります。

ATTACH	SELECT
CANCEL MODE	(SET,SHOW) ABORT_KEY
CANCEL WINDOW	(SET,SHOW) KEY
DEFINE/KEY	(SET,SHOW) MARGINS
DELETE/KEY	SET MODE [NO]KEYPAD
DISPLAY	SET MODE [NO]SCREEN
EXAMINE/SOURCE	SET MODE [NO]SCROLL
EXPAND	SET OUTPUT [NO]TERMINAL
EXTRACT	(SET,SHOW) TERMINAL
HELP <sup>1</sup>	(SET,SHOW) WINDOW
MOVE	(SHOW,CANCEL) DISPLAY
SAVE	SHOW SELECT
SCROLL	SPAWN

---

<sup>1</sup> コマンドに関するヘルプはデバッガ・ウィンドウの Help メニューから入手できます。

コマンド・プロンプトに対して上記の使用できないコマンドを入力しようとした場合、またはデバッガが上記のコマンドのうちのどれか 1 つでも含むコマンド・プロシージャを実行すると、デバッガはエラー・メッセージを発行します。

MONITOR コマンドは HP DECwindows Motif for OpenVMS ユーザ・インタフェースでだけ作用します (このコマンドはモニタ・ビューを使用するためです)。

## 1.4 デバッガ診断メッセージ

次の例でデバッグ診断メッセージの要素を示します。

```
%DEBUG-W-NOSYMBOL, symbol 'X' is not in the symbol table
```

**1      2      3                                  4**

- 1 ファシリティ名 (DEBUG)。
- 2 重大度 (この例では W)。
- 3 メッセージ識別子 (この例では NOSYMBOL)。メッセージ識別子はメッセージ文の短縮形になっている。
- 4 メッセージ文。

識別子を使用すれば、デバッガのオンライン・ヘルプから診断メッセージの説明を(必要ならそのメッセージに対するアクションも)見つけられます。

デバッガ・メッセージについてのオンライン・ヘルプを獲得するには、次のコマンド形式を使用します。

HELP MESSAGES *message-identifier*

診断メッセージの重大度として考えられるものは次のとおりです。

S (正常終了)  
I (情報)  
W (警告)  
E (エラー)  
F (回復不可能, または重大なエラー)

正常終了メッセージと情報メッセージは、デバッガが要求を実行したことをユーザに通知します。

警告メッセージは、要求の一部しか実行しなかったため結果を調べる必要があるということを通知します。

エラー・メッセージは、デバグが要求を実行できなかったが、デバグ・セッションの状態は変更されなかったことを示します。ただし、メッセージ識別子が DBGERR または INTERR だった場合は例外です。これらの識別子は内部デバグ・エラーを意味するため、その場合には弊社のサポート要員にご連絡ください。

回復不可能なエラー・メッセージは、デバッガが要求を実行できなかったことと、デバッグ・セッションが不確定な状態になって、その状態からユーザが抜け出せないかもしれないということを示します。通常、このエラーが発生するとデバッグ・セッションは終了します。

---

## 1.5 デバッガ・コマンド・ディクショナリ

デバッガ・コマンド・ディクショナリはデバッガ・コマンドの1つ1つを詳しく説明します。コマンドはアルファベット順に並んでいます。各コマンドごとに説明，形式，パラメータ，修飾子，例を示します。ドキュメントの表記法については，本書のまえがきを参照してください。



---

## デバッグ・コマンド・ディクショナリ

この章では、アルファベット順にデバッグ・コマンドを説明しています。

---

## @ (Execute Procedure)

デバッガ・コマンド・プロシージャを実行します。

---

### フォーマット

*@file-spec [parameter[, ... ]]*

---

### パラメータ

#### file-spec

実行の対象となるコマンド・プロシージャを指定します。完全ファイル指定の中で指定されていない部分については、デバッガは、最新の SET ATSIGN コマンドで設定されたファイル指定があればそれを使用します。ファイル指定の欠けている部分が SET ATSIGN コマンドで設定されていなかった場合には、デバッガは省略時のファイル指定として SYS\$DISK:[ ]DEBUG.COM を割り当てます。file-spec には論理名を指定できます。

#### parameter

コマンド・プロシージャに渡すパラメータを指定します。パラメータにはアドレス式、現在の言語での値式、デバッガ・コマンドを指定できます。コマンドは二重引用符(")で囲む必要があります。DCL の場合とは異なり、各パラメータはコンマで区切らなくてはなりません。また、コマンド・プロシージャの中にある仮パラメータ宣言と同じ数のパラメータを渡せます。コマンド・プロシージャへのパラメータの引き渡しについての詳しい説明は、DECLARE コマンドを参照してください。

---

### 説明

デバッガ・コマンド・プロシージャには、他の実行プロシージャ (@) コマンドを含めたどのデバッガ・コマンドも指定できます。デバッガは EXIT コマンドまたは QUIT コマンドに到達するか、コマンド・プロシージャが終わるまで、コマンド・プロシージャのコマンドを実行します。コマンド・プロシージャの実行が終了すると、デバッガはコマンド・プロシージャを起動したコマンド・ストリームに制御を戻します。この場合のコマンド・ストリームとは端末、外側 (当該コマンド・プロシージャを含んでいる) のコマンド・プロシージャ、SET BREAK などのコマンド内の DO 句、画面表示定義の中の DO 句などです。

省略時の設定では、コマンド・プロシージャから読み込まれたコマンドはエコーバックされません。SET OUTPUT VERIFY コマンドを入力すると、コマンド・プロシ

ージャから読み込まれたコマンドはすべて、DBG\$OUTPUT による指定に従い、現在の出力装置にエコーバックされます (省略時の出力装置は SYSS\$OUTPUT)。

コマンド・プロシージャへのパラメータの引き渡しについての詳しい説明は、DECLARE コマンドを参照してください。

#### 関連コマンド

```
DECLARE
  (SET,SHOW) ATSIGN
  SET OUTPUT [NO]VERIFY
  SHOW OUTPUT
```

---

#### 例

```
DBG> SET ATSIGN USER:[JONES.DEBUG].DBG
DBG> SET OUTPUT VERIFY
DBG> @CHECKOUT
%DEBUG-I-VERIFYICF, entering command procedure CHECKOUT
  SET MODULE/ALL
  SET BREAK SUB1
  GO
break at routine PROG5\SUB2
  EXAMINE X
PROG5\SUB2\X: 376
  . . .
%DEBUG-I-VERIFYICF, exiting command procedure MAIN
DBG>
```

この例の、SET ATSIGN コマンドは、デバッガ・コマンド・プロシージャが省略時の設定では USER:[JONES.DEBUG]の中にあり、ファイル・タイプとして.DBGを持つように設定しています。@CHECKOUT コマンドはコマンド・プロシージャ USER:[JONES.DEBUG]CHECKOUT.DBG を実行します。SET OUTPUT VERIFY コマンドが指定されているため、デバッガは@CHECKOUT コマンドの中のコマンドをエコーバックします。

---

# ACTIVATE BREAK

以前に無効に設定したブレークポイントを有効にします。

---

## フォーマット

ACTIVATE BREAK *[address-expression[, . . . ]]*

---

## パラメータ

address-expression

有効にするブレークポイントを指定します。ワイルドカード文字のアスタリスク(\*)は使用できません。代わりに/ALL 修飾子を使用してください。/EVENT , /PREDEFINED , /USER 以外の修飾子を使用する場合は、アドレス式は指定できません。

---

## 修飾子

/ACTIVATING

前回の SET BREAK/ACTIVATING コマンドで設定されたブレークポイントを有効にします。

/ALL

省略時の設定では、すべてのユーザ定義ブレークポイントを有効にします。  
/PREDEFINED といっしょに使用すると、定義済みブレークポイントはすべて有効になりますが、ユーザ定義ブレークポイントは有効になりません。すべてのブレークポイントを有効にするには/ALL/USER/PREDEFINED を使用します。

/BRANCH

前回の SET BREAK/BRANCH コマンドで設定されたブレークポイントを有効にします。

/CALL

前回の SET BREAK/CALL コマンドで設定されたブレークポイントを有効にします。

/EVENT=event-name

前回の SET BREAK/EVENT=*event-name* コマンドで設定されたブレークポイントを有効にします。イベント名 (および必要であればアドレス式) は SET BREAK/EVENT コマンドで指定したとおりに指定してください。

現在のイベント機能とそれに対応するイベント名を表示するには、SHOW EVENT\_FACILITY コマンドを使用します。

#### /EXCEPTION

前回の SET BREAK/EXCEPTION コマンドで設定されたブレークポイントを有効にします。

#### /HANDLER

前回の SET BREAK/HANDLER コマンドの作用を取り消します。

#### /INSTRUCTION

前回の SET BREAK/INSTRUCTION コマンドで設定されたブレークポイントを有効にします。

#### /LINE

前回の SET BREAK/LINE コマンドで設定されたブレークポイントを有効にします。この修飾子を指定する場合は、アドレス式は指定できません。

#### /PREDEFINED

ユーザ定義ブレークポイントには全く影響を及ぼさずに、指定の定義済みブレークポイントを有効にします。/ALL といっしょに使用すると、定義済みブレークポイントがすべて有効になります。

#### /SYSEMULATE

(Alpha のみ) 前回の SET BREAK/SYSEMULATE コマンドによって設定されたブレークポイントを有効にします。

#### /TERMINATING

前回の SET BREAK/TERMINATING コマンドで設定されたブレークポイントを有効にします。

#### /UNALIGNED\_DATA

(Alpha および I64 のみ) 前回の SET BREAK/UNALIGNED\_DATA コマンドで設定されたブレークポイントを有効にするか、または DEACTIVATE /UNALIGNED\_DATA コマンドにより無効になっていたブレークポイントを再度有効にします。

#### /USER

定義済みブレークポイントには全く影響を及ぼさずに指定のユーザ定義ブレークポイントを有効にします。/ALL といっしょに使用すると、すべてのユーザ定義ブレークポイントが有効になります。

---

## 説明

ユーザ定義ブレークポイントは SET BREAK コマンドで設定すると有効になります。定義済みブレークポイントは省略時の設定で有効になります。DEACTIVATE BREAK を使用して無効にした 1 つまたは複数のブレークポイントを有効にするには、ACTIVATE BREAK コマンドを使用します。

ブレークポイントを有効にしたり無効にしたりすることにより、プログラムの実行や再実行のときに、ブレークポイントを取り消して再設定する手間をかけずに、ブレークポイントを使用したり使用しなかったりすることができます。省略時の設定で RERUN コマンドを実行すると、すべてのブレークポイントの現在の状態 (有効か無効か) が保存されます。

有効や無効にするのはユーザ定義ブレークポイントと定義済みブレークポイントのどちらか一方でも両方でもかまいません。ブレークポイントが有効になっているかを確認するには、SHOW BREAK コマンドを使用します。

### 関連コマンド

```
CANCEL ALL
RERUN
(SET,SHOW,CANCEL,DEACTIVATE) BREAK
(SET,SHOW) EVENT_FACILITY
```

---

### 例

1. DBG> ACTIVATE BREAK MAIN\LOOP+10  
このコマンドはアドレス式 MAIN\LOOP+10 で設定されたユーザ定義ブレークポイントを有効にします。
2. DBG> ACTIVATE BREAK/ALL  
このコマンドはすべてのユーザ定義ブレークポイントを有効にします。
3. DBG> ACTIVATE BREAK/ALL/USER/PREDEFINED  
このコマンドはユーザ定義ブレークポイントと定義済みブレークポイントの両方を含めたすべてのブレークポイントを有効にします。

---

# ACTIVATE TRACE

以前に設定したあと無効にしたトレースポイントを有効にします。

---

## フォーマット

ACTIVATE TRACE *[address-expression[, . . . ]]*

---

## パラメータ

address-expression

有効にするトレースポイントを指定します。ワイルドカード文字のアスタリスク(\*)は使用できません。代わりに/ALL 修飾子を使用してください。/EVENT , /PREDEFINED , /USER 以外の修飾子を使用する場合は、アドレス式は指定できません。

---

## 修飾子

/ACTIVATING

前回の SET TRACE/ACTIVATING コマンドで設定されたトレースポイントを有効にします。

/ALL

省略時の設定では、すべてのユーザ定義トレースポイントを有効にします。/PREDEFINED といっしょに使用すると、定義済みトレースポイントはすべて有効になりますが、ユーザ定義トレースポイントは有効になりません。すべてのトレースポイントを有効にするには/ALL/USER/PREDEFINED を使用します。

/BRANCH

前回の SET TRACE/BRANCH コマンドで設定されたトレースポイントを有効にします。

/CALL

前回の SET TRACE/CALL コマンドで設定されたトレースポイントを有効にします。

/EVENT=event-name

前回の SET TRACE/EVENT=*event-name* コマンドで設定されたトレースポイントを有効にします。イベント名 (および必要であればアドレス式) は SET TRACE/EVENT コマンドで指定したとおりに指定してください。

## ACTIVATE TRACE

現在のイベント機能とそれに対応するイベント名を表示するには、SHOW EVENT\_FACILITY コマンドを使用します。

/EXCEPTION

前回の SET TRACE/EXCEPTION コマンドで設定されたトレースポイントを有効にします。

/INSTRUCTION

前回の SET TRACE/INSTRUCTION コマンドで設定されたトレースポイントを有効にします。

/LINE

前回の SET TRACE/LINE コマンドで設定されたトレースポイントを有効にします。

/PREDEFINED

ユーザ定義トレースポイントには全く影響を及ぼさずに指定の定義済みトレースポイントを有効にします。/ALL といっしょに使用すると、定義済みトレースポイントがすべて有効になります。

/TERMINATING

前回の SET TRACE/TERMINATING コマンドで設定されたトレースポイントを有効にします。

/USER

定義済みトレースポイントには全く影響を及ぼさずに指定のユーザ定義トレースポイントを有効にします。/ALL 修飾子を使用すると、ユーザ定義トレースポイントがすべて有効になります。

---

## 説明

ユーザ定義トレースポイントは SET TRACE コマンドで設定すると有効になります。定義済みトレースポイントは省略時の設定で有効になります。DEACTIVATE TRACE を使用して無効にした 1 つまたは複数のトレースポイントを有効にするには ACTIVATE TRACE コマンドを使用します。

トレースポイントを有効にしたり無効にしたりすることにより、プログラムの実行や再実行のときに、トレースポイントを取り消して再設定する手間をかけずに、トレースポイントを使用したり使用しなかったりすることができます。省略時の設定で RERUN コマンドを実行すると、すべてのトレースポイントの現在の状態 (有効か無効か) が保存されます。

有効や無効にするのはユーザ定義トレースポイントと定義済みトレースポイントのどちらか一方でも両方でもかまいません。トレースポイントが有効になっているかを確認するには、SHOW TRACE コマンドを使用してください。



## 関連コマンド

CANCEL ALL  
RERUN  
(SET,SHOW) EVENT\_FACILITY  
(SET,SHOW,CANCEL,DEACTIVATE) TRACE

---

例

1. DBG> ACTIVATE TRACE MAIN\LOOP+10

このコマンドは MAIN\LOOP+10 の位置にあるユーザ定義トレースポイントを有効にします。

2. DBG> ACTIVATE TRACE/ALL

このコマンドはユーザ定義トレースポイントをすべて有効にします。

---

# ACTIVATE WATCH

以前に設定したあと無効にしたウォッチポイントを有効にします。

---

## フォーマット

ACTIVATE WATCH *[address-expression], . . . ]]*

---

## パラメータ

address-expression

有効にするウォッチポイントを指定します。高級言語を使用している場合、これは通常、変数の名前になります。ワイルドカード文字のアスタリスク(\*)は使用できません。代わりに/ALL 修飾子を使用してください。/ALL を指定する場合は、アドレス式は指定できません。

---

## 修飾子

/ALL

すべてのウォッチポイントを有効にします。

---

## 説明

ウォッチポイントは SET WATCH コマンドで設定すると有効になります。DEACTIVATE WATCH で無効にした 1 つまたは複数のウォッチポイントを有効にするには ACTIVATE WATCH コマンドを使用します。

ウォッチポイントを有効にしたり無効にしたりすることにより、プログラムの実行や再実行のときに、ウォッチポイントを取り消して再設定する手間をかけずに、ウォッチポイントを使用したり使用しなかったりすることができます。

省略時の設定では、RERUN コマンドを実行すると、すべての静的ウォッチポイントの現在の状態(有効か無効か)が保存されます。特定の非静的ウォッチポイントは、(実行を再開する)メイン・プログラム・ユニットを基準にした、ウォッチの対象になっている変数の有効範囲によって保存されることもあり、保存されないこともあります。

ウォッチポイントが有効になっているかを確かめるには、SHOW WATCH コマンドを使用します。

## 関連コマンド

CANCEL ALL

RERUN

(SET,SHOW,CANCEL,DEACTIVATE) WATCH

---

例

1. DBG> ACTIVATE WATCH SUB2\TOTAL

このコマンドはモジュール SUB2 の中の TOTAL 変数でのウォッチポイントを有効にします。

2. DBG> ACTIVATE WATCH/ALL

このコマンドは設定したあと無効にしたすべてのウォッチポイントを有効にします。

---

## ANALYZE/CRASH\_DUMP

システム・ダンプ・デバッガで解析するためにシステム・ダンプをオープンします (保持デバッガのみ)。

---

### フォーマット

ANALYZE/CRASH\_DUMP *dumpfile*

---

### パラメータ

*dumpfile*

解析するシステム・ダンプ・ファイルの名前。ファイル・タイプは.DMP でなければなりません。

---

### 説明

OpenVMS I64 システムと Alpha システムでは、システム・ダンプを解析するためにシステム・ダンプ・デバッガ (SDD) を起動します。

SDD は概念上はシステム・コード・デバッガ (SCD) に似ています。SCD では実行中のシステムに接続して、システムの実行を制御し、変数の確認と変更を行うことができますが、SDD ではシステム・ダンプに記録されているメモリを解析することができます。

SDD では一般に 2 つのシステムを使用しますが、必要な環境すべてを 1 つのシステム上に設定することも可能です。以下の説明では、2 つのシステムを使用するものと仮定しています。

- ビルド・システム。システム・クラッシュを引き起こしたイメージがビルドされたシステム
- テスト・システム。イメージが実行され、システム・クラッシュが起こったシステム

SCD と同様に、OpenVMS デバッガのユーザ・インタフェースでは、ソース・コードに使われているのと全く同じ変数名やルーチン名などを指定することができます。また、SDD は、システム・クラッシュの時点で実行されていたソース・コードの位置を表示することができます。

SDD は個々の言語の構文，データ型，演算子，式，有効範囲規則，およびその他の言語要素を認識します。コードまたはドライバが複数の言語で書かれている場合には，デバッグ・セッションの途中でデバッグ・コンテキストの言語を切り替えることが可能です。

SDD を使用するためには，以下の操作を行う必要があります。

- システム・クラッシュを引き起こしているシステム・イメージまたはデバイス・ドライバをビルドする。
- そのシステム・イメージまたはデバイス・ドライバを含むシステムをブートし，システム・クラッシュを引き起こすための手順を実行する。
- システムをリブートし，ダンプ・ファイルを保存する。
- OpenVMS デバッガに統合されている SDD を起動する。

サンプルの SDD セッションを含む SDD の詳しい使い方については，『OpenVMS System Analysis Tools Manual』を参照してください。

関連コマンド

```
ANALYZE/PROCESS_DUMP
CONNECT %NODE
SDA
```

---

## 例

1. DBG> ANALYZE/CRASH\_DUMP

DBG>

保持デバッガの中から SDD を起動します。

---

## ANALYZE/PROCESS\_DUMP

システム・コード・デバッガで解析するためにプロセス・ダンプをオープンします  
(保持デバッガのみ)。

---

### フォーマット

ANALYZE/PROCESS\_DUMP *dumpfile*

---

### パラメータ

*dumpfile*

解析するプロセス・ダンプ・ファイルの名前。ファイル・タイプは.DMP でなければなりません。

---

### 修飾子

/IMAGE\_PATH=directory-spec

デバッガがデバッガ・シンボル・テーブル (DST) を含んでいるファイルを探すための検索パスを指定します。ファイルは、ダンプファイルと同じイメージ名を持ち、ファイル・タイプは.DSF または.EXE でなければなりません。たとえば、ダンプ・ファイルにイメージ名 *foo.exe* が含まれている場合、デバッガは *foo.dsf* または *foo.exe* を探します。

---

### 説明

(保持デバッガのみ。) システム・コード・デバッガで解析するためにプロセス・ダンプをオープンします。修飾子/PROCESS\_DUMP は必須で、このコマンドをシステム・ダンプ・デバッガ (SDD) を起動するコマンドANALYZE/CRASH\_DUMPと区別する役割を果たしています。

修飾子/IMAGE\_PATH=directory-specはオプションで、デバッガがデバッガ・シンボル・テーブル (DST) ファイルを探すために使用する検索パスを指定します。デバッガは保存済みプロセス・イメージ・リストからイメージ・リストを構築します。イメージを設定すると (メイン・イメージは自動的に設定されます)、デバッガはそのイメージをオープンして DST を探します。

/IMAGE\_PATH=directory-spec修飾子を含めると、デバッガは指定されたディレクトリの中で.DST ファイルを探します。デバッガはまずdirectory-specをディレクトリ検索リストの論理名として解釈しようとし、これに失敗すると、デバッガはdirectory-specをディレクトリ指定として解釈し、そのディレクトリの中で、対応する.DSF または.EXE ファイルを探します。.DSF ファイルは.EXE ファイルよりも優先されます。.DSF または.EXE ファイルの名前は、イメージ名と一致していなければなりません。

/IMAGE\_PATH=directory-spec修飾子を含めなかった場合、デバッガは最初にダンプ・ファイルを含んでいるディレクトリの中で DST ファイルを探します。これに失敗すると、デバッガはまずディレクトリ SYSS\$SHARE を、次にディレクトリ SYSS\$MESSAGE を探します。デバッガがイメージの DST ファイルを発見できなかった場合、デバッガが使用できるシンボリック情報はグローバルおよびユニバーサル・シンボル名に限定されます。

デバッガは、ダンプ・ファイル・イメージと DST ファイルの間でリンクの日付と時刻が一致しているかどうかをチェックし、一致していなければ警告を発します。

パラメータdumpfileは、解析するプロセス・ダンプ・ファイルの名前です。プロセス・ダンプ・ファイルのタイプは.DMP でなければならず、DST ファイルのタイプは.DSF または.EXE でなければなりません。

SCD の詳しい使い方については、『OpenVMS System Analysis Tools Manual』を参照してください。

#### 関連コマンド

```
ANALYZE/CRASH_DUMP
CONNECT %NODE
SDA
```

---

#### 例

1. DBG> ANALYZE/PROCESS/IMAGE\_DUMP=my\_disk\$:[my\_dir] my\_disk\$:[my\_dir]wecrash.dmp  
%SYSTEM-F-IMGDMP, dynamic image dump signal at PC=001C0FA0B280099C, PS=001C003C  
break on unhandled exception preceding WECRASH\  
th\_run  
\\%LINE 26412 in THREAD 8  
26412: if (verify) {  
DBG> SET RADIX HEXADECIMAL; EXAMINE PC  
WECRASH\th\_run\%PC: 0000000000030244  
DBG>

---

# ATTACH

端末の制御を現在のプロセスから他のプロセスに渡します。

---

## 注意

このコマンドは、デバッガへの HP DECwindows Motif for OpenVMS ユーザ・インタフェースでは使用できません。

---

---

## フォーマット

ATTACH *process-name*

---

## パラメータ

*process-name*

端末の接続先になるプロセスを指定します。指定するプロセスは、それに接続しようとする前に存在しているものでなければなりません。プロセス名に英数字以外の文字またはスペース文字が含まれている場合、その文字を二重引用符(")で囲む必要があります。

---

## 説明

ATTACH コマンドを使用すれば、デバッグ・セッションとコマンド・インタプリタの間または2つのデバッグ・セッションの間を行ったり来たりすることができます。このためには、まず SPAWN コマンドを使用してサブプロセスを作成しなければなりません。そのあとはいつでも望むときにそのサブプロセスに接続できます。システムのオーバーヘッドを最小にして元のプロセスに戻るには、もう一度 ATTACH コマンドを使用します。

関連コマンド

SPAWN



---

## 例

1. 

```
DBG> SPAWN
$ ATTACH JONES
%DEBUG-I-RETURNED, control returned to process JONES
DBG> ATTACH JONES_1
$
```

この例では、一連のコマンドによりデバッガ (現在 JONES というプロセスで実行中) から JONES\_1 という名前のサブプロセスが作成され、作成されたサブプロセスに接続します。

2. 

```
DBG> ATTACH "Alpha One"
$
```

この例は、スペース文字を含むプロセス名を二重引用符で囲む方法を示しています。

---

# CALL

プログラムにリンクされたルーチンを呼び出します。

---

## フォーマット

CALL *routine-name* [(*argument* [, ... ])]

---

## パラメータ

*routine-name*

呼び出し先のルーチンの名前またはメモリ・アドレスを指定します。

*argument*

呼び出し先ルーチンが必要とする引数を指定します。引数は次に示すように、アドレス、ディスクリプタ、参照、値で渡すことができます。

**%ADDR** (C および C++ を除く、省略時の設定) 引数をアドレスで渡します。形式は次のとおりです。

CALL *routine-name* (%ADDR *address-expression*)

デバッガはアドレス式を評価し、そのアドレスを指定されたルーチンに渡します。(X などの) 単純な変数の場合、X のアドレスがルーチンに渡されます。この受け渡し方法は Fortran が ROUTINE(X) をインプリメントする方法です。つまり、命名された変数の場合、%ADDR の使用は Fortran での参照による呼び出しに対応します。ただし他の式の場合には、参照によって呼び出すには %REF 関数を使用しなければなりません。(配列、レコード、アクセス・タイプのように) 複雑な変数または複合変数の場合、%ADDR を指定するとアドレスが渡されますが、呼び出されたルーチンは渡されたデータを正しく処理しない場合があります。% ADDR を使用するときは、リテラル値 (数字または数字からなる式) は指定できません。

**%DESCR** 引数をディスクリプタ渡します。形式は次のとおりです。

CALL *routine-name* (%DESCR *language-expression*)

デバッガは言語式を評価し、値を記述するための標準ディスクリプタを作成します。作成したディスクリプタが指定されたルーチンに渡されます。この方法は文字列を FORT RAN ルーチンに渡す場合などに使用できます。

**%REF** 引数を参照渡します。形式は次のとおりです。

CALL *routine-name* (%REF *language-expression*)

デバッガは言語式を評価し、評価結果の値を指すポインタを呼び出し先ルーチンに渡します。この受け渡し方法は Fortran が式の結果を渡す方法に対応します。

**%VAL** (C および C++ の省略時の値) 引数を値渡します。形式は次のとおりです。

CALL *routine-name* (%VAL *language-expression*)

デバッガは言語式を評価し、値を呼び出し先ルーチンに直接渡します。

---

## 修飾子

/AST (省略時の設定)

/NOAST

呼び出し先ルーチンの実行時に非同期システム・トラップ (AST) の実行要求を許可するか禁止するかを制御します。/AST 修飾子を指定すると、呼び出し先ルーチンでの AST の実行要求は許可されます。/NOAST 修飾子を指定すると、呼び出し先ルーチンでの AST の実行要求は禁止されます。CALL コマンドで/AST と/NOAST のどちらも指定しないと、DISABLE AST コマンドを以前に入力していた場合を除き、AST の実行要求は許可されます。

/SAVE\_VECTOR\_STATE

/NOSAVE\_VECTOR\_STATE (省略時の設定)

VAX のベクタ化されたプログラムに適用されます。ベクタ型プロセッサの現在の状態を保存したあと、CALL コマンドでルーチンを呼び出したときにその状態を復元するかどうかを制御します。

ベクタ型プロセッサの状態は次の情報で構成されます。

- ベクタ・レジスタ (V0 から V15) とベクタ制御レジスタ (VCR, VLR, VMR) の値
- 実行要求を保留にしている可能性のあるベクタ例外 (ベクタ命令の実行により引き起こされる例外)

ルーチンを実行するために CALL コマンドを使用すると、ルーチンの実行によって次のようにベクタ型プロセッサの状態が変わる可能性があります。

- ベクタ・レジスタまたはベクタ制御レジスタの値を変更する
- ベクタ例外を引き起こす
- CALL コマンドの実行時に保留中になっていたベクタ例外の実行要求を引き起こす

/SAVE\_VECTOR\_STATE 修飾子は呼び出し先ルーチンの実行が終了したあと、CALL コマンドの実行前に存在していたベクタ型プロセッサの状態を復元することをデバッガが指定します。これにより、呼び出し先ルーチンの実行が終了した後、次のことが保証されます。

- CALL コマンドの実行前に実行要求を保留していたベクタ例外はすべて実行要求を保留した状態のままになります。
- ルーチン呼び出し時に発生したベクタ例外は実行要求を保留した状態のままになりません。
- ベクタ・レジスタの値は CALL コマンドの実行前の値と同じになります。

/NOSAVE\_VECTOR\_STATE 修飾子 (これが省略時の設定) は、CALL コマンドの実行前に存在していたベクタ型プロセッサの状態を、呼び出し先ルーチンの実行が終了したあと、復元しないことをデバuggaが指定します。この場合、ルーチンの呼び出し後のベクタ型プロセッサの状態は、呼び出し先ルーチンによる作用があればそれによって決まります。

/[NO]SAVE\_VECTOR\_STATE 修飾子は汎用レジスタには全く影響しません。汎用レジスタの値は、CALL コマンドでルーチンが実行されると必ず保存され、復元されます。

---

## 説明

CALL コマンドはプログラムを実行するために使用できる 4 つのデバugga・コマンドのうちの 1 つです (他の 3 つは GO, STEP, EXIT)。CALL コマンドを使用すれば、プログラムの通常の実行とは別にルーチンを実行できます。CALL コマンドは、ルーチンがプログラムにリンクされていたものであるかぎり、プログラムに実際にそのルーチンへの呼び出しがあるかないかにかかわらずそのルーチンを実行します。

CALL コマンドを入力すると、デバuggaは次の処理を行います。詳しい説明は、修飾子の説明を参照してください。

1. 汎用レジスタの現在の値を保存します。
2. 引数リストを構成します。
3. コマンドで指定されたルーチンへの呼び出しを実行し、引数を引き渡します。
4. ルーチンを実行します。
5. 戻り状態レジスタ内のルーチンにより返される値を表示します。便宜上、呼び出し先ルーチンの実行後、R0 レジスタには関数戻り値 (ルーチンが関数の場合) またはプロシージャ完了ステータス (ルーチンが状態値を返すプロシージャの場合) が入ります。呼び出し先プロシージャが状態値も関数値も返さない場合には、R0 の値は意味がないことになり、"value returned"メッセージは無視してかまいません。
6. 汎用レジスタの値を CALL コマンドの実行前の値に復元します。
7. プロンプトを表示します。

デバuggaは呼び出し先ルーチンがプロシージャ呼び出し規則 (『OpenVMS Calling Standard』を参照) に従っているものとみなします。ただし、デバuggaはサポートされているあらゆる言語に対する引数の受け渡し方法全部についての知識を持っているわけではありません。したがって、パラメータの受け渡し方法、たとえば、CALL SUB1(X) ではなく CALL SUB1(%VAL X) を使用するなどを指定する必要があるかもしれません。ルーチンへ引数を引き渡す方法についての詳しい説明は、各言語のマニュアルを参照してください。

現在の言語が C または C++ の場合には、省略時の設定により、CALL コマンドは参照によってではなく、値によって引数を渡します。さらに、受け渡しメカニズム・レキシカル (%REF や %VAL など) を使用せずに、次の引数を渡すことができます。

- ルーチンの参照
- 引用符で囲まれた文字列 (%REF 文字列として取り扱われる)
- 構造体、レコード、オブジェクト
- F 浮動小数点、D 浮動小数点、G 浮動小数点、S 浮動小数点パラメータと、その型の変数を逆参照することによる T 浮動小数点形式

ルーチンに読み込み専用ではないパラメータが含まれている場合、パラメータに割り当てられている値は可視状態ではなく、値へのアクセスは確実ではありません。これは、デバッガが、プログラム引数ではなく内部引数リストのパラメータ値を調整するためです。値の変化を調べるには、パラメータの代わりに静的変数を使用してみてください。

CALL コマンドは、すべての浮動小数点リテラルを、VAX システムと Alpha システムでは F 浮動小数点形式に、I64 では T 浮動小数点形式に変換します。

VAX と Alpha では、浮動小数点リテラルの引き渡しは、F 浮動小数点以外の形式ではサポートされていません (後述の例を参照)。

SET BREAK/EXCEPTION コマンドまたは STEP/EXCEPTION コマンドにより発生する例外ブレークポイントでの一般的なデバッグ方法は、CALL コマンドでダンプ・ルーチンを呼び出すことです。例外ブレークポイントで CALL コマンドを入力すると、呼び出し先ルーチンの中で以前に設定されていたブレークポイント、トレースポイントまたはウォッチポイントは、デバッガが例外コンテキストを失わないようにするために一時的に無効になります。ただし、例外ブレークポイント以外の位置で CALL コマンドを入力した場合には、このようなイベントポイントはアクティブになります。

例外ブレークポイントが検出されると、アプリケーションで宣言された条件ハンドラが起動される前に例外が中断されます。例外ブレークポイントでは、CALL コマンドでルーチンを実行したあとに GO コマンドまたは STEP コマンドを入力すると、デバッガが例外を再度シグナル通知します (GO コマンドと STEP コマンドを参照)。

Alpha プロセッサでは、CALL コマンドで起動されたルーチンの前に起動されたルーチンはデバッグできません。たとえば、プログラムを MAIN ルーチンで終了し、ブレークポイントを SORT ルーチンに設定するとします。そこでデバッガ・コマンドの CALL SORT を実行します。SORT ルーチンのデバッグ中は、MAIN ルーチンはデバッグできません。まず SORT ルーチンへの呼び出しから戻らなければなりません。

マルチプロセス・プログラムをデバッグしている場合、CALL コマンドは、現在のプロセス・セットのコンテキストで実行されます。また、マルチプロセス・プログラムをデバッグしているときには、実行がプロセスの中でどのように継続されるかは、SET MODE [NO]INTERRUPT コマンドや SET MODE [NO]WAIT コマンドを入力したかどうかによって決まります。省略時の設定 (SET MODE NOINTERRUPT) では、いずれかのプロセスが停止したときに、デバッガは他のプロセスに対して何もアクションをとりません。また、省略時の設定 (SET MODE WAIT) では、デバッガは、現在のプロセス・セット内のすべてのプロセスが停止するまで、次のコマンドの入力を受け付けません。詳細は、『OpenVMS デバッガ説明書』の第 15 章を参照してください。

#### 関連コマンド

GO  
EXIT  
SET PROCESS  
SET MODE [NO]INTERRUPT  
SET VECTOR\_MODE [NO]SYNCHRONIZED (VAX のみ)  
STEP  
SYNCHRONIZE VECTOR\_MODE (VAX のみ)

---

#### 例

1. DBG> CALL SUB1(X)  
value returned is 19  
DBG>

このコマンドは、パラメータ X を使用してサブルーチン SUB1 を呼び出します (省略時の設定により、X のアドレスが渡されます)。このルーチンは 19 という値を戻します。

2. DBG> CALL SUB(%REF 1)  
value returned is 1  
DBG>

この例は、数値リテラル 1 のある記憶位置へのポインタを SUB ルーチンに渡します。

3. DBG> SET MODULE SHARE\$LIBRTL  
DBG> CALL LIB\$SHOW\_VM  
1785 calls to LIB\$GET\_VM, 284 calls to LIB\$FREE\_VM, 122216 bytes  
still allocated, value returned is 00000001  
DBG>

この例は、(共用可能イメージ LIBRTL の) LIB\$SHOW\_VM 実行時ライブラリ・ルーチン呼び出して、メモリ統計情報を表示しています。SET MODULE コマンドは LIBRTL 内のユニバーサル・シンボル (ルーチン名) をメイン・イメージの

中で可視状態にします。詳しい説明は SHOW MODULE/SHARE コマンドを参照してください。

4. DBG> CALL testsub (%val 11.11, %val 22.22, %val 33.33)

この例は、関数プロトタイプ `void testsub (float, float, float)` を持つ C サブルーチンを仮定した、値による浮動小数点パラメータの引き渡しています。浮動小数点パラメータは、F 浮動形式で渡されます。

5. SUBROUTINE CHECK\_TEMP(TEMPERATURE,ERROR\_MESSAGE)  
 REAL TOLERANCE /4.7/  
 REAL TARGET\_TEMP /92.0/  
 CHARACTER\*(\*) ERROR\_MESSAGE  
  
 IF (TEMPERATURE .GT. (TARGET\_TEMP + TOLERANCE)) THEN  
 TYPE \*, 'Input temperature out of range:',TEMPERATURE  
 TYPE \*,ERROR\_MESSAGE  
 ELSE  
 TYPE \*, 'Input temperature in range:',TEMPERATURE  
 END IF  
 RETURN  
END  
  
DBG> CALL CHECK\_TEMP(%REF 100.0, %DESCR 'TOLERANCE-CHECK 1 FAILED')  
Input temperature out of range: 100.0000  
TOLERANCE-CHECK 1 FAILED  
value returned is 0  
DBG> CALL CHECK\_TEMP(%REF 95.2, %DESCR 'TOLERANCE-CHECK 2 FAILED')  
Input temperature in range: 95.2000  
value returned is 0  
DBG>

この例のソース・コードは、2つのパラメータ、TEMPERATURE (実数) と ERROR\_MESSAGE (文字列) を受け付ける Fortran ルーチン (CHECK\_TEMP) のソース・コードです。ルーチンは、温度の値によってさまざまな出力を表示します。2つの CALL コマンドはそれぞれ参照で温度値とディスクリプタでエラー・メッセージを渡します。このルーチンには正式な戻り値がないため、返される値は未定義となり、この場合には0になります。

---

## CANCEL ALL

ブレークポイント、トレースポイント、ウォッチポイントをすべて取り消します。有効範囲と型をそれぞれの省略時の値に戻します。SET MODE コマンドで設定された行モード、シンボリック・モード、G 浮動モードをそれぞれの省略時の値に戻します。

---

### フォーマット

CANCEL ALL

---

### 修飾子

/PREDEFINED

定義済みのブレークポイント、トレースポイントをすべて取り消します (ただしユーザ定義のものは取り消しません)。

/USER

ユーザ定義のブレークポイント、トレースポイント、ウォッチポイントをすべて取り消します (ただし定義済みのものは取り消さない)。/PREDEFINED を指定した場合を除き、これが省略時の設定です。CANCEL ALL コマンドは次のステップを行います。

---

### 説明

CANCEL ALL コマンドは次のことを行います。

1. SET BREAK コマンド、SET TRACE コマンド、SET WATCH コマンドで作成されたユーザ定義イベントポイントのすべてを取り消します。これは CANCEL BREAK/ALL コマンド、CANCEL TRACE/ALL コマンド、CANCEL WATCH /ALL コマンドを入力することと同じです。プログラムの種類によっては (たとえば Ada プログラムやマルチプロセス・プログラム)、デバグガを起動すると、特定の定義済みのブレークポイントまたはトレースポイントが自動的に設定される場合があります。定義済みで、ユーザ定義ではないすべてのイベントポイントを取り消すには、CANCEL ALL/PREDEFINED を使用します。定義済みで、ユーザ定義のイベントポイントをすべて取り消すには、CANCEL ALL/PREDEFINED /USER を使用します。
2. 有効範囲検索リストをその省略時の値 (0,1,2, ...,  $n$ ) に戻します。これは CANCEL SCOPE コマンドを入力することと同じです。



3. コンパイラ生成型に対応する記憶位置のデータ型を対応する型に戻します。コンパイラ生成型に対応しない記憶位置の型は"ロングワード整数"に戻します。これは CANCEL TYPE/OVERRIDE コマンドと SET TYPE LONGWORD コマンドを入力することと同じです。
4. SET MODE コマンドで設定された行モード、シンボリック・モード、G 浮動モードをそれぞれの省略時の値に戻します。これは次のコマンドを入力することと同じです。

```
DBG> SET MODE LINE,SYMBOLIC,NOG_FLOAT
```

CANCEL ALL コマンドは実行時シンボル・テーブルに含まれている現在の言語設定やモジュールには影響しません。

#### 関連コマンド

```
(CANCEL,DEACTIVATE) BREAK
CANCEL SCOPE
(CANCEL,DEACTIVATE) TRACE
CANCEL TYPE/OVERRIDE
(CANCEL,DEACTIVATE) WATCH
(SET,CANCEL) MODE
SET TYPE
```

---

## 例

1. DBG> CANCEL ALL

このコマンドはすべてのユーザ定義のブレークポイントとトレースポイント、すべてのウォッチポイントを取り消し、有効範囲、型、一定のモードをそれぞれの省略時の値に戻します。この例では、定義済みのブレークポイント、トレースポイントはありません。

2. DBG> CANCEL ALL

```
%DEBUG-I-PREDEPTNOT, predefined eventpoint(s) not canceled
```

このコマンドはすべてのユーザ定義のブレークポイントとトレースポイント、すべてのウォッチポイントを取り消し、有効範囲、タイプ、一定のモードをそれぞれの省略時の値に戻します。この例では、定義済みのブレークポイント、トレースポイントがいくつかあります。それらは省略時の設定では取り消されません。

3. DBG> CANCEL ALL/PREDEFINED

このコマンドは定義済みのブレークポイント、トレースポイントをすべて取り消し、有効範囲、タイプ、一定のモードをそれぞれの省略時の値に戻します。ユーザ定義のブレークポイントやトレースポイントは影響を受けません。

---

## CANCEL BREAK

ブレークポイントを取り消します。

---

### フォーマット

CANCEL BREAK *[address-expression[, ... ]]*

---

### パラメータ

address-expression

取り消すブレークポイントを指定します。ワイルドカード文字のアスタリスク(\*)は使用できません。代わりに/ALL 修飾子を使用してください。/EVENT , /PREDEFINED , /USER 以外の修飾子を使用する場合は、アドレス式は指定できません。

---

### 修飾子

/ACTIVATING

前回の SET BREAK/ACTIVATING コマンドの作用を取り消します。

/ALL

省略時の設定では、すべてのユーザ定義ブレークポイントを取り消します。  
/PREDEFINED といっしょに使用すると、定義済みブレークポイントはすべて取り消されますが、ユーザ定義ブレークポイントは取り消されません。すべてのブレークポイントを取り消すには、CANCEL BREAK/ALL/USER/PREDEFINED を使用します。

/BRANCH

前回の SET BREAK/BRANCH コマンドの作用を取り消します。

/CALL

前回の SET BREAK/CALL コマンドの作用を取り消します。

/EVENT=event-name

前回の SET BREAK/EVENT=event-name コマンドの作用を取り消します。イベント名 (および必要であればアドレス式) は SET BREAK/EVENT コマンドで指定したと  
おりに指定してください。現在のイベント機能とそれに対応するイベント名を表示するには、SHOW EVENT\_FACILITY コマンドを使用してください。

/EXCEPTION

前回の SET BREAK/EXCEPTION コマンドの作用を取り消します。

**/HANDLER**

前回の SET BREAK/HANDLER コマンドの作用を取り消します。

**/INSTRUCTION**

前回の SET BREAK/INSTRUCTION コマンドの作用を取り消します。

**/LINE**

前回の SET BREAK/LINE コマンドの作用を取り消します。

**/PREDEFINED**

ユーザ定義ブレークポイントには全く影響を及ぼさずに指定の定義済みブレークポイントを取り消します。/ALL といっしょに使用すると、すべての定義済みブレークポイントが取り消されます。

**/SYSEMULATE**

(Alpha のみ) 前回の SET BREAK/SYSEMULATE コマンドの作用を取り消します。

**/TERMINATING**

前回の SET BREAK/TERMINATING コマンドの作用を取り消します。

**/UNALIGNED\_DATA**

(Alpha のみ) 前回の SET BREAK/UNALIGNED\_DATA コマンドの作用を取り消します。

**/USER**

定義済みブレークポイントには全く影響を及ぼさずに指定のユーザ定義ブレークポイントを取り消します。/PREDEFINED を指定した場合を除き、これが省略時の設定です。すべてのユーザ定義ブレークポイントを取り消すには、/ALL 修飾子を使用します。

---

## 説明

ブレークポイントには、ユーザが定義するものと定義済みのものとがあります。ユーザ定義のブレークポイントは、SET BREAK コマンドで明示的に設定したブレークポイントです。定義済みのブレークポイントは、デバッグするプログラムの種類 (Ada あるいは ZQUIT マルチプロセスなど) によって異なりますが、デバッガの起動時に自動的に設定されます。現在設定されているすべてのブレークポイントを表示するには、SHOW BREAK コマンドを使用します。定義済みのブレークポイントは定義済みのものとして表示されます。

ユーザ定義ブレークポイントと定義済みブレークポイントは、それぞれ別々に設定したり取り消したりします。たとえば、1 つの記憶位置またはイベントに、ユーザ定義ブレークポイントと定義済みブレークポイントの両方を設定することができます。ユーザ定義ブレークポイントを取り消しても、定義済みブレークポイントは影響を受けません。逆も同様です。

ユーザ定義ブレークポイントだけを取り消すには、CANCEL BREAK コマンドを指定するときに/PREDEFINED を指定しないでください(省略時の設定は/USER)。定義済みブレークポイントだけを取り消すには、/USER ではなく/PREDEFINED を指定します。定義済みブレークポイントとユーザ定義ブレークポイントを両方とも取り消すには、/PREDEFINED と/USER を両方指定します。

通常、SET BREAK コマンドはユーザ定義ブレークポイントに対してだけ使用されますが、CANCEL BREAK コマンドの作用はSET BREAK コマンドの作用の反対です。したがって、特定の記憶位置に設定されたブレークポイントを取り消すには、CANCEL BREAK コマンドでそれと同じ記憶位置(アドレス式)を指定します。命令またはイベントのクラスに対して設定されたブレークポイントを取り消すには、対応する修飾子(/LINE、/BRANCH、/ACTIVATING、/EVENT=など)を使用して命令またはイベントのクラスを指定します。詳しくは、修飾子の説明を参照してください。

ブレークポイントを取り消す手間をかけずに、デバッガにそのブレークポイントを無視させたい場合(たとえば、プログラムを再実行するときにブレークポイントを使用したり、使用しない場合)には、CANCEL BREAK コマンドではなく DEACTIVATE BREAK コマンドを使用してください。あとで、(ACTIVATE BREAK を使用して)そのブレークポイントを有効にすることができます。

#### 関連コマンド

```
(ACTIVATE,DEACTIVATE) BREAK
CANCEL ALL
(SET,SHOW) BREAK
(SET,SHOW) EVENT_FACILITY
(SET,SHOW,CANCEL) TRACE
```

---

#### 例

1. DBG> CANCEL BREAK MAIN\LOOP+10  
このコマンドはアドレス式 MAIN\LOOP+10 で設定されたユーザ定義ブレークポイントを取り消します。
2. DBG> CANCEL BREAK/ALL  
このコマンドはユーザ定義ブレークポイントをすべて取り消します。
3. DBG> CANCEL BREAK/ALL/USER/PREDEFINED  
このコマンドはユーザ定義ブレークポイントと定義済みブレークポイントをすべて取り消します。

4. `all> CANCEL BREAK/ACTIVATING`

このコマンドは前回のユーザ定義 `SET BREAK/ACTIVATING` コマンドを取り消します。この結果、デバッガは新しいプロセスがデバッガの制御下に入っても実行を一時停止しません。

5. `DBG> CANCEL BREAK/EVENT=EXCEPTION_TERMINATED/PREDEFINED`

このコマンドは未処理例外が原因でタスク終端に設定された定義済みブレークポイントを取り消します。このブレークポイントは Ada プログラムと POSIX Threads または Ada ルーチンを呼び出すプログラムに対しては定義済みのものです。

---

## CANCEL DISPLAY

画面表示を永久に削除します。

---

### 注意

このコマンドは、デバッガへの HP DECwindows Motif for OpenVMS ユーザ・インタフェースでは使用できません。

---

---

## フォーマット

CANCEL DISPLAY *[display-name[, . . . ]]*

---

## パラメータ

*display-name*

取り消す表示の名前を指定します。PROMPT 表示は取り消すことができないため、指定できません。ワイルドカード文字のアスタリスク(\*)は使用できません。代わりに/ALL 修飾子を使用してください。/ALL を指定する場合は、ディスプレイ名は指定できません。

---

## 修飾子

/ALL

PROMPT 表示を除くすべての表示を取り消します。

---

## 説明

表示を取り消すと、その内容は永久に失われ、表示リストから削除され、その表示に割り当てられたメモリはすべて解放されます。

PROMPT 表示は取り消せません。

関連コマンド

(SHOW) DISPLAY

(SET,SHOW,CANCEL) WINDOW

---

## 例

1. `DBG> CANCEL DISPLAY SRC2`

このコマンドは SRC2 表示を削除します。

2. `DBG> CANCEL DISPLAY/ALL`

このコマンドは PROMPT 表示を除き，すべての表示を削除します。

---

## CANCEL MODE

SET MODE コマンドにより設定された行モード，シンボリック・モード，G 浮動モードをそれぞれの省略時の値に戻します。また，入出力値の基数も省略時の設定に戻します。

---

### 注意

このコマンドは，デバッガへの HP DECwindows Motif for OpenVMS ユーザ・インタフェースでは使用できません。

---

---

## フォーマット

CANCEL MODE

---

## 説明

CANCEL MODE コマンドの作用は次のコマンドと同じです。

```
DBG> SET MODE LINE,SYMBOLIC,NOG_FLOAT
DBG> CANCEL RADIX
```

ほとんどの言語の場合，データの入力と表示の両方に対する省略時の基数は 10 進数です。

VAX プロセッサでは，例外は BLISS と MACRO-32 です。この 2 つの省略時の基数は 16 進数です。

Alpha プロセッサでは，例外は BLISS と MACRO-32 と MACRO-64 です。この 3 つの省略時の基数は 16 進数です。

Intel Itanium プロセッサでは，例外は BLISS，MACRO，および Intel® Assembler (IAS) です。

### 関連コマンド

(SET,SHOW) MODE  
(SET,SHOW,CANCEL) RADIX



---

**例**

```
DBG> CANCEL MODE
```

このコマンドは省略時の基数モードとすべての省略時のモード値を復元します。

---

## CANCEL RADIX

整数データの入力と表示のための基数モードを省略時の設定に戻します。

---

### フォーマット

CANCEL RADIX

---

### 修飾子

/OVERRIDE

前回の SET RADIX/OVERRIDE コマンドで設定された変更型の基数を上書きします。現在の変更型の基数を "none" に設定し、出力基数モードを前回の SET RADIX コマンドまたは SET RADIX/OUTPUT コマンドで設定された値に戻します。SET RADIX コマンドまたは SET RADIX/OUTPUT コマンドを使用して基数モードを変更していなかった場合は、CANCEL RADIX/OVERRIDE コマンドは基数モードをその省略時の設定に戻します。

---

### 説明

CANCEL RADIX コマンドは前回の SET RADIX コマンドと SET RADIX /OVERRIDE コマンドの作用を取り消します。入力と出力の基数をそれぞれの省略時の設定に戻します。

ほとんどの言語の場合、データの入力と表示の両方に対する省略時の基数は 10 進数です。例外は BLISS と MACRO です。これらの言語での省略時の基数は 16 進数です。

CANCEL RADIX/OVERRIDE コマンドの作用には制限がもっとあります。この作用については/OVERRIDE 修飾子で説明します。

関連コマンド

EVALUATE  
(SET,SHOW) RADIX

---

## 例

1. DBG> CANCEL RADIX

このコマンドは省略時の入出力の基数を復元します。

2. DBG> CANCEL RADIX/OVERRIDE

このコマンドはSET RADIX/OVERRIDE コマンドで設定していた可能性のある変更型の基数を上書きします。

---

## CANCEL SCOPE

シンボル検索のための有効範囲検索リストを省略時の設定に戻します。

---

### フォーマット

CANCEL SCOPE

---

### 説明

CANCEL SCOPE コマンドは前回の SET SCOPE コマンドで設定された現在の有効範囲の検索リストを取り消し、省略時の有効範囲検索リスト、すなわち 0,1,2, ...,  $n$  を有効にします。ここで  $n$  は呼び出しスタック内の呼び出しの数です。

省略時の有効範囲検索リストは、パス名接頭識別子が付かないシンボルの場合、EXAMINE X のようなシンボル検索はまず現在実行中のルーチン (有効範囲 0) で X を探します。そこで X が可視状態になっていない場合、デバッガはそのルーチンの呼び出し元 (有効範囲 1) を探し、呼び出しスタックを順におりていきます。X が有効範囲  $n$  で見つからないと、デバッガは実行時シンボル・テーブル (RST) の残りを検索し、必要ならばグローバル・シンボル・テーブル (GST) を検索します。

関連コマンド

(SET,SHOW) SCOPE

---

### 例

DBG> CANCEL SCOPE

このコマンドは現在の有効範囲を取り消します。

---

## CANCEL SOURCE

前回の SET SOURCE コマンドで設定された、ソース・ディレクトリの検索リストと検索方法のどちらか一方、あるいは両方を取り消します。

---

### フォーマット

CANCEL SOURCE

---

### 修飾子

/DISPLAY

前回の SET SOURCE/DISPLAY コマンドの作用を取り消します。SET SOURCE/DISPLAY コマンドは、ソース・コードを表示するときにデバッガが使用するディレクトリ検索リストを指定します。このコマンドの作用を取り消すと、デバッガはコンパイルされたディレクトリからソース・ファイルを検索するようになります。

/EDIT

前回の SET SOURCE/EDIT コマンド (デバッガが EDIT コマンドの実行中に使用するディレクトリ検索リストを指定するコマンド) の作用を取り消します。CANCEL SOURCE/EDIT により、デバッガはソース・ファイルをコンパイル時のディレクトリから検索するようになります。

/EXACT

前回の SET SOURCE/EXACT コマンド (ディレクトリ検索方法を指定するコマンド) の作用を取り消します。CANCEL SOURCE/EXACT により、デバッガはコンパイル以降の正確に一致するバージョンのソース・ファイルの検索は行わなくなり、省略時の検索方法である最新バージョンのファイル検索に戻ります。

/LATEST

前回の SET SOURCE/LATEST コマンド (ディレクトリ検索方法を指定するコマンド) の作用を取り消します。CANCEL SOURCE/LATEST により、デバッガはコンパイル以降の正確に一致するバージョンのソース・ファイルを検索するようになります。/LATEST は省略時の設定であるため、この修飾子は他の修飾子 (たとえば/MODULE) と一緒に使用した場合にのみ意味があります。

/MODULE=module-name

同じモジュール名と修飾子が指定されていた前回の SET SOURCE /MODULE=module-name コマンドを取り消します。/MODULE 修飾子を使用すると、指定されたモジュールに対して個別のディレクトリ検索リストとディレクトリ検索方法のどちらか一方、あるいは両方を指定することができます。SET SOURCE /MODULE コマンドや CANCEL SOURCE/MODULE コマンドには、上に挙げた修飾子を追加することができます。

他の修飾子を追加して CANCEL SOURCE/MODULE コマンドを実行すると、指定された各修飾子がモジュールに与える作用が取り消されます。他の修飾子を追加せずに CANCEL SOURCE/MODULE コマンドを実行すると、デバッガは指定されていたモジュールをディレクトリ内の他のモジュールと区別しなくなります。

/ORIGINAL

STDL プログラムにだけ適用されます。レイヤード・プロダクトであるコリレーション・ファシリティ (Correlation Facility) をインストールする必要があり、保持デバッガ (Kept Debugger) を起動する必要があります。前回の SET SOURCE/ORIGINAL コマンドの作用を取り消します。SET SOURCE/ORIGINAL コマンドは、STDL ソース・ファイルをデバッグしなければならず、他の言語で作成されたソース・ファイルをデバッグする場合には取り消さなければなりません。

---

## 説明

CANCEL SOURCE は前回の SET SOURCE コマンドの作用を取り消します。この取り消しの種類は、前回の SET SOURCE コマンドによって有効にされた修飾子によって決まります。CANCEL SOURCE と SET SOURCE の相互関係については、CANCEL SOURCE の例を参照してください。

SET SOURCE コマンドを実行する場合、2 つの修飾子 /LATEST と /EXACT のいずれかが必ず有効でなければなりません。これらの修飾子は、デバッガの検索方法に影響を与えます。/LATEST 修飾子は、最後に作成されたバージョン (ディレクトリ内の一番大きい番号のバージョン) を検索するようデバッガに指示します。/EXACT 修飾子は、最後にコンパイルされたバージョン (コンパイル時に作成されたデバッガ・シンボル・テーブルに記録されているバージョン) を検索するようデバッガに指示します。たとえば、SET SOURCE/LATEST コマンドは SORT.FOR;3 を検索し、SET SOURCE/EXACT コマンドは SORT.FOR;1 を検索するという具合です。

/DISPLAY または /EDIT 修飾子が指定されていない CANCEL SOURCE は、SET SOURCE/DISPLAY と SET SOURCE/EDIT の両方が前に指定されていた場合、これらのコマンドの作用を取り消します。

表示するファイルがコンパイル・ディレクトリに存在しない場合には、/DISPLAY 修飾子が必要です。

ソース・コードの表示用に使用されたファイルが編集可能ファイルと異なっている場合には、/EDIT 修飾子が必要です。Ada プログラムを使用する場合がそうです。Ada プログラムの場合、(SET, SHOW, CANCEL) SOURCE コマンドはソース表示に使用するファイル (Ada プログラム・ライブラリ内の"複写された"ソース・ファイル) の検索に影響します。(SET, SHOW, CANCEL) SOURCE/EDIT コマンドは、EDIT コマンドを使用するときに編集するソース・ファイルの検索に影響します。

Ada プログラムに特有の情報については、HELP Language\_Support Adaをタイプしてください。

関連コマンド

(SET,SHOW) SOURCE

---

## 例

```
1. DBG> SET SOURCE/MODULE=CTEST/EXACT [],SYSTEM::DEVICE:[PROJD]
DBG> SET SOURCE [PROJA],[PROJB],[PETER.PROJC]
...
```

```
DBG> SHOW SOURCE
source directory search list for CTEST,
match the exact source file version:
[]
SYSTEM::DEVICE:[PROJD]
source directory list for all other modules,
match the latest source file version:
[PROJA]
[PROJB]
[PETER.PROJC]
```

```
DBG> CANCEL SOURCE
DBG> SHOW SOURCE
source directory search list for CTEST,
match the exact source file version:
[]
SYSTEM::DEVICE:[PROJD]
all other source files will try to match
the latest source file version
```

この例ではSET SOURCE コマンドによって、CTEST 以外のソース・ファイルにディレクトリの検索リストと検索方法(省略時の最新バージョン検索)を設定しています。CANCEL SOURCE コマンドは、ディレクトリ検索リストは取り消しますが、検索方法は取り消しません。

```
2. DBG> SET SOURCE/MODULE=CTEST/EXACT [],SYSTEM::DEVICE:[PROJD]
DBG> SET SOURCE [PROJA],[PROJB],[PETER.PROJC]
...
```

```
DBG> SHOW SOURCE
source directory search list for CTEST,
match the exact source file version:
[]
SYSTEM::DEVICE:[PROJD]
source directory list for all other modules,
match the latest source file version:
[PROJA]
[PROJB]
[PETER.PROJC]
```

## CANCEL SOURCE

```
DBG> CANCEL SOURCE/MODULE=CTEST/EXACT
DBG> SHOW SOURCE
source directory search list for CTEST,
match the latest source file version:
[]
SYSTEM::DEVICE:[PROJD]
source directory list for all other modules,
match the latest source file version:
[PROJA]
[PROJB]
[PETER.PROJC]
DBG> CANCEL SOURCE/MODULE=CTEST
DBG> SHOW SOURCE
source directory list for all modules,
match the latest source file version:
[PROJA]
[PROJB]
[PETER.PROJC]
```

この例では SET SOURCE/MODULE=CTEST/EXACT コマンドによって、ソース・ファイル CTEST にディレクトリの検索リストと検索方法 (一致バージョンの検索) を設定しています。CANCEL SOURCE/MODULE=CTEST/EXACT コマンドは、CTEST の検索方法を取り消して省略時の最新バージョン検索に戻しています。また、CANCEL SOURCE/MODULE=CTEST コマンドは、CTEST のディレクトリ検索リストを取り消しています。

```
3. DBG> SET SOURCE /EXACT
DBG> SHOW SOURCE
no directory search list in effect,
match the exact source file
DBG> SET SOURCE [JONES]
DBG> SHOW SOURCE
source directory list for all modules,
match the exact source file version:
[JONES]
DBG> CANCEL SOURCE /EXACT
DBG> SHOW SOURCE
source directory list for all modules,
match the latest source file version:
[JONES]
```

この例では、最初の SET SOURCE/EXACT コマンドで設定した検索方法 (一致バージョンの検索) は、SET SOURCE [JONES] コマンドでもそのまま有効です。CANCEL SOURCE/EXACT コマンドは、SET SOURCE/EXACT コマンドを取り消し、さらに SET SOURCE [JONES] に影響を与えています。



---

# CANCEL TRACE

トレースポイントを取り消します。

---

## フォーマット

CANCEL TRACE *[address-expression[, ... ]]*

---

## パラメータ

address-expression

取り消すトレースポイントを指定します。ワイルドカード文字のアスタリスク(\*)は使用できません。代わりに/ALL 修飾子を使用してください。/EVENT , /PREDEFINED , /USER 以外の修飾子を使用すると、アドレス式は指定できません。

---

## 修飾子

/ACTIVATING

前回の SET TRACE/ACTIVATING コマンドの作用を取り消します。

/ALL

省略時の設定では、すべてのユーザ定義トレースポイントを取り消します。/PREDEFINED といっしょに使用すると、定義済みトレースポイントはすべて取り消されますが、ユーザ定義トレースポイントは取り消されません。すべてのトレースポイントを取り消すには、/ALL/USER/PREDEFINED を使用します。

/BRANCH

前回の SET TRACE/BRANCH コマンドの作用を取り消します。

/CALL

前回の SET TRACE/CALL コマンドの作用を取り消します。

/EVENT=event-name

前回の SET TRACE/EVENT=*event-name* コマンドの作用を取り消します。イベント名 (および必要であればアドレス式) は SET TRACE/EVENT コマンドで指定したとおり指定してください。現在のイベント機能とそれに対応するイベント名を表示するには、SHOW EVENT\_FACILITY コマンドを使用します。

/EXCEPTION

前回の SET TRACE/EXCEPTION コマンドの作用を取り消します。

### /INSTRUCTION

前回の SET TRACE/INSTRUCTION コマンドの作用を取り消します。

### /LINE

前回の SET TRACE/LINE コマンドの作用を取り消します。

### /PREDEFINED

ユーザ定義トレースポイントには全く影響を及ぼさずに指定の定義済みトレースポイントを取り消します。/ALL といっしょに使用すると、すべての定義済みトレースポイントが取り消されます。

### /TERMINATING

前回の SET TRACE/TERMINATING コマンドの作用を取り消します。

### /USER

定義済みトレースポイントには全く影響を及ぼさずに、指定のユーザ定義トレースポイントを取り消します。/PREDEFINED を指定した場合を除き、これが省略時の設定です。すべてのユーザ定義トレースポイントを取り消すには/ALL を使用します。

---

## 説明

トレースポイントには、ユーザが定義するものと定義済みのものとがあります。ユーザ定義のトレースポイントは、ユーザが SET TRACE コマンドで明示的に設定したトレースポイントです。定義済みのトレースポイントは、デバッグするプログラムの種類 (Ada あるいはマルチプロセスなど) によって異なりますが、デバッグの起動時に自動的に設定されます。現在設定されているすべてのトレースポイントを表示するには、SHOW TRACE コマンドを使用します。定義済みのトレースポイントは定義済みのものとして表示されます。

ユーザ定義トレースポイントと定義済みトレースポイントは、それぞれ別々に設定したり取り消したりします。たとえば、1 つの記憶位置またはイベントに、ユーザ定義トレースポイントと定義済みトレースポイントの両方を設定することができます。ユーザ定義トレースポイントを取り消しても、定義済みトレースポイントは影響を受けません。逆も同様です。

ユーザ定義トレースポイントだけを取り消すには、CANCEL TRACE コマンドを指定するときに/PREDEFINED を指定しないでください (省略時の設定は/USER)。定義済みトレースポイントだけを取り消すには、/USER ではなく/PREDEFINED を指定します。定義済みトレースポイントとユーザ定義トレースポイントを両方とも取り消すには、CANCEL TRACE/ALL/USER/PREDEFINED を使用します。

通常、SET TRACE コマンドはユーザ定義トレースポイントに対してだけ使用されますが、CANCEL TRACE コマンドの作用は SET TRACE コマンドの作用の反対です。したがって、特定の記憶位置に設定されたトレースポイントを取り消すには、CANCEL TRACE コマンドでそれと同じ記憶位置 (アドレス式) を指定します。命令

またはイベントのクラスに対して設定されたトレースポイントを取り消すには、対応する修飾子 (/LINE, /BRANCH, /ACTIVATING, /EVENT=など) を使用して命令またはイベントのクラスを指定します。詳しい説明は、修飾子の説明を参照してください。

デバッガがトレースポイントを一時的に無視するように設定し、しかもトレースポイントの定義はそのまま残しておきたい場合には、DEACTIVATE TRACE コマンドを使用します。トレースポイントは後で有効に設定できます (その場合は ACTIVATE TRACE を使用します)。

#### 関連コマンド

```
(ACTIVATE,DEACTIVATE,SET,SHOW) TRACE
CANCEL ALL
(SET,SHOW,CANCEL) BREAK
(SET,SHOW) EVENT_FACILITY
```

---

#### 例

1. DBG> CANCEL TRACE MAIN\LOOP+10

このコマンドは、MAIN\LOOP+10 の位置に設定されているユーザ定義のトレースポイントをキャンセルします。

2. DBG> CANCEL TRACE/ALL

このコマンドはすべてのユーザ定義トレースポイントを取り消します。

3. all> CANCEL TRACE/TERMINATING

このコマンドは前回の SET TRACE/TERMINATING コマンドを取り消します。取り消すと、プロセスがイメージの終了を行っても、ユーザ定義トレースポイントは検出されません。

4. DBG> CANCEL TRACE/EVENT=RUN %TASK 3

このコマンドはタスク 3 (タスク ID = 3) が RUN 状態に入ったときに検出されるように設定されたトレースポイントを取り消します。

---

## CANCEL TYPE/OVERRIDE

前回の SET TYPE/OVERRIDE コマンドで設定された変更型を上書きします。

---

### フォーマット

CANCEL TYPE/OVERRIDE

---

### 説明

CANCEL TYPE/OVERRIDE コマンドは現在の上書き型を "none" に設定します。その結果、コンパイラ生成型に対応するプログラム記憶位置はその型に従って解釈されます。

#### 関連コマンド

DEPOSIT  
EXAMINE  
(SET,SHOW) EVENT\_FACILITY  
(SET,SHOW) TYPE/OVERRIDE

---

### 例

DBG> CANCEL TYPE/OVERRIDE

このコマンドは前回の SET TYPE/OVERRIDE コマンドの作用を取り消します。

---

# CANCEL WATCH

ウォッチポイントを取り消します。

---

## フォーマット

CANCEL WATCH *[address-expression[, ... ]]*

---

## パラメータ

address-expression

取り消すウォッチポイントを指定します。高級言語を使用する場合、これは通常変数の名前になります。ワイルドカード文字のアスタリスク(\*)は使用できません。代わりに/ALL 修飾子を使用してください。/ALL を指定する場合は、アドレス式は指定できません。

---

## 修飾子

/ALL

すべてのウォッチポイントを取り消します。

---

## 説明

CANCEL WATCH コマンドの作用は SET WATCH コマンドの作用の反対です。SET WATCH コマンドで特定の記憶位置に設定されたウォッチポイントを取り消すには、CANCEL WATCH で同じ記憶位置を指定します。したがって、集合体全体に設定されたウォッチポイントを取り消すには、CANCEL WATCH コマンドでその集合体を指定します。集合体の 1 つの要素に設定されたウォッチポイントを取り消すには、CANCEL WATCH コマンドでその要素を指定します。

CANCEL ALL コマンドもすべてのウォッチポイントを取り消します。

ウォッチポイントを取り消す手間をかけずに、デバッガにそのウォッチポイントを見せたい場合には、DEACTIVATE WATCH コマンドを使用してください。あとで、ACTIVATE WATCH を使用してそのウォッチポイントを有効にすることができます。

関連コマンド

(ACTIVATE,DEACTIVATE,SET,SHOW) WATCH  
CANCEL ALL  
(SET,SHOW,CANCEL) BREAK  
(SET,SHOW,CANCEL) TRACE

---

例

1. DBG> CANCEL WATCH SUB2\TOTAL

このコマンドはモジュール SUB2 にある変数 TOTAL でのウォッチポイントを取り消します。

2. DBG> CANCEL WATCH/ALL

このコマンドはユーザが設定したすべてのウォッチポイントを取り消します。

---

# CANCEL WINDOW

画面ウィンドウ定義を永久に削除します。

---

## 注意

このコマンドは、デバッガへの HP DECwindows Motif for OpenVMS ユーザ・インタフェースでは使用できません。

---

---

## フォーマット

CANCEL WINDOW *[window-name], ... ]]*

---

## パラメータ

window-name

取り消す画面ウィンドウ定義の名前を指定します。ワイルドカード文字のアスタリスク(\*)は使用できません。代わりに/ALL 修飾子を使用してください。/ALL を指定する場合は、ウィンドウ定義の名前は指定できません。

---

## 修飾子

/ALL

すべての定義済みウィンドウ定義とユーザ定義ウィンドウ定義を取り消します。

---

## 説明

ウィンドウ定義を取り消すと、その名前は DISPLAY コマンドで使用できなくなります。CANCEL WINDOW コマンドは表示には全く影響を及ぼしません。

関連コマンド

(SHOW,CANCEL) DISPLAY  
(SET,SHOW) WATCH

---

### 例

```
DBG> CANCEL WINDOW MIDDLE
```

このコマンドは画面ウィンドウ定義の MIDDLE を永久に削除します。



---

## CONNECT

(保持デバッガのみ。) 他のプロセスでデバッガによる制御を受けずに実行されているイメージに割り込みをかけ、そのプロセスをデバッガの制御下に置きます。パラメータを指定しないで CONNECT を使用すると、CONNECT はデバッガに接続されるのを待っている作成されたプロセスすべてをデバッガの制御下に置きます。

Alpha システムの場合、CONNECT コマンドは、Alpha オペレーティング・システムを稼動しているターゲット・システムを OpenVMS Alpha システム・コード・デバッガの制御下に置きます。OpenVMS Alpha システム・コード・デバッガは、OpenVMS デバッガから起動するカーネル・デバッガです。

I64 システムでは、CONNECT デバッガ・コマンドを使って、I64 オペレーティング・システムが稼動しているターゲット・システムを OpenVMS I64 システム・コード・デバッガの制御下に置くこともできます。OpenVMS I64 システム・コード・デバッガは、OpenVMS デバッガから起動するカーネル・デバッガです。

CONNECT コマンドを使用して Alpha オペレーティング・システムをデバッグする場合は、このコマンドを実行する前に、『OpenVMS System Analysis Tools Manual』の "System Code Debugger" の章で説明されている指示を完了していなければなりません。これらの指示には、Alpha デバイス・ドライバの作成や、OpenVMS Alpha システム・コード・デバッガを起動するコマンドの設定などがあります。また、OpenVMS デバッガを DCL コマンドの DEBUG/KEEP で起動していなければなりません。

---

### フォーマット

```
CONNECT  [process-spec]
CONNECT  %NODE_NAME node-name
```

---

### パラメータ

*process-spec*

割り込みをかけたいイメージが実行されているプロセスを指定します。指定するプロセスはデバッガが起動されたプロセスと同じ OpenVMS ジョブにあるものでなければなりません。次のいずれかの形式で指定します。

*[%PROCESS\_NAME] proc-name*

スペースや小文字を全く含まない OpenVMS プロセス名。プロセス名にはワイルドカード文字のアスタリスク(\*)を含むこともできます。

`[%PROCESS_NAME] "proc-name"`

スペースや小文字を含む OpenVMS プロセス名。二重引用符(")の代わりに一重引用符(')を使用することもできます。

`%PROCESS_PID proc-id`

OpenVMS プロセス識別子 (PID, 16 進数)。

`node-name`

(Alpha または I64 のみ) Alpha オペレーティング・システムまたは I64 オペレーティング・システムをデバッグするときは、接続先のマシン (Alpha オペレーティング・システムまたは I64 オペレーティング・システムを稼動しているターゲット・マシン) のノード名を指定します。

---

## 修飾子

`/PASSWORD="password"`

(Alpha または I64 のみ) Alpha オペレーティング・システムまたは I64 オペレーティング・システムをデバッグするときは、接続先のマシン (Alpha オペレーティング・システムまたは I64 オペレーティング・システムを稼動しているターゲット・マシン) に対するパスワードを指定します。そのマシンに対してパスワードが設定されていない場合は、この修飾子を省略することができます。

`/IMAGE_PATH="image-path"`

(Alpha または I64 のみ) Alpha オペレーティング・システムをデバッグするときは、接続元のマシン (デバッガを実行しているホスト・マシン) に対するイメージ・パスを指定します。イメージ・パスは、システム・イメージの位置を指し示す論理名です。省略時の論理名は `DBGHK$IMAGE_PATH:` です。

---

## 説明

(保持デバッガのみ。) プロセスを指定するとき、CONNECT コマンドを使用すれば、指定するプロセスでデバッガによる制御を受けずに実行されているイメージに割り込みをかけ、そのプロセスをデバッガの制御下に置くことができます。このコマンドは、DCL コマンドの `RUN/NODEBUG` でデバッグ可能なイメージを実行している場合、またはプログラムがデバッガを起動しない `LIB$SPAWN` 実行時ライブラリ呼び出しを行う場合などに役立ちます。`$CREPRC` システム・サービス呼び出しで作成したプロセスには接続することはできません。

システムで実行中のデバッガのバージョンに応じて、ユーザは、ユーザが作成したプロセスとの接続に限定される場合もあり、また利用者識別コード (UIC) グループのメンバが作成したプロセスに接続することができる場合もあります。ときには、プロセスの作成前に `SYSGEN SECURITY_POLICY` パラメータを 8 に設定しなければならないことがあります。表 2-1 は、デバッガの個々のバージョンに適用される制限事項の一覧表です。

表 2-1 プロセス作成の制限事項 (デバグガのバージョン別)

デバグガのバージョン	接続可能なプロセス	SYSGEN パラメータ設定の有無
VAX バージョン 5.5-2 以前 , Alpha バージョン 6.0 以前	ユーザ自身が起動したプロセス	なし
VAX バージョン 6.0	ユーザ自身が起動したプロセス と、UIC グループのメンバが起動 したプロセス	あり
VAX バージョン 6.1 以降 , Alpha バージョン 6.1 以降	ユーザ自信が起動したプロセス と、UIC グループのメンバが起動 したプロセス	なし

デバグガ論理名 (JSY\$DEBUG, JSY\$DEBUGSHR, JSY\$DEBUGUIHR, JSY\$DBGTBKMSG, DBG\$PROCESS, JSY\$DBG\_HELP, JSY\$DBG\_UIHELP, DEBUGAPPCLASS, JSY\$VMSDEBUGUIL) がある場合、デバグガおよびターゲット・プロセスの両方で同一の定義に変換しなければなりません。

/DEBUG 修飾子付きでコンパイルおよびリンクしておかなければなりません。このイメージは、/NOTRACEBACK 修飾子付きでリンクしておくことはできません。

プロセスがデバグガの制御下に置かれたとき、イメージの実行は割り込みをかけられた時点で中断されます。

プロセスを指定しないと、CONNECT コマンドはデバグ・セッションに接続されるのを待っているプロセスすべてをデバグガの制御下に置きます。待機中のプロセスがない場合には、Ctrl/C を押せば CONNECT コマンドを強制終了することができます。

省略時の設定では、プロセスがデバグガの制御下に置かれると、トレースポイントが検出されます。この定義済みトレースポイントは SET TRACE/ACTIVATING コマンドを入力した結果生じるものと同じです。デバグガの制御下に置かれたプロセスはデバグガによって認識されるようになり、SHOW PROCESS コマンドで表示できるようになります。

CONNECT コマンドを使用して、デバグガの制御下にあるプロセスのサブプロセスに接続することはできません。そのようなサブプロセスに接続するには、SET PROCESS コマンドを使用してください。

#### 関連コマンド

DISCONNECT  
Ctrl/Y  
(SET,SHOW,CANCEL) TRACE

CONNECT コマンドを使用した OpenVMS オペレーティング・システムのデバッグ (Alpha および I64 のみ)

CONNECT コマンドを使用して OpenVMS システム・コード・デバッガ (SCD) で、Alpha オペレーティング・システムまたは I64 オペレーティング・システムのコードをデバッグできます。この機能には、ホストと呼ばれるシステムと、ターゲットと呼ばれるシステムの 2 つのシステムが必要です。ホストとターゲットは、同じオペレーティング・システム (Alpha または I64) を実行していなければなりません。ホストは、標準の OpenVMS システムとして構成します。このシステムから、DEBUG/KEEP を使用してデバッガを実行し、その後 CONNECT コマンドを入力します。ターゲットはスタンドアロン・システムで、SCD を有効にする特殊な方法でブートします。ホストとターゲットの間の通信は、イーサネット・ネットワークを通じて行います。

OpenVMS システム・コード・デバッガの使用についての詳細は、『OpenVMS System Analysis Tools Manual』を参照してください。

---

## 例

### 1. DBG\_1> CONNECT

このコマンドはデバッガに接続されるのを待っているプロセスをすべてデバッガの制御下に置きます。

### 2. DBG\_1> CONNECT JONES\_3

このコマンドは JONES\_3 というプロセスで実行されているイメージに割り込みをかけ、そのプロセスをデバッガの制御下に置きます。JONES\_3 プロセスはデバッガが起動されたプロセスと同じ UIC グループになければなりません。また、イメージは/NOTRACEBACK 修飾子でリンクされていたものであってはなりません。

### 3. DBG> CONNECT %NODE\_NAME SCDTST /PASSWORD="eager\_beaver" %DEBUG-I-NOLOCALS, image does not contain local symbols DBG>

この CONNECT コマンドは、OpenVMS オペレーティング・システムを実行しているターゲット・システムをデバッガの制御下に置きます。この例では、ターゲット・システムのノード名に SCDTST、パスワードに eager\_beaver を指定しています。

---

## Ctrl/C

デバッグ・セッション中に入力された場合、Ctrl/C はデバッグ・セッションを中断せずに、デバッガ・コマンドの実行を強制終了するか、プログラムの実行を中断します。

---

### 注意

デバッグ・セッション中は Ctrl/Y は使用できません。

---

---

## フォーマット

Ctrl/C

---

## 説明

Ctrl/C を押せば、デバッグ・セッションを中断せずに、デバッガ・コマンドの実行を強制終了するか、プログラムの実行を中断できます。この機能は、たとえばプログラムがブレークポイントのない無限ループを実行している場合や、完了するまでに長い時間を要するデバッガ・コマンドを強制終了したい場合に役立ちます。Ctrl/C を押すと、デバッガ・プロンプトが表示され、デバッガ・コマンドを入力できる状態になります。

プログラムがすでに Ctrl/C AST サービス・ルーチンを使用可能にしている場合には、SET ABORT\_KEY コマンドを使用して、デバッガの強制終了機能を他の Ctrl キー・シーケンスに割り当てます。ただし、多くの Ctrl キー・シーケンスの機能が定義済みであり、SET ABORT\_KEY コマンドを使用すれば、そのような定義も上書きできることに注意してください(『OpenVMS ユーザーズ・マニュアル』を参照)。オペレーティング・システムによって使用されていない Ctrl キー文字としては、G、K、N、P があります。

プログラムが Ctrl/C AST サービス・ルーチンを使用可能にしていない場合、デバッガの強制終了機能を他の Ctrl キー・シーケンスに割り当てると、Ctrl/C は Ctrl/Y と同じように動作します。つまり、デバッグ・セッションに割り込みをかけ、DCL レベルに戻します。

デバッグ・セッション中には Ctrl/Y は使用できません。代わりに Ctrl/C を使用するか、SET ABORT\_KEY コマンドで設定した同等の Ctrl キー・シーケンスを使用してください。

SPAWN コマンドと ATTACH コマンドを使用すれば、デバッグ・コンテキストを失わずに、デバッグ・セッションから移行したり、デバッグ・セッションに戻ったりすることができます。

#### 関連コマンド

ATTACH  
Ctrl/Y  
(SET,SHOW) ABORT\_KEY  
SPAWN

---

#### 例

```
DBG> GO
. . .
[Ctrl/C]
DBG> EXAMINE/BYTE 1000:101000 !should have typed 1000:1010
1000: 0
1004: 0
1008: 0
1012: 0
1016: 0
[Ctrl/C]
%DEBUG-W-ABORTED, command aborted by user request
DBG>
```

この例は、Ctrl/C を使用して、プログラム実行を中断し、デバッガ・コマンドの実行を強制終了する方法を示しています。

---

## Ctrl/W

Ctrl/W は (DISPLAY/REFRESH と同様に) 画面モードで画面を再表示します。

---

## フォーマット

Ctrl/W

---

## 説明

Ctrl/W についての詳しい説明は、DISPLAY コマンドの/REFRESH 修飾子の項目を参照してください。

---

## Ctrl/Y

DCL レベルから入力されると、Ctrl/Y はデバッグ制御を受けずに実行されているイメージに割り込みをかけ、ユーザが DCL コマンドの DEBUG でデバッグを起動できるようにします。

---

### 注意

デバッグ・セッション中は Ctrl/Y は使用できません。代わりに Ctrl/C を使用するか、SET ABORT\_KEY コマンドで設定した同等の強制終了キー・シーケンスを使用してください。

Ctrl/Y-DEBUG シーケンスでデバッグを起動したときは、デバッグの RUN コマンドも RERUN コマンドも使用できません。

---

---

## フォーマット

Ctrl/Y

---

## 説明

DCL レベルで Ctrl/Y を押すと、デバッグによる制御を受けずに実行しているイメージに割り込みをかけ、DCL コマンドの DEBUG でデバッグを起動できるようになります。

イメージをデバッグの制御下に置くことができるのは、最低でも、イメージが /TRACEBACK 修飾子 (/TRACEBACK は LINK コマンドの省略時の設定) でリンクされている場合だけです。

イメージの実行に割り込みをかけるために Ctrl/Y を押すと、制御が DCL に渡されます。そのあと DCL コマンドの DEBUG を入力すると、割り込みをかけられたイメージがデバッグの制御下に置かれます。デバッグはその言語固有のパラメータを、割り込みをかけられたモジュールのソース言語に設定し、プロンプトを表示します。プロンプトが表示されたら、ユーザは SHOW CALLS コマンドを入力することにより、どこで実行が中断されたのかを判断できます。

Ctrl/Y-DEBUG シーケンスは、保持デバッグ (Kept Debugger) の構成では使用できません。

Ctrl/Y-DEBUG シーケンスは、デバッグへの HP DECwindows Motif for OpenVMS ユーザ・インタフェースでは使用できません。代わりに、STOP ボタンを使用してください。



デバッグ・セッション中は、CONNECT コマンドを使用して、(同じジョブの) 他のプロセスでデバッガ制御を受けずに実行されているイメージをそのデバッグ・セッションに接続できます。

#### 関連コマンド

CONNECT  
Ctrl/C  
DEBUG (DCL コマンド)  
RUN (DCL コマンド)

---

#### 例

1. \$ RUN/NODEBUG TEST\_B

```

. . .
[Ctrl/Y]
Interrupt
$ DEBUG

```

Debugger Banner and Version Number

```

Language: ADA, Module: SWAP
DBG>

```

この例では、RUN/NODEBUG コマンドはデバッガによる制御を受けずに TEST\_B イメージを実行します。Ctrl/Y で割り込みをかけます。すると、DEBUG コマンドによってデバッガが起動されます。デバッガはバナーを表示し、割り込みをかけられたモジュール (SWAP) の言語 (この場合には Ada) に言語固有のパラメータを設定します。

2. \$ RUN/NODEBUG PROG2

```

. . .
[Ctrl/Y]
Interrupt
$ DEBUG

```

Debugger Banner and Version Number

```

Language: FORTRAN, Module: SUB4
predefined trace on activation at SUB4\%LINE 12 in %PROCESS_NUMBER 1
DBG>

```

この例では、DEFINE/JOB コマンドによってマルチプロセス・デバッグ構成が設定されます。RUN/NODEBUG コマンドはデバッガの制御を受けずに PROG2 イメージを実行します。Ctrl/Y-DEBUG シーケンスは実行に割り込みをかけ、デバッガを起動します。バナーは新しいデバッグ・セッションが開始されたことを示します。起動時トレースポイントはデバッガがプロセスを制御下に置いたときに実行が割り込みをかけられた場所を表します。

---

## Ctrl/Z

Ctrl/Z は (EXIT と同様に) デバッグ・セッションを終了します。

---

### フォーマット

Ctrl/Z

---

### 説明

Ctrl/Z についての詳しい説明は、EXIT コマンドを参照してください。

---

## DEACTIVATE BREAK

ブレークポイントを無効にします。そのブレークポイントはあとで有効にすることができます。

---

### フォーマット

```
DEACTIVATE BREAK [address-expression[, ... ]]
```

---

### パラメータ

address-expression

無効にするブレークポイントを指定します。ワイルドカード文字のアスタリスク(\*)は使用できません。代わりに/ALL 修飾子を使用してください。/EVENT , /PREDEFINED または/USER 以外の修飾子を使用する場合は、アドレス式は指定できません。

---

### 修飾子

/ACTIVATING

前回の SET BREAK/ACTIVATING コマンドで設定されたブレークポイントを無効にします。

/ALL

省略時の設定では、すべてのユーザ定義ブレークポイントを無効にします。/PREDEFINED といっしょに使用すると、定義済みブレークポイントはすべて無効になりますが、ユーザ定義ブレークポイントは無効になりません。すべてのブレークポイントを無効にするには/ALL/USER/PREDEFINED を使用します。

/BRANCH

前回の SET BREAK/BRANCH コマンドで設定されたブレークポイントを無効にします。

/CALL

前回の SET BREAK/CALL コマンドで設定されたブレークポイントを無効にします。

/EVENT=event-name

前回の SET BREAK/EVENT=event-name コマンドで設定されたブレークポイントを無効にします。イベント名 (および必要であればアドレス式) は SET BREAK/EVENT コマンドで指定したとおりに指定してください。

現在のイベント機能とそれに対応するイベント名を表示するには、SHOW EVENT\_FACILITY コマンドを使用します。

/EXCEPTION

前回の SET BREAK/EXCEPTION コマンドで設定されたブレークポイントを無効にします。

/HANDLER

前回の SET BREAK/HANDLER コマンドによって設定されたブレークポイントを無効にします。

/INSTRUCTION

前回の SET BREAK/INSTRUCTION コマンドで設定されたブレークポイントを無効にします。

/LINE

前回の SET BREAK/LINE コマンドで設定されたブレークポイントを無効にします。

/PREDEFINED

ユーザ定義ブレークポイントには全く影響を及ぼさずに、指定の定義済みブレークポイントを無効にします。/ALL といっしょに使用すると、定義済みブレークポイントがすべて無効になります。

/SYSEMULATE

(Alpha のみ) 前に実行した SET BREAK/SYSEMULATE コマンドによって設定されたブレークポイントを無効にします。

/TERMINATING

前回の SET BREAK/TERMINATING コマンドで設定されたブレークポイントを無効にします。

/UNALIGNED\_DATA

(Alpha のみ) 前回の SET BREAK/UNALIGNED\_DATA コマンドで設定されたブレークポイントを無効にします。

/USER

ユーザ定義ブレークポイントを無効にします。すべてのユーザ定義ブレークポイントを無効にするには、/ALL 修飾子を使用します。

---

## 説明

ユーザ定義ブレークポイントは SET BREAK コマンドで設定すると有効になります。定義済みブレークポイントは省略時の状態で有効になります。1 つまたは複数のブレークポイントを無効にするには DEACTIVATE BREAK コマンドを使用します。

ブレークポイントを無効にすると、デバッガはプログラムの実行中そのブレークポイントを無視します。無効にされたブレークポイントを有効にするには、ACTIVATE BREAK コマンドを使用します。ユーザ定義ブレークポイントと定義済みブレークポイントは別々に有効にしたり、無効にしたりすることができます。ブレークポイントを有効にしたり無効にしたりすることにより、プログラムの実行や再実行のときに、ブレークポイントを取り消して再設定する手間をかけずに、ブレークポイントを使用したり使用しなかったりすることができます。省略時の設定で RERUN コマンドを実行すると、すべてのブレークポイントの現在の状態 (有効か無効か) が保存されます。

ブレークポイントが無効になっているかどうかを確認するには、SHOW BREAK コマンドを使用します。

#### 関連コマンド

```
CANCEL ALL
RERUN
(SET,SHOW,CANCEL,ACTIVATE) BREAK
(SET,SHOW) EVENT_FACILITY
```

---

#### 例

1. DBG> DEACTIVATE BREAK MAIN\LOOP+10

このコマンドはアドレス式 MAIN\LOOP+10 で設定されたユーザ定義ブレークポイントを無効にします。

2. DBG> DEACTIVATE BREAK/ALL

このコマンドはすべてのユーザ定義ブレークポイントを無効にします。

---

## DEACTIVATE TRACE

以前に設定したあと無効にしたトレースポイントを有効にします。

---

### フォーマット

```
DEACTIVATE TRACE [address-expression[, ... ]]
```

---

### パラメータ

*address-expression*

有効にするトレースポイントを指定します。ワイルドカード文字のアスタリスク(\*)は使用できません。代わりに/ALL 修飾子を使用してください。/EVENT , /PREDEFINED , /USER 以外の修飾子を使用する場合は、アドレス式は指定できません。

---

### 修飾子

/ACTIVATING

前回の SET TRACE/ACTIVATING コマンドで設定されたトレースポイントを有効にします。

/ALL

省略時の設定では、すべてのユーザ定義トレースポイントを有効にします。/PREDEFINED といっしょに使用すると、定義済みトレースポイントはすべて有効になりますが、ユーザ定義トレースポイントは有効になりません。すべてのトレースポイントを有効にするには/ALL/USER/PREDEFINED を使用します。

/BRANCH

前回の SET TRACE/BRANCH コマンドで設定されたトレースポイントを有効にします。

/CALL

前回の SET TRACE/CALL コマンドで設定されたトレースポイントを有効にします。

/EVENT=*event-name*

前回の SET TRACE/EVENT=*event-name* コマンドで設定されたトレースポイントを有効にします。イベント名 (および必要であればアドレス式) は SET TRACE/EVENT コマンドで指定したとおりに指定してください。

現在のイベント機能とそれに対応するイベント名を表示するには、SHOW EVENT\_FACILITY コマンドを使用します。

/EXCEPTION

前回の SET TRACE/EXCEPTION コマンドで設定されたトレースポイントを有効にします。

/INSTRUCTION

前回の SET TRACE/INSTRUCTION コマンドで設定されたトレースポイントを有効にします。

/LINE

前回の SET TRACE/LINE コマンドで設定されたトレースポイントを有効にします。

/PREDEFINED

ユーザ定義トレースポイントには全く影響を及ぼさずに指定の定義済みトレースポイントを有効にします。/ALL といっしょに使用すると、定義済みトレースポイントがすべて有効になります。

/TERMINATING

前回の SET TRACE/TERMINATING コマンドで設定されたトレースポイントを有効にします。

/USER

定義済みトレースポイントには全く影響を及ぼさずに指定のユーザ定義トレースポイントを有効にします。/ALL といっしょに使用すると、ユーザ定義トレースポイントがすべて有効になります。/PREDEFINED を指定した場合を除き、/USER 修飾子が省略時の設定です。

---

## 説明

ユーザ定義トレースポイントは SET TRACE コマンドで設定すると有効になります。定義済みトレースポイントは省略時の設定で有効になります。DEACTIVATE TRACE を使用して無効にした 1 つまたは複数のトレースポイントを有効にするには ACTIVATE TRACE コマンドを使用します。

トレースポイントを有効にしたり無効にしたりすることにより、プログラムの実行や再実行のときに、トレースポイントを取り消して再設定する手間をかけずに、トレースポイントを使用したり使用しなかったりすることができます。省略時の設定で RERUN コマンドを実行すると、すべてのトレースポイントの現在の状態 (有効か無効か) が保存されます。

有効や無効にするのはユーザ定義トレースポイントと定義済みトレースポイントのどちらか一方でも両方でもかまいません。トレースポイントが有効になっているかを確認するには、SHOW TRACE コマンドを使用してください。

## DEACTIVATE TRACE

### 関連コマンド

CANCEL ALL  
RERUN  
(SET,SHOW) EVENT\_FACILITY  
(SET,SHOW,CANCEL,ACTIVATE) TRACE

---

### 例

1. DBG> DEACTIVATE TRACE MAIN\LOOP+10

このコマンドは MAIN\LOOP+10 の位置にあるユーザ定義トレースポイントを有効にします。

2. DBG> DEACTIVATE TRACE/ALL

このコマンドはユーザ定義トレースポイントをすべて有効にします。



---

# DEACTIVATE WATCH

以前に設定したあと無効にしたウォッチポイントを有効にします。

---

## フォーマット

DEACTIVATE WATCH *[address-expression[, ... ]]*

---

## パラメータ

address-expression

有効にするウォッチポイントを指定します。高級言語を使用している場合、これは通常、変数の名前になります。ワイルドカード文字のアスタリスク(\*)は使用できません。代わりに/ALL 修飾子を使用してください。/ALL を指定する場合は、アドレス式は指定できません。

---

## 修飾子

/ALL

すべてのウォッチポイントを有効にします。

---

## 説明

ウォッチポイントは SET WATCH コマンドで設定すると有効になります。DEACTIVATE WATCH で無効にした 1 つまたは複数のウォッチポイントを有効にするには ACTIVATE WATCH コマンドを使用します。

ウォッチポイントを有効にしたり無効にしたりすることにより、プログラムの実行や再実行のときに、ウォッチポイントを取り消して再設定する手間をかけずに、ウォッチポイントを使用したり使用しなかったりすることができます。

省略時の設定では、RERUN コマンドを実行すると、すべての静的ウォッチポイントの現在の状態(有効か無効か)が保存されます。特定の非静的ウォッチポイントは、(実行を再開する)メイン・プログラム・ユニットを基準にした、ウォッチの対象になっている変数の有効範囲によって保存されることもあり、保存されないこともあります。

ウォッチポイントが有効になっているかを確かめるには、SHOW WATCH コマンドを使用します。

## DEACTIVATE WATCH

### 関連コマンド

CANCEL ALL

RERUN

(SET,SHOW,CANCEL,ACTIVATE) WATCH

---

### 例

1. DBG> DEACTIVATE WATCH SUB2\TOTAL

このコマンドはモジュール SUB2 の中の TOTAL 変数でのウォッチポイントを有効にします。

2. DBG> DEACTIVATE WATCH/ALL

このコマンドは設定したあと無効にしたすべてのウォッチポイントを有効にします。

---

## DECLARE

コマンド・プロシージャの中で仮パラメータを宣言します。これにより、@ (実行プロシージャ) コマンドの入力時に実パラメータを渡すことができます。

---

### フォーマット

```
DECLARE p-name:p-kind [p-name:p-kind [, . . . ]]
```

---

### パラメータ

*p-name*

コマンド・プロシージャの中で宣言される仮パラメータ (シンボル) を指定します。

(連続する 2 つのコンマまたはコマンドの最後のコンマで表される) 空パラメータは指定できません。

*p-kind*

仮パラメータのパラメータ種別を指定します。有効なキー・ワードは次のとおりです。

ADDRESS	実パラメータをアドレス式として解釈することを指定します。DEFINE /ADDRESS <i>p-name</i> = <i>actual-parameter</i> と同じ働きです。
COMMAND	実パラメータをコマンドとして解釈することを指定します。DEFINE /COMMAND <i>p-name</i> = <i>actual-parameter</i> コマンドと同じ働きです。
VALUE	実パラメータを現在の言語での値式として解釈することを指定します。DEFINE/VALUE <i>p-name</i> = <i>actual-parameter</i> と同じ働きです。

---

### 説明

DECLARE コマンドはコマンド・プロシージャの中でだけ有効です。

DECLARE コマンドは、@ (実行プロシージャ) コマンドに続くコマンド行に指定された 1 つまたは複数の実パラメータを、コマンド・プロシージャの中で宣言された仮パラメータ (シンボル) にバインドします。

DECLARE コマンドで指定される各 *p-name:p-kind* の組は 1 つの仮パラメータを 1 つの実パラメータにバインドします。仮パラメータはデバッガがパラメータ宣言を処理する順番で実パラメータにバインドされます。1 つの DECLARE コマンドに複数の仮パラメータを指定すると、左端の仮パラメータが最初の実パラメータにバインドされ、次の仮パラメータが 2 番目の実パラメータにバインドされるというようになります。DECLARE コマンドをループで使用すると、仮パラメータはループの 1 回目の

## DECLARE

最初の実パラメータにバインドされます。そして同じ仮パラメータが 2 回目の 2 番目の実パラメータにバインドされるというようになります。

各パラメータ宣言は DEFINE コマンドと同様に動作します。指定されたパラメータ種別に従って、アドレス式、コマンドまたは現在の言語での値式に仮パラメータを対応づけるのです。仮パラメータそのものは DEFINE コマンドで受け入れられるものと一致しており、実際に DELETE コマンドでシンボル・テーブルから削除することができます。

%PARCNT 組み込みシンボルはコマンド・プロシージャの中でだけ使用できます。これを使用すれば、可変数のパラメータをコマンド・プロシージャに渡すことができます。%PARCNT の値はコマンド・プロシージャに渡される実パラメータの数です。

### 関連コマンド

@ (実行プロシージャ)  
DEFINE  
DELETE

---

## 例

```
1. ! ***** コマンド・プロシージャ EXAM.COM *****
   SET OUTPUT VERIFY
   DECLARE K:ADDRESS
   EXAMINE K

   DBG> @EXAM ARR4
   %DEBUG-I-VERIFYIC, entering command procedure EXAM
   DECLARE K:ADDRESS
   EXAMINE K
   PROG 8\ARR4
   (1):      18
   (2):       1
   (3):       0
   (4):       1
   %DEBUG-I-VERIFYIC, exiting command procedure EXAM
   DBG>
```

この例では、DECLARE K:ADDRESS コマンドはコマンド・プロシージャ EXAM.COM の内部の仮パラメータ K を宣言しています。EXAM.COM を実行すると、EXAM.COM に渡される実パラメータはアドレス式として解釈され、EXAMINE K コマンドはそのアドレス式の値を表示します。SET OUTPUT VERIFY コマンドを実行すると、コマンドをデバッガが読んだときに、それらのコマンドがエコーバックされます。

デバッガ・プロンプトが表示された状態で、@EXAM ARR4 コマンドは EXAM.COM を実行し、実パラメータ ARR4 を渡します。EXAM.COM の中では、ARR4 はアドレス式 (この場合には配列変数) として解釈されます。

2. ! \*\*\*\*\* デバッガ・コマンド・プロシージャ EXAM\_GO.COM \*\*\*\*\*  
 DECLARE L:ADDRESS, M:COMMAND  
 EXAMINE L; M  
  
 DBG> @EXAM\_GO X "@DUMP"

この例では、コマンド・プロシージャ EXAM\_GO.COM はアドレス式(L)とコマンド文字列(M)の2つのパラメータを受け付けます。そしてアドレス式がチェックされ、コマンドが実行されます。

デバッガ・プロンプトが表示された状態で、@EXAM\_GO X "@DUMP" コマンドは EXAM\_GO.COM を実行し、アドレス式 X とコマンド文字列の @DUMP を渡します。

3. ! \*\*\*\*\* デバッガ・コマンド・プロシージャ VAR.DBG \*\*\*\*\*  
 SET OUTPUT VERIFY  
 FOR I = 1 TO %PARCNT DO (DECLARE X:VALUE; EVALUATE X)  
 DBG> @VAR.DBG 12,37,45  
 %DEBUG-I-VERIFYIC, entering command procedure VAR.DBG  
 FOR I = 1 TO %PARCNT DO (DECLARE X:VALUE; EVALUATE X)  
 12  
 37  
 45  
 %DEBUG-I-VERIFYIC, exiting command procedure VAR.DBG  
 DBG>

この例では、コマンド・プロシージャ VAR.DBG が受け取るパラメータの個数は固定されていません。パラメータの個数は組み込みシンボル %PARCENT に格納されます。

デバッガ・プロンプトが表示された状態で、@VAR.DBG コマンドは VAR.DBG を実行し、実パラメータ 12, 37, 45 を渡します。したがって、%PARCNT の値は 3 になり、FOR ループが 3 回繰り返されます。FOR ループにより、DECLARE コマンドは 3 つの実パラメータ (12 から始まる) の各々を X の新しい宣言にバインドするようになります。各実パラメータは現在の言語での値式として解釈され、EVALUATE X コマンドがその値を表示します。

---

## DEFINE

アドレス式，コマンドまたは値にシンボリック名を割り当てます。

---

### フォーマット

```
DEFINE symbol-name=parameter [symbol-name=parameter [, ... ]]
```

---

### パラメータ

*symbol-name*

アドレス，コマンドまたは値に割り当てるシンボリック名を指定します。シンボリック名は英字とアンダスコアで構成できます。デバッガは英小文字を大文字に変換します。先頭文字は数字であってはなりません。シンボリック名は 31 文字を超えてはなりません。

*parameter*

指定された修飾子によって異なります。

---

### 修飾子

/ADDRESS

省略時の設定。定義したシンボルがアドレス式の短縮形であることを指定します。この場合，*parameter*はアドレス式になります。

/COMMAND

定義したシンボルを新しいデバッガ・コマンドとして扱うことを指定します。この場合，*parameter*は引用符で囲まれた文字列になります。この修飾子は，単純な場合には基本的に次の DCL コマンドと同じ機能を提供します。

```
$ symbol := string
```

複雑なコマンドを定義するには，仮パラメータ付きのコマンド・プロシージャの使用が必要となることがあります。コマンド・プロシージャに対するパラメータの宣言についての詳しい説明は，DECLARE コマンドを参照してください。

/LOCAL

定義されたコマンド・プロシージャ内でのみ有効になるように定義を指定します。定義されたシンボルはデバッガ・コマンド・レベルでは可視状態ではありません。省略時の設定では，コマンド・プロシージャの中で定義されたシンボルはそのプロシージャの外では可視状態になります。

## /VALUE

定義されたシンボルが値の短縮形であることを指定します。この場合、*parameter*は現在の言語での言語式になります。

---

## 説明

DEFINE/ADDRESS コマンドを使用すれば、プログラム内のアドレス式にシンボリック名を割り当てられます。たとえば、シンボル以外のプログラム記憶位置に対してか、長いパス名接頭識別子を持つシンボルのプログラム記憶位置に対してシンボルを定義できます。シンボルを定義したら、そのプログラム記憶位置を新しく定義したシンボルで参照できるようになります。/ADDRESS 修飾子が省略時の設定です。

DEFINE/COMMAND コマンドを使用すれば、デバッグ・コマンド・レベルまたはコマンド・プロシージャから、デバッグ・コマンドの短縮形を定義するか、新しいコマンドを定義することができます。

DEFINE/VALUE コマンドを使用すれば、シンボリック名を値 (または言語式を評価する結果) に割り当てられます。

シンボル定義をコマンド・プロシージャ内に限定するには、/LOCAL 修飾子を使用します。省略時の設定では、定義されたシンボルはグローバル (コマンド・プロシージャの外で可視状態) になります。

同じ修飾子を使用して複数の DEFINE コマンドを入力する場合は、まず SET DEFINE コマンドを使用して新しい省略時の修飾子を設定できます。たとえば、SET DEFINE COMMAND を使用すれば、DEFINE コマンドが DEFINE /COMMAND と同じように動作するようになります。そのあと、DEFINE コマンドで同じ修飾子を使用する必要はありません。他の修飾子を指定することにより、1 つの DEFINE コマンドの継続中に現在の省略時の修飾子を上書きできます。

シンボル変換では、デバッグはまずデバッグ・セッションの間に定義されたシンボルを検索します。したがって、プログラムにすでに存在しているシンボルが定義されると、パス名の接頭識別子が指定された場合を除き、デバッグはそのシンボルをもともと定義されていた定義に従って変換します。

シンボルを再定義すると、DEFINE コマンドで異なる修飾子を使用していた場合でも、もとの定義は取り消されます。

DEFINE/ADDRESS コマンドと DEFINE/VALUE コマンドで作成された定義は、それらが作成されたコンテキストの中のイメージが現在のイメージになっているときだけ使用できます。新しい現在のイメージを設定するために SET IMAGE コマンドを使用すると、これらの定義は一時的に使用できない状態になります。ただし、DEFINE/COMMAND コマンドと DEFINE/KEY コマンドで作成された定義はいつでもすべてのイメージに使用できます。

## DEFINE

シンボルの同値を求めるには SHOW SYMBOL/DEFINED コマンドを使用します。

シンボル定義を取り消すには DELETE コマンドを使用します。

関連コマンド

```
DECLARE
DELETE
SET IMAGE
SHOW DEFINE
SHOW SYMBOL/DEFINED
```

---

### 例

1. DBG> DEFINE CHK=MAIN\LOOP+10

このコマンドはシンボル CHK をアドレス MAIN\LOOP+10 に割り当てます。

2. DBG> DEFINE/VALUE COUNTER=0  
DBG> SET TRACE/SILENT R DO (DEFINE/VALUE COUNTER = COUNTER+1)

この例では、DEFINE/VALUE コマンドは値の 0 を COUNTER シンボルに割り当てます。SET TRACE コマンドを使用すると、デバッガはアドレス R が検出されるたびにシンボル COUNTER の値を 1 だけ増分します。つまり、この例では R への呼び出し回数を数えているのです。

3. DBG> DEFINE/COMMAND BRE = "SET BREAK"

このコマンドはシンボル BRE をデバッガ・コマンドの SET BREAK に割り当てます。



---

## DEFINE/KEY

ファンクション・キーに文字列を割り当てます。

---

### 注意

---

このコマンドは、デバッガへの HP DECwindows Motif for OpenVMS ユーザ・インタフェースでは使用できません。

---



---

## フォーマット

DEFINE/KEY *key-name* "*equivalence-string*"

---

## パラメータ

*key-name*

文字列を割り当てるファンクション・キーを指定します。有効なキー名は次のとおりです。

キー名	LK201 キーボード	VT100 型	VT52 型
PF1	PF1	PF1	Blue
PF2	PF2	PF2	Red
PF3	PF3	PF3	Black
PF4	PF4	PF4	
KP0-KP9	キーパッドの 0 ~ 9	キーパッドの 0 ~ 9	キーパッドの 0 ~ 9
PERIOD	キーパッドのピリオド(.)	キーパッドのピリオド(.)	
COMMA	キーパッドのコンマ(,)	キーパッドのコンマ(,)	
MINUS	キーパッドのマイナス(-)	キーパッドのマイナス(-)	
ENTER	Enter	ENTER	ENTER
E1	Find		
E2	Insert Here		
E3	Remove		
E4	Select		
E5	Prev Screen		
E6	Next Screen		
HELP	Help		
DO	Do		
F6-F20	F6-F20		

---

LK201 キーボードでは、次のようになります。

- F1 から F5 のキーや矢印キー (E7 から E10) は定義できません。
- DCL コマンドの SET TERMINAL/NOLINE\_EDITING を最初に入力していた場合にだけ F6 から F14 のキーは定義できます。入力していた場合には、左向き矢印キーと右向き矢印キー (E8 と E9) の行編集機能は使用できません。

equivalence-string

指定のキーを押したときに処理する文字列を指定します。通常、これは 1 つまたは複数のデバッグ・コマンドになります。文字列にスペースまたは英数字以外の文字 (たとえば、2 つのコマンドを分けるセミコロン) が含まれている場合には、文字列を二重引用符(")で囲んでください。

---

## 修飾子

/ECHO (省略時の設定)

/NOECHO

キーが押されたあとにコマンド行を表示するかどうかを制御します。

/NOTERMINATE を指定する場合は、/NOECHO は使用できません。

/IF\_STATE=(state-name[, . . . ])

/NOIF\_STATE (省略時の設定)

キー定義が適用される 1 つまたは複数の状態を指定します。/IF\_STATE 修飾子は指定された状態にキー定義を割り当てます。DEFAULT や GOLD などの定義済み状態またはユーザ定義状態を指定できます。状態名は適切な英数字文字列にすることができます。/NOIF\_STATE 修飾子を使用すると、キー定義が現在の状態に割り当てられません。

/LOCK\_STATE

/NOLOCK\_STATE (省略時の設定)

指定のキーを押したあと、/SET\_STATE で設定された状態がどれだけの間有効であり続けるかを制御します。/LOCK\_STATE 修飾子を指定すると、状態は (たとえば SET KEY/STATE コマンドにより) 明示的に変更されるまで有効のままです。/NOLOCK\_STATE 修飾子を指定すると、次の終了文字が入力されるまで、または次の定義済みファンクション・キーが押されるまでの間だけしか状態は有効ではありません。

/LOG (省略時の設定)

/NOLOG

キー定義が正常に作成されていることを示すメッセージを表示するかどうかを制御します。/LOG 修飾子を指定すると、メッセージが表示されます。/NOLOG 修飾子を指定すると、メッセージは表示されません。

/SET\_STATE=state-name

/NOSET\_STATE (省略時の設定)

キーを押して現在のキー状態を変化させるかどうかを制御します。/SET\_STATE 修飾子を指定すると、キーを押したときに現在の状態が指定の状態に変化します。

/NOSET\_STATE 修飾子を指定すると、現在の状態がそのまま有効になります。

/TERMINATE

/NOTERMINATE (省略時の設定)

キーを押したときに指定の文字列を終了 (処理) するかどうかを制御します。

/TERMINATE 修飾子を指定すると、キーを押したときに文字列が終了します。

/NOTERMINATE 修飾子を指定すると、Return キーを押して文字列を終了する前に他のキーを押すことができます。

---

## 説明

このコマンドを使用する前にキーパッド・モードが使用できる状態 (SET MODE KEYPAD) に設定しておかなければなりません。省略時の設定では、キーパッド・モードは使用可能です。

DEFINE/KEY コマンドを使用すれば、ファンクション・キーに文字列を割り当て、そのキーにバインドされている定義済みの機能を上書きできます。そのあと、そのキーを押すと、デバッグは現在対応づけられている文字列をコマンド行に入力します。DEFINE/KEY コマンドは DCL コマンドの DEFINE/KEY に似ています。

定義済みのキー機能のリストについては、「CI のキーパッド定義」オンライン・ヘルプ・トピックを参照してください。

VT52 シリーズと VT100 シリーズの端末では、ユーザが使用できるファンクション・キーは数値キーパッド・キーです。これより新しい端末やワークステーションには LK201 のキーボードが備っています。LK201 のキーボードで使用できるファンクション・キーとしては、数値キーパッド・キー全部、編集キーパッドの矢印以外のキー (Find, Insert Here など)、キーボードの上部にある F6 から F20 のキーなどがあります。

キー定義は、そのキーが再定義されるか、そのキーに対して DELETE/KEY コマンドが入力されるか、デバッグが終了されるまで有効です。デバッグ初期化ファイルなどのコマンド・プロシージャにはキー定義を含めることができます。

/IF\_STATE 修飾子を使用すれば、端末で使用できるキー定義の数を増加できます。各定義が異なる状態に対応づけられているかぎり、同じキーにいくつでも定義を割り当てられます。

省略時の状態では、現在のキー状態は "DEFAULT" 状態です。現在の状態は SET KEY/STATE コマンドを使用するか、状態を変更するキー (DEFINE/KEY/LOCK\_STATE/SET\_STATE で定義されたキー) を押すことにより変更できます。

## 関連コマンド

DELETE/KEY

(SET,SHOW) KEY

## 例

1. 

```
DBG> SET KEY/STATE=GOLD
%DEBUG-I-SETKEY, keypad state has been set to GOLD
DBG> DEFINE/KEY/TERMINATE KP9 "SET RADIX/OVERRIDE HEX"
%DEBUG-I-DEFKEY, GOLD key KP9 has been defined
```

この例では、SET KEY コマンドは現在のキー状態として GOLD を設定しています。/DEFINE/KEY コマンドは SET RADIX/OVERRIDE HEX コマンドを現在の状態 (GOLD) のキーパッド・キー 9 (KP9) に割り当てます。このコマンドは KP9 キーが押されると処理されます。

2. 

```
DBG> DEFINE/KEY/IF_STATE=BLUE KP9 "SET BREAK %LINE "
%DEBUG-I-DEFKEY, BLUE key KP9 has been defined
```

このコマンドは、未完成のコマンド文字列 "SET BREAK%LINE" を BLUE 状態のキーパッド・キー 9 に割り当てます。BLUE-KP9 を押したあと、行番号を入力して Return キーを押せば、SET BREAK コマンドを終了して処理することができます。

3. 

```
DBG> SET KEY/STATE=DEFAULT
%DEBUG-I-SETKEY, keypad state has been set to DEFAULT
DBG> DEFINE/KEY/SET_STATE=RED/LOCK_STATE F12 ""
%DEBUG-I-DEFKEY, DEFAULT key F12 has been defined
```

この例では、SET KEY コマンドは現在の状態として DEFAULT を設定します。DEFINE/KEY コマンドは (LK201 キーボード上の) F12 キーを状態キーにします。DEFAULT 状態にあるときに F12 キーを押すと、現在の状態が RED になります。キー定義は終了されず、他に何の作用も持ちません (空文字列が F12 に割り当てられたことになります)。F12 キーを押したあと、RED 状態に対応づけられている定義を持つキーを押せば、"RED" コマンドを入力できます。

# DEFINE/PROCESS\_SET

シンボリック名をプロセス指定のリストに割り当てます。

## フォーマット

DEFINE/PROCESS\_SET *process-set-name* =*process-spec*[, ... ]

## パラメータ

*process-set-name*

プロセス指定のリストに割り当てるシンボリック名を指定します。シンボリック名は英数字とアンダスコアで構成することができます。デバッガは英小文字を英大文字に変換します。先頭文字は数字でなければなりません。シンボリック名は 31 文字を超えてはなりません。

*process-spec*

現在デバッガの制御下にあるプロセスを指定します。次のいずれかの形式で指定します。

[%PROCESS\_NAME] *process-name*

スペースや小文字を含まないプロセス名。プロセス名にはワイルドカード文字(\*)を含めることができる。

[%PROCESS\_NAME] "*process-name*"

スペースまたは小文字を含むプロセス名。二重引用符(")の代わりに、一重引用符(') 使用することもできる。

%PROCESS\_PID *process\_id*

プロセス識別子 (PID, 16 進数)。

[%PROCESS\_NUMBER] *process-number*  
(または%PROC *process-number*)

デバッガの制御下に入ったときにプロセスに割り当てられた番号。新しい番号は、1 から順番に各プロセスに割り当てられる。EXIT コマンドまたは QUIT コマンドによってプロセスが終了した場合、そのデバッグ・セッション中にその番号が再割り当てされることがある。プロセス番号は SHOW PROCESS コマンドの実行で表示される。プロセスは、組み込みシンボル%PREVIOUS\_PROCESSおよび%NEXT\_PROCESSによってインデックスづけできるように、循環リスト内に順序づけされる。

*process-set-name*

DEFINE/PROCESS\_SET コマンドで定義された、プロセスのグループを表すシンボル。

%NEXT\_PROCESS

デバッガの循環プロセス・リスト中で可視プロセスの次のプロセス。

%PREVIOUS\_PROCESS

デバッガの循環プロセス・リスト中で可視プロセスの前のプロセス。

%VISIBLE\_PROCESS

シンボル、レジスタ値、ルーチン呼び出し、ブレークポイントなどの検索時に現在のコンテキストになっているスタック、レジスタ・セット、およびイメージを持つプロセス。

プロセスを指定しないと、シンボリック名は作成されますが、プロセス・エントリは含まれません。

---

## 説明

DEFINE/PROCESS\_SET コマンドはプロセス指定のリストにシンボルを割り当てます。割り当てたシンボルは、プロセス指定のリストが許されているコマンドであればどの中でも使用できます。

DEFINE/PROCESS\_SET コマンドは指定されたプロセスの有無を調べません。このため、まだ存在していないプロセスでも指定できます。

DEFINE/PROCESS\_SET コマンドで定義されたシンボルを表示するには、SHOW SYMBOL/DEFINED コマンドを使用します。DEFINE/PROCESS\_SET コマンドで定義されたシンボルを削除するには、DELETE コマンドを使用します。

関連コマンド

DELETE  
(SET,SHOW) DEFINE  
SHOW SYMBOL/DEFINED

---

## 例

```
1. all> DEFINE/PROCESS_SET SERVERS=FILE_SERVER,NETWORK_SERVER
   all> SHOW PROCESS SERVERS
      Number Name      Hold State   Current PC
   *    1 FILE_SERVER      step   FS_PROG\%LINE 37
      2 NETWORK_SERVER    break  NET_PROG\%LINE 24
   all>
```

この DEFINE/PROCESS\_SET コマンドはシンボリック名 SERVERS を FILE\_SERVER と NETWORK\_SERVER で構成されるプロセス・グループに割り当てます。SHOW PROCESS SERVERS コマンドは SERVERS グループを構成するプロセスについての情報を表示します。

2. all> DEFINE/PROCESS\_SET G1=%PROCESS\_NUMBER 1,%VISIBLE\_PROCESS  
 all> SHOW SYMBOL/DEFINED G1  
 defined G1  
     bound to: "%PROCESS\_NUMBER 1, %VISIBLE\_PROCESS"  
     was defined /process\_set  
 all> DELETE G1

この DEFINE/PROCESS\_SET コマンドはシンボリック名 G1 をプロセス 1 と可視プロセス (プロセス 3) で構成されるプロセス・グループに割り当てます。SHOW SYMBOL/DEFINED G1 コマンドは定義されたシンボル G1 を表示します。DELETE G1 コマンドは DEFINE シンボル・テーブルからそのシンボルを削除します。

3. all> DEFINE/PROCESS\_SET A = B,C,D  
 all> DEFINE/PROCESS\_SET B = E,F,G  
 all> DEFINE/PROCESS\_SET E = I,J,A  
 %DEBUG-E-NORECSYM, recursive PROCESS\_SET symbol definition  
     encountered at or near "A"

この一連の DEFINE/PROCESS\_SET コマンドは DEFINE/PROCESS\_SET コマンドの無効な用法と有効な用法を示しています。

---

## DELETE

DEFINE コマンドで設定されたシンボル定義を削除します。

---

### フォーマット

```
DELETE  [symbol-name[, ... ]]
```

---

### パラメータ

*symbol-name*

その定義を DEFINE シンボル・テーブルから削除するシンボルを指定します。ワイルドカード文字のアスタリスク(\*)は使用できません。代わりに/ALL 修飾子を使用してください。/ALL を使用する場合は、シンボル名は指定できません。/LOCAL 修飾子を使用する場合、指定するシンボルは DEFINE/LOCAL コマンドで事前に定義されていたものでなければなりません。/LOCAL を指定しない場合には、指定されるシンボルは/LOCAL を指定しないで DEFINE コマンドで事前に定義されていたものでなければなりません。

---

### 修飾子

/ALL

すべてのグローバル DEFINE 定義を削除します。/ALL/LOCAL を使用すると、現在のコマンド・プロシージャに対応づけられたローカルな DEFINE 定義はすべて削除されます(グローバル DEFINE 定義は削除されません)。

/LOCAL

指定されたシンボルの(ローカル)定義を現在のコマンド・プロシージャから削除します。指定するシンボルは DEFINE/LOCAL コマンドで事前に定義されていたものでなければなりません。

---

### 説明

DELETE コマンドはグローバルな DEFINE シンボルまたはローカルな DEFINE シンボルを削除します。グローバルな DEFINE シンボルは/LOCAL 修飾子を指定せずに DEFINE コマンドにより定義されたシンボルです。ローカルな DEFINE シンボルは DEFINE/LOCAL コマンドによりデバグ・コマンド・プロシージャで定義されたシンボルです。そのため、その定義はそのコマンド・プロシージャ内に限定されます。



## 関連コマンド

DECLARE  
DEFINE  
SHOW DEFINE  
SHOW SYMBOL/DEFINED

---

例

1. DBG> DEFINE X = INARR, Y = OUTARR  
DBG> DELETE X,Y

この例では、DEFINE コマンドは X と Y をそれぞれ INARR と OUTARR に対応するグローバル・シンボルとして定義します。DELETE コマンドはこの 2 つのシンボル定義をグローバル・シンボル・テーブルから削除します。

2. DBG> DELETE/ALL/LOCAL

このコマンドはすべてのローカル・シンボル定義を現在のコマンド・プロシージャから削除します。

## DELETE/KEY

DELETE/KEY コマンドで設定されたキー定義および省略時にデバッガによって設定されたキー定義を削除します。

### 注意

このコマンドは、デバッガへの HP DECwindows Motif for OpenVMS ユーザ・インタフェースでは使用できません。

## フォーマット

DELETE/KEY *[key-name]*

## パラメータ

key-name

キー定義を削除するキーを指定します。ワイルドカード文字のアスタリスク(\*)は使用できません。代わりに/ALL 修飾子を使用してください。/ALL を指定する場合は、キー名は指定できません。有効なキー名は次のとおりです。

キー名	LK201 キーボード	VT100 型	VT52 型
PF1	PF1	PF1	Blue
PF2	PF2	PF2	Red
PF3	PF3	PF3	Black
PF4	PF4	PF4	
KP0-KP9	Keypad 0-9	Keypad 0-9	Keypad 0-9
KP0-KP9	キーパッドの 0 ~ 9	キーパッドの 0 ~ 9	キーパッドの 0 ~ 9
PERIOD	キーパッドのピリオド(.)	キーパッドのピリオド(.)	
COMMA	キーパッドのコンマ(,)	キーパッドのコンマ(,)	
MINUS	キーパッドのマイナス(-)	キーパッドのマイナス(-)	
ENTER	Enter	ENTER	ENTER
E1	Find		
E2	Insert Here		
E3	Remove		
E4	Select		

キー名	LK201 キーボード	VT100 型	VT52 型
E5	Prev Screen		
E6	Next Screen		
HELP	Help		
DO	Do		
F6-F20	F6-F20		

## 修飾子

/ALL

指定された状態のキー定義をすべて削除します。状態を指定しないと、現在の状態のキー定義がすべて削除されます。1 つまたは複数の状態を指定するには、/STATE=*state-name*を使用します。

/LOG (省略時の設定)

/NOLOG

指定されたキー定義が削除されていることを示すメッセージを表示するかどうかを制御します。/LOG 修飾子 (これが省略時の設定) を指定すると、メッセージが表示されます。/NOLOG 修飾子を指定すると、メッセージは表示されません。

/STATE=(*state-name* [, . . . ])

/NOSTATE (省略時の設定)

キー定義を削除したい状態を選択します。/STATE 修飾子を指定すると、指定した状態に対するキー定義が削除されます。DEFAULT や GOLD などの定義済みキー状態またはユーザ定義状態を指定できます。状態名は適切な英数字文字列にすることができます。/NOSTATE 修飾子を指定すると、現在の状態に対するキー定義だけが削除されます。

省略時の設定では、現在のキーの状態は"DEFAULT"状態です。現在の状態は SET KEY/STATE コマンドを使用するか、状態を変更するキー (DEFINE/KEY/LOCK\_STATE/SET\_STATE で定義されたキー) を押すことにより変更できます。

## 説明

DELETE/KEY コマンドは DCL コマンドの DELETE/KEY と似ています。

このコマンドを使用する前にキーパッド・モードが使用可能な状態 (SET MODE KEYPAD) に設定されていなければなりません。省略時の設定では、キーパッド・モードは使用可能です。

### 関連コマンド

DEFINE/KEY  
(SET,SHOW) KEY

---

### 例

1. DBG> DELETE/KEY KP4  
%DEBUG-I-DELKEY, DEFAULT key KP4 has been deleted

このコマンドは前回 SET KEY コマンドで設定された状態 (省略時の設定では、これは DEFAULT 状態) で KP4 に対するキー定義を削除します。

2. DBG> DELETE/KEY/STATE=(BLUE,RED) COMMA  
%DEBUG-I-DELKEY, BLUE key COMMA has been deleted  
%DEBUG-I-DELKEY, RED key COMMA has been deleted

このコマンドは BLUE と RED の状態で COMMA キーに対するキー定義を削除します。

---

## DEPOSIT

プログラム変数の値を変更します。より通常は、アドレス式で示される記憶位置に新しい値を格納します。

---

### フォーマット

DEPOSIT *address-expression* = *language-expression*

---

### パラメータ

#### *address-expression*

言語式の値を格納する記憶位置を指定します。高級言語を使用する場合、これは通常、変数の名前になり、変数を一意に指定するためのパス名を含めることができます。典型的なアドレス式は、メモリ・アドレスやレジスタであったり、数字 (オフセット) やシンボルだけでなく 1 つまたは複数の演算子やオペランド、区切り文字などで構成されていたりします。レジスタのデバッガ・シンボルについての詳しい説明とアドレス式の中で使用できる演算子についての詳しい説明は、「組み込みシンボル」と「アドレス式」のヘルプ・トピックを参照してください。

集合体変数全体 (配列またはレコードなどの複合データ構造) は指定できません。個々の配列要素またはレコードの構成要素を指定するには、現在の言語の構文に従ってください。

#### *language-expression*

格納する値を指定します。現在の言語で有効な言語式であればどれでも指定できます。ほとんどの言語の場合、式には単純変数 (複合でない単一値) の名前をいれることはできますが、集合体変数 (配列やレコードなど) の名前は入れられません。式に異なるコンパイラ生成型が含まれている場合には、デバッガはその式を評価するために現在の言語の規則を使用します。

式が ASCII 文字列またはアセンブリ言語命令の場合には、その式を二重引用符 (") または一重引用符 (') で囲まなければなりません。文字列に二重引用符も一重引用符も含まれていない場合には、他の区切り文字を使用してその文字列を囲んでください。

文字列に含まれている文字数 (ASCII, バイト単位) がアドレス式で示されるプログラム記憶位置に収まる文字数より多い場合には、デバッガは右側から余分な文字を切り捨てます。文字列に含まれている文字数が少ない場合には、デバッガは文字列の右側に ASCII スペース文字を挿入することにより足りない文字を埋め込みます。

---

修飾子

/ASCIC

/AC

指定された長さの ASCII 文字列を指定された記憶位置に格納します。文字列は等号の右側に指定しなければなりません。格納される文字列の前には、文字列の長さを指定する 1 バイトのカウント・フィールドが付きます。

/ASCID

/AD

指定された記憶位置にある文字列ディスクリプタによって指定されるアドレスに ASCII 文字列を格納します。引用符で囲まれた文字列は等号の右側に指定しなければなりません。指定の記憶位置には文字列ディスクリプタが入らなければなりません。文字列の長さが一致しない場合には、文字列は右側で切り捨てられるか、右側にスペース文字を埋め込まれます。

/ASCII:n

指定された記憶位置に ASCII 文字列の  $n$  バイトを格納します。引用符で囲まれた文字列は等号の右側に指定しなければなりません。文字列の長さが  $n$  でない場合には、文字列は右側で切り捨てられるか、スペース文字を埋め込まれます。 $n$  を省略すると、指定された記憶位置のデータ項目の実際の長さが使用されます。

/ASCIW

/AW

指定された長さの ASCII 文字列を指定された記憶位置に格納します。引用符で囲まれた文字列は等号の右側に指定しなければなりません。格納される文字列の前には、その文字列の長さを指定する 2 バイトのカウント・フィールドが付きます。

/ASCIZ

/AZ

最後部に 0 の付く ASCII 文字列を指定された記憶位置に格納します。引用符で囲まれた文字列は等号の右側に指定しなければなりません。格納された文字列の最後部には文字列の終わりを示す 0 が 1 バイト分つきます。

/BYTE

指定された記憶位置に 1 バイトの整数を格納します。

/D\_FLOAT

等号の右側の式を D 浮動小数点型 (8 バイト長) に変換し、その結果を指定の記憶位置に格納します。

/DATE\_TIME

日付と時刻を表す文字列 (たとえば、21-DEC-1988 21:08:47.15) を日付と時刻の内部形式に変換し、その値 (8 バイト長) を指定された記憶位置に格納します。絶対日付と絶対時刻は次の形式で指定します。

[dd-mmm-yyyy[:]] [hh:mm:ss.cc]

/EXTENDED\_FLOAT  
/X\_FLOAT

(Alpha のみ) 等号の右側にある式を IEEE の X 浮動小数点型 (16 バイト長) に変換し、その結果を指定された記憶位置に格納します。

/FLOAT

VAX プロセッサでは、等号の右側にある式を F 浮動小数点型 (4 バイト長) に変換し、その結果を指定された記憶位置に格納します。

Alpha プロセッサでは、等号の右側にある式を IEEE の T 浮動小数点型 (8 バイト長) に変換し、その結果を指定された記憶位置に格納します。

/F\_FLOAT

(VAX のみ) 等号の右側の式を F 浮動小数点型 (4 バイト長) に変換し、その結果を指定の記憶位置に格納します。

/G\_FLOAT

等号の右側にある式を G 浮動小数点型 (8 バイト長) に変換し、その結果を指定された記憶位置に格納します。

/H\_FLOAT

(VAX のみ) 等号の右側にある式を H 浮動小数点型 (16 バイト長) に変換し、その結果を指定された記憶位置に格納します。

/INSTRUCTION

(VAX のみ) 指定された記憶位置に命令を格納します。等号の右側にある式はアセンブリ言語を表す文字列でなければなりません。

/LONG\_FLOAT  
/S\_FLOAT

(Alpha および I64 のみ) 等号の右側にある式を IEEE S 浮動小数点型 (単一精度, 4 バイト長) に変換し、その結果を指定された記憶位置に格納します。

/LONG\_LONG\_FLOAT  
/T\_FLOAT

(Alpha および I64 のみ) 等号の右側にある式を IEEE T 浮動小数点型 (倍精度, 8 バイト長) に変換し、その結果を指定の記憶位置に格納します。

/LONGWORD

指定された記憶位置にロングワード整数 (4 バイト長) を格納します。

/OCTAWORD

指定された記憶位置にオクタワード整数 (16 バイト長) を格納します。

/PACKED:n

等号の右側にある式をパック 10 進数表現に変換し、その結果として生じた値を指定された記憶位置に格納します。*n* の値は 10 進数の桁数です。各桁は 1 ニブル (4 ビット) を占めます。

**/QUADWORD**

指定された記憶位置にクォードワード整数 (8 バイト長) を格納します。

**/TASK**

タスキング (マルチスレッド) プログラムの場合に指定できます。指定された記憶位置にタスクの値 (タスク名または %TASK 3 などのタスク ID) を格納します。格納される値は有効なタスク値でなければなりません。

**/TYPE=(name)**

格納する式を *name* (プログラムで宣言された変数またはデータ型の名前でない限り) で示される型に変換し、指定された記憶位置に変換結果の値を格納します。これにより、ユーザ宣言型を指定できるようになります。型式は括弧で囲まなければなりません。

**/WCHAR\_T[:n]**

変換したマルチバイト・ファイル・コード・シーケンスの中で、最大 *n* ロングワード (*n* 文字) を指定の記憶位置に格納します。省略時の値は 1 ロングワードです。等号の右側に文字列を指定しなければなりません。

指定された文字列を変換する場合、デバッガはそのデバッガが実行されているプロセスのロケール・データベースを使用します。省略時の設定は C ロケールです。

**/WORD**

ワード整数 (2 バイト長) を指定の記憶位置に格納します。

---

**説明**

DEPOSIT コマンドを使用すれば、プログラムでアクセス可能なメモリ記憶位置またはレジスタの内容を変更できます。高級言語の場合、このコマンドは通常、変数 (整数、実数、文字列、配列、レコードなど) の値を変更する場合に使用されます。

DEPOSIT コマンドはほとんどのプログラミング言語での代入文と似ています。等号の右側に指定された式の値が等号の左側に指定された変数やその他の記憶位置に代入されます。Ada と Pascal の場合、コマンド構文では "=" の代わりに ":=" を使用できます。

デバッガはシンボリック・アドレス式 (プログラムで宣言されるシンボリック名) に対応するコンパイラ生成型を認識します。シンボリック・アドレス式には次の要素が含まれます。

- 変数名。DEPOSIT コマンドで変数を指定するときは、ソース・コードで使われるものと同じ構文を使用します。
- ルーチン名、ラベル、行番号。VAX システムではこれらは命令と対応づけられます。命令は基本的に文字列変数に格納するときと同じ方法を使用して格納できます。ただし、/INSTRUCTION 修飾子も使用するか、あるいはまず SET TYPE



INSTRUCTION コマンドまたは SET TYPE/OVERRIDE INSTRUCTION コマンドを入力する必要があります。

通常、DEPOSIT コマンドを入力すると、デバッガは次の動作を行います。

- 等号の左側に指定されたアドレス式を評価し、プログラム記憶位置を求めます。
- プログラム記憶位置にシンボリック名がある場合には、デバッガはその記憶位置をシンボルのコンパイラ生成型と対応づけます。記憶位置にシンボリック名がない場合 (したがって、対応するコンパイラ生成型もない場合) には、省略時の設定ではデバッガはその位置をロングワード整数型に対応づけます。これは省略時の状態では、4 バイトを超えない整数値をこれらの記憶位置に格納できることを意味します。
- 等号の右側に指定された言語式を現在の言語の構文と現在の基数で評価し、値を求めます。現在の言語は前回 SET LANGUAGE コマンドで設定された言語です。省略時の設定では、SET LANGUAGE コマンドを入力しなかった場合、現在の言語はメイン・プログラムを含むモジュールの言語になります。
- 言語式の値と型がアドレス式の型と一致しているかを調べます。アドレス式の型と互換性のない値を格納しようとする、デバッガは診断メッセージを発行します。互換性のある値の場合には、デバッガはその値をアドレス式で示される記憶位置に格納します。

言語の規則で許されている場合には、デバッガは格納操作時に型変換を行うことがあります。たとえば、等号の右側に指定される実数値が整数型を持つ記憶位置に格納される場合には、整数値に変換されるでしょう。通常、デバッガは現在の言語の代入規則に従うようにします。

異なる型のデータをプログラム記憶位置に格納できるようにするために、プログラム記憶位置に対応する型を変更する方法はいくつもあります。

- シンボリック名を持たないすべての記憶位置に対する省略時の型を変更するには、SET TYPE コマンドを使用して新しい型を指定できます。
- すべての記憶位置 (シンボリック名を持つものと持たないものの両方) に対して省略時の型を変更するには、SET TYPE/OVERRIDE コマンドを使用して新しい型を指定できます。
- 1 つの DEPOSIT コマンドの継続中に特定の記憶位置に現在対応している型を変更するには、修飾子 (/ASCII:*n* , /BYTE , /TYPE=(*name*) などを使用することにより新しい型を指定できます。

C または大文字小文字を区別する言語で記述されたプログラムをデバッグする場合、指定した型が大文字小文字混合または小文字のときは、DEPOSIT/TYPE コマンドを使用できません。たとえば、次のような関数を含むプログラムを仮定します。

## DEPOSIT

```
xyzzy_type foo ()
{
  xyzzy_type    z;
  z = get_z ();
  return (z);
}
```

次のコマンドを入力しようとする、デバッガは、“xyzzy\_type”型が見つからないというメッセージを発行します。

```
DBG> DEPOSIT/TYPE=(xyzzy_type) z="whatever"
```

デバッガは 2 進数，10 進数，16 進数，8 進数の 4 つの基数のどれか 1 つで整数データを解釈したり表示したりできます。

ほとんどの言語の場合，データの入力と表示の両方に対する省略時の基数は 10 進数です。例外は BLISS と MACRO です。これらの言語での省略時の基数は 16 進数です。

省略時の基数を変更するには，SET RADIX コマンドと SET RADIX/OVERRIDE コマンドを使用できます。

DEPOSIT コマンドは指定されたアドレス式で示される位置に現在の値の組み込みシンボル%CURLOC とピリオド(.)を設定します。論理的先行データ (%PREVLOC またはサーカンフレックス文字(^)) と論理的後続データ (%NEXTLOC) は現在の値に基づきます。

### 関連コマンド

```
CANCEL TYPE/OVERRIDE
EVALUATE
EXAMINE
MONITOR
(SET,SHOW,CANCEL) RADIX
(SET,SHOW) TYPE
```

---

## 例

1. DBG> DEPOSIT I = 7  
このコマンドは値 7 を整数変数 I に格納します。
2. DBG> DEPOSIT WIDTH = CURRENT\_WIDTH + 24.80  
このコマンドは CURRENT\_WIDTH + 24.80 という式の値を実変数 WIDTH に格納します。

3. DBG> DEPOSIT STATUS = FALSE

このコマンドは値 FALSE をブール変数 STATUS に格納します。

4. DBG> DEPOSIT PART\_NUMBER = "WG-7619.3-84"

この値は文字列 WG-7619.3-84 を文字列変数の PART\_NUMBER に格納します。

5. DBG> DEPOSIT EMPLOYEE.ZIPCODE = 02172

このコマンドは値 02172 をレコード EMPLOYEE の構成要素 ZIPCODE に格納します。

6. DBG> DEPOSIT ARR(8) = 35

DBG> DEPOSIT ^ = 14

この例では、最初の DEPOSIT コマンドは値 35 を配列 ARR の要素 8 に格納します。結果として、要素 8 が現在の値になります。2 番目のコマンドは値の 14 を要素 8 の論理的先行データ、つまり要素 7 に格納します。

7. DBG> FOR I = 1 TO 4 DO (DEPOSIT ARR(I) = 0)

このコマンドは配列 ARR の要素 1 から 4 に値 0 を格納します。

8. DBG> DEPOSIT COLOR = 3

%DEBUG-E-OPTNOTALLOW, operator "DEPOSIT" not allowed on  
given data type

間違った型のデータを変数に格納しようとする（この場合、整数値を列挙型変数に格納しようとした場合）デバッガはユーザに警告します。メッセージの重大度 E（エラー）はデバッガが代入を行わないことを表します。

9. DBG> DEPOSIT VOLUME = - 100

%DEBUG-I-IVALOUTBND, value assigned is out of bounds  
at or near '-'

範囲外の値（この場合には負の値）を変数に格納しようとする、デバッガはユーザに警告します。メッセージの重大度 I（情報）は、デバッガが代入を行うことを表します。

10. DBG> DEPOSIT/BYTE WORK = %HEX 21

このコマンドは%HEX 21 という式を WORK という記憶位置に格納し、それをバイト整数に変換します。

11. DBG> DEPOSIT/OCTAWORD BIGINT = 111222333444555

このコマンドは 111222333444555 の式を BIGINT 記憶位置に格納し、それをオクタワード整数に変換します。

12. DBG> DEPOSIT/FLOAT BIGFLT = 1.11949\*10\*\*35

このコマンドは 1.11949\*10\*\*35 を F 浮動小数点型の値に変換し、それを BIGFLT 記憶位置に格納します。

## DEPOSIT

13. DBG> DEPOSIT/ASCII:10 WORK+20 = 'abcdefghij'

このコマンドはシンボル **WORK** で示される記憶位置の 20 バイト先の記憶位置に文字列 "abcdefghij" を格納します。

14. DBG> DEPOSIT/INSTR SUB2+2 = 'MOVL #20A,R0'

VAX システムでは、このコマンドは命令 **MOVL #20A,R0** を SUB2 + 2 バイトの記憶位置に格納します。

15. DBG> DEPOSIT/TASK VAR = %TASK 2

DBG> EXAMINE/HEX VAR

SAMPLE.VAR: 0016A040

DBG> EXAMINE/TASK VAR

SAMPLE.VAR: %TASK 2

DBG>

DEPOSIT コマンドは Ada タスク値の %TASK 2 を記憶位置 VAR に格納します。以降の EXAMINE コマンドの 1 つは VAR の内容を 16 進数形式で表示します。もう 1 つの EXAMINE コマンドは VAR の内容をタスク値として表示します。

---

## DISABLE AST

プログラムでの非同期システム・トラップ (AST) の実行要求を無効にします。

---

### フォーマット

DISABLE AST

---

### 説明

DISABLE AST コマンドはプログラムでの AST の実行要求を無効にし、それにより、プログラムの実行中に割り込みが発生しないようにします。デバッガの動作中 (コマンドの処理中など) に AST の実行要求があると、それらの AST はキューに登録され、制御がプログラムに戻ったとき実行要求が受け付けられます。

ENABLE AST コマンドは保留中の AST (配付されるのを待っている AST) も含めて、AST の実行要求を再度可能にします。

---

#### 注意

AST を可能にする \$SETAST システム・サービスへのプログラムによる呼び出しは前回の DISABLE AST コマンドを上書きします。

---

### 関連コマンド

(ENABLE,SHOW) AST

---

### 例

```
DBG> DISABLE AST
DBG> SHOW AST
ASTs are disabled
DBG>
```

DISABLE AST コマンドはプログラムでの AST の実行要求を無効にします。これは SHOW AST コマンドによって確認できます。

---

# DISCONNECT

プロセスを終了せずに、そのプロセスをデバッガの制御から解放します。(保持デバッガのみ。)

---

## フォーマット

DISCONNECT *process-spec*

---

## パラメータ

*process-spec*

現在デバッガの制御下にあるプロセスを指定します。次のいずれかの形式で指定します。

`[%PROCESS_NAME] process-name`

スペースや小文字を含まないプロセス名。プロセス名にはワイルドカード文字(\*)を含めることができる。

`[%PROCESS_NAME] "process-name"`

スペースまたは小文字を含むプロセス名。二重引用符(")の代わりに、一重引用符(')を使用することもできる。

`%PROCESS_PID process_id`

プロセス識別子 (PID, 16 進数)。

`[%PROCESS_NUMBER] process-number`  
(または `%PROC process-number`)

デバッガの制御下に入ったときにプロセスに割り当てられた番号。新しい番号は、1 から順番に各プロセスに割り当てられる。EXIT コマンドまたは QUIT コマンドによってプロセスが終了した場合、そのデバッグ・セッション中にその番号が再割り当てされることがある。プロセス番号は SHOW PROCESS コマンドの実行で表示される。プロセスは、組み込みシンボル %PREVIOUS\_PROCESS および %NEXT\_PROCESS によってインデックスづけできるように、循環リスト内に順序づけされる。

`process-set-name`

DEFINE/PROCESS\_SET コマンドで定義された、プロセスのグループを表すシンボル。

`%NEXT_PROCESS`

デバッガの循環プロセス・リスト中で可視プロセスの次のプロセス。

`%PREVIOUS_PROCESS`

デバッガの循環プロセス・リスト中で可視プロセスの前のプロセス。

`%VISIBLE_PROCESS`

シンボル、レジスタ値、ルーチン呼び出し、ブレークポイントなどの検索時に現在のコンテキストになっているスタック、レジスタ・セット、およびイメージを持つプロセス。

---

## 説明

(保持デバッグのみ。) DISCONNECT コマンドは、指定されたプロセスを終了せずに、そのプロセスをデバッガの制御から解放します。このコマンドは、たとえば、CONNECT コマンドで実行中プログラムをデバッガの制御下に置いた後に、そのイメージを終了せずに解放したい場合などに役立ちます。(一方、プロセスに EXIT コマンドか QUIT コマンドを指定した場合、そのプロセスは終了します。)

---

### 注意

デバッガ・カーネルは、デバッグ中のイメージと同一のプロセスで稼働します。DISCONNECT コマンドをこのプロセスに実行すると、プロセスは解放されますが、カーネルはアクティブな状態のままです。

この状態は、プログラム・イメージの実行が終了するまで続きます。

新バージョンのデバッガをインストールする際に、切り離されているがアクティブな状態のいくつかのカーネルがユーザ・プログラム・スペースを占めている状態であると、それらのカーネルの 1 つに再接続しようとすると、デバッガの動きに問題が出る場合があります。

---

## 関連コマンド

EXIT  
QUIT  
CONNECT

---

## 例

DBG> DISCONNECT JONES

このコマンドは、プロセス JONES を終了せずに、そのプロセスをデバッガの制御から解放します。

---

## DISPLAY

新しい画面表示を作成するか、既存の表示を変更します。

---

### 注意

---

このコマンドは、デバッガへの HP DECwindows Motif for OpenVMS ユーザ・インタフェースでは使用できません。

---



---

## フォーマット

DISPLAY *display-name* [*AT window-spec*] [*display-kind*] [, . . . ]

---

## パラメータ

*display-name*

作成または変更の対象となる表示を指定します。

新しい表示を作成している場合には、ディスプレイ名としてまだ使用されていない名前を指定します。

既存の表示を変更している場合には、次の値のどれかを指定できます。

- 定義済み表示

SRC  
OUT  
PROMPT  
INST  
REG  
FREG (Alpha および I64 のみ)  
IREG

- DISPLAY コマンドで以前に作成した表示
- 表示組み込みシンボル

%CURDISP  
%CURSCROLL  
%NEXTDISP  
%NEXTINST  
%NEXTOUTPUT  
%NEXTSCROLL



## %NEXTSOURCE

/GENERATE (省略可能なパラメータ) または/REFRESH (許されていないパラメータ) を使用する場合以外は、画面を指定しなければなりません。

複数のオプションのウィンドウと表示対象を指定できます。

### window-spec

表示を位置づける画面ウィンドウを指定します。次の値のいずれかを指定できます。

- 定義済みウィンドウ。たとえば RH1 (右上半分)。
- SET WINDOW コマンドで以前に設定したウィンドウ定義。
- (*start-line, line-count*[, *start-column, column-count*])形式のウィンドウ指定。この指定で使用する式は組み込みシンボル%PAGE と%WIDTH (たとえば%WIDTH/4) を基準にできます。

ウィンドウ指定を省略すると、画面位置は既存の表示を指定したか、新しい表示を指定したかによって異なります。

- 既存の表示を指定している場合には、表示の位置は変わりません。
- 新しい表示を指定している場合には、その表示はウィンドウ H1 または H2 に位置づけられます。H1 と H2 は交互に使用され、新しい表示を作成するたびに切り替わります。

### display-kind

表示対象を指定します。有効なキーワードは次のとおりです。

DO ( <i>command</i> [; ... ])	自動的に更新される出力表示を指定します。デバッガに制御が移るたびに、コマンドがリストされている順番で実行されます。コマンドの出力が表示の内容となります。複数のコマンドを指定する場合、それぞれをセミコロンで区切る必要があります。
INSTRUCTION	機械語命令ディスプレイを指定します。SELECT/INSTRUCTION コマンドで現在の機械語命令ディスプレイとして選択された場合には、後続の EXAMINE/INSTRUCTION コマンドからの出力を表示します。
INSTRUCTION ( <i>command</i> )	自動的に更新される機械語命令ディスプレイを指定します。指定されるコマンドは EXAMINE/INSTRUCTION コマンドでなければなりません。機械語命令ディスプレイはデバッガに制御が移るたびに更新されます。
OUTPUT	出力表示を指定します。SELECT/OUTPUT コマンドで現在の出力表示として選択された場合には、他の表示には出力されないデバッガ出力をすべて表示します。SELECT/INPUT コマンドで現在の入力表示として選択された場合には、デバッガ入力をエコーバックします。SELECT/ERROR コマンドで現在のエラー表示として選択された場合には、デバッガ診断メッセージを表示します。

REGISTER	(VAX のみ) 自動的に更新されるレジスタ表示を指定します。デバッガに制御が移るたびに、表示は更新されます。
SOURCE	ソース表示を指定します。SELECT/SOURCE コマンドで現在のソース表示として選択された場合には、以降の TYPE コマンドまたは EXAMINE/SOURCE コマンドからの出力を表示します。
SOURCE ( <i>command</i> )	自動的に更新されるソース表示を指定します。指定されるコマンドは TYPE コマンドまたは EXAMINE/SOURCE コマンドでなければなりません。デバッガに制御が移るたびにソース表示は更新されます。

PROMPT 表示の表示対象は変更できません。

display-kind パラメータを省略すると、表示対象は既存の表示を指定しているのか、新しい表示を指定しているのかによって異なります。

- 既存の表示を指定している場合には、表示対象は変わりません。
- 新しい表示を指定している場合には、OUTPUT 表示が作成されます。

---

## 修飾子

### /CLEAR

指定の表示の内容全体を消去します。/GENERATE を指定したとき、または新しい表示を作成しているときにはこの修飾子は指定できません。

### /DYNAMIC (省略時の設定) /NODYNAMIC

SET TERMINAL コマンドによって画面の高さまたは幅が変更されたときに、表示がそのウィンドウの寸法を変更に応じて自動的に調整するかどうかを制御します。省略時の設定 (/DYNAMIC) では、ユーザ定義表示と定義済み表示はすべて自動的に寸法を調整します。

### /GENERATE

指定の表示の内容を再生成します。自動的に生成される表示だけが再生成されます。それには、DO 表示、レジスタ表示、ソース (*cmd-list*) 表示、命令 (*cmd-list*) 表示があります。デバッガは各プロンプトの前にこれらの種類の表示をすべて自動的に再生成します。表示を指定しないと、自動的に生成される表示すべての内容が再生成されます。/CLEAR を指定したとき、または新しい表示を作成したときには、この修飾子は使用できません。

### /HIDE

(/PUSH と同じく) 表示のペーストボードの下部に指定の表示を位置づけます。これは指定された表示を、画面の同じ領域を共用する他の表示の後ろに置き、表示しないようにします。PROMPT 表示を表示しないようにすることはできません。

/MARK\_CHANGE

/NOMARK\_CHANGE (省略時の設定)

自動的に更新されるたびに DO 表示の中で変化する行をマークするかどうかを制御します。他の種類の表示の場合には指定できません。

/MARK\_CHANGE を使用すると、最後の表示更新以降に内容が変更された行があればすべて反転表示で強調表示されます。この修飾子が特に役に立つのは、自動的に更新される表示の中の変数を変化したときにすべて強調表示させたいときです。

/NOMARK\_CHANGE 修飾子 (省略時の設定) は DO 表示の中で変化しない行は全くマークしないことを指定します。この修飾子は指定された表示に関する前回の /MARK\_CHANGE の作用を取り消します。

/POP (省略時の設定)

/NOPOP

省略時の設定。表示ペーストボードの上部の、他のすべての表示の前だが PROMPT の後に指定の表示を位置づけるかどうかを指定します。省略時の設定 (/POP) では、表示はペーストボードの上部に位置づけられ、画面の同じ領域を共有する PROMPT 表示以外の他の表示をすべて非表示にします。

/NOPOP 修飾子は (/NOPUSH と同じく) ペーストボード上のすべての表示の順番をそのままにします。

/PROCESS[=(process-spec)]

/NOPROCESS (省略時の設定)

マルチプロセス・プログラムをデバッグするときのみ使用 (保持デバッガのみ)。指定された表示をプロセス固有にするかどうか (つまり、指定された表示を特定のプロセスだけに対応させるか) を制御します。プロセス固有の表示の内容はそのプロセスのコンテキストで生成され、変更されます。PROMPT 表示以外であればどの表示でもプロセス固有なものにすることができます。

/PROCESS=(process-spec) 修飾子を使用すると、指定された表示を指定されたプロセスに対応づけられます。この修飾子を使用するときは括弧を含めなくてはなりません。次のいずれかの process-spec 形式で指定します。

[%PROCESS\_NAME] *proc-name*

スペースや小文字を含まないプロセス名。プロセス名にはワイルドカード文字 (\*) を含めることができる。

[%PROCESS\_NAME] "*proc-name*"

スペースまたは小文字を含むプロセス名。二重引用符 (") の代わりに一重引用符 (') を使用することもできる。

%PROCESS\_PID *proc-id*

プロセス識別子 (PID, 16 進数)。

%PROCESS\_NUMBER *proc-number*  
(または %PROC *proc-number*)

デバッガの制御下に入ったときにプロセスに割り当てられた番号。プロセス番号は SHOW PROCESS コマンドの実行で表示される。

<i>proc-group-name</i>	DEFINE/PROCESS_GROUP コマンドで定義された、プロセスのグループを表すシンボル。再帰的なシンボル定義を指定してはならない。
%NEXT_PROCESS	デバッガの循環プロセス・リスト中で可視プロセスの次のプロセス。
%PREVIOUS_PROCESS	デバッガの循環プロセス・リスト中で可視プロセスの前のプロセス。
%VISIBLE_PROCESS	シンボル、レジスタ値、ルーチン呼び出し、ブレークポイントなどの検索時に現在のコンテキストになっている呼び出しスタック、レジスタ・セット、およびイメージを持つプロセス。

/PROCESS 修飾子を使用すると、指定した表示は DISPLAY/PROCESS コマンドが実行されたときに可視プロセスになっていたプロセスに対応づけられます。

/NOPROCESS 修飾子 (これが省略時の設定) を使用すると、指定した表示は可視プロセスに対応づけられます。可視プロセスはプログラムの実行中に変化する可能性があります。

/PROCESS を指定しないと、指定した表示の (もしあれば) 現在のプロセス固有の動作はもとのままになります。

/PUSH

/NOPUSH

/PUSH 修飾子は/HIDE と同じ作用があります。/NOPUSH 修飾子は (/NOPOP と同じく) ペーストボードでのすべての表示の順番をそのまま維持します。

/REFRESH

端末画面を再表示します。この修飾子を指定した場合、コマンド・パラメータは指定できません。Ctrl/W を使用して画面を再表示することもできます。

/REMOVE

表示ペーストボードから除去されたものとして表示にマークし、他の DISPLAY コマンドで明示的に要求した場合を除き、その表示を画面上で示さないようにします。除去された表示は画面では見えませんが、その表示は依然として存在し、その内容は維持されます。PROMPT 表示を除去することはできません。

/SIZE:n

表示の最大サイズを *n* 行に設定します。*n* 行を超える行数が表示に書き込まれると、新しい行を追加するたびに一番古い行が失われます。この修飾子を省略すると、表示の最大サイズは次のようになります。

- 既存の表示を指定する場合、最大サイズはもとのままです。
- 表示を作成している場合、省略時のサイズは 64 行です。

出力または DO 表示の場合、/SIZE:*n*は*n*行分の最新出力が表示されるように指定します。ソース表示または機械語命令ディスプレイの場合、*n*はいつでも 1 つの時点でメモリ・バッファにいれられるソース行または命令の行数を定義します。ただし、そのコードを表示するモジュールのソース・コード全体に渡りソース表示をスクロールできます (ソース行は必要に応じてバッファにページングされます)。同様に、その命令を表示するルーチンの命令すべてに渡り機械語命令ディスプレイをスクロールできます (命令は必要に応じてイメージからデコードされます)。

---

## 説明

DISPLAY コマンドを使用すれば表示、作成、または既存表示の変更ができます。

表示を作成するには、ディスプレイ名としてまだ使用されていない名前を指定します (SHOW DISPLAY コマンドを使用すれば、既存の表示がすべて表示されます)。

省略時の設定では、DISPLAY コマンドは、表示ペーストボードの上部の PROMPT 表示の後ろで他のすべての表示の前に指定された表示を置きます。PROMPT 表示は非表示にすることはできません。指定された表示によって (PROMPT 表示以外の) 他の表示で画面上の同じ領域を共用する部分が表示されなくなります。

DISPLAY コマンドに対応するキー定義のリストは、ヘルプ・トピックの Keypad\_Definitions\_CI を参照してください。また、現在のキー定義を調べるには SHOW KEY コマンドを使用してください。

### 関連コマンド

```
Ctrl/W
EXPAND
MOVE
SET PROMPT
(SET,SHOW) TERMINAL
(SET,SHOW,CANCEL) WINDOW
SELECT
(SHOW,CANCEL) DISPLAY
```

---

## 例

1. DBG> DISPLAY REG

このコマンドは REG という定義済みレジスタ表示をその現在のウィンドウ位置に表示します。

2. DBG> DISPLAY/PUSH INST

このコマンドは表示 INST を表示ペーストボードの下部で、他のすべての表示の後ろに差し込みます。

3. DBG> DISPLAY NEWDISP AT RT2  
DBG> SELECT/INPUT NEWDISP

この例では、DISPLAY コマンドは画面の中央右の 3 分の 1 にユーザ定義表示 NEWDISP を表示します。SELECT/INPUT コマンドは現在の入力表示として NEWDISP を選択します。NEWDISP はデバッガ入力をエコーバックします。

4. DBG> DISPLAY DISP2 AT RS45  
DBG> SELECT/OUTPUT DISP2

この例では、DISPLAY コマンドは基本的に画面の右下半分で、PROMPT 表示の上に DISP2 という名前の表示を作成します。PROMPT 表示は S6 に位置しています。省略時の設定では、これは出力表示です。SELECT/OUTPUT コマンドは現在の出力表示として DISP2 を選択します。

5. DBG> SET WINDOW TOP AT (1,8,45,30)  
DBG> DISPLAY NEWINST AT TOP INSTRUCTION  
DBG> SELECT/INST NEWINST

この例では、SET WINDOW コマンドは 1 行目の 45 列目から始まり、下に 8 行、右に 30 列続く TOP という名前のウィンドウを作成します。DISPLAY コマンドは TOP を通じて表示される NEWINST という名前の機械語命令ディスプレイを作成します。SELECT/INST コマンドは現在の機械語命令ディスプレイとして NEWINST を選択します。

6. DBG> DISPLAY CALLS AT Q3 DO (SHOW CALLS)

このコマンドはウィンドウ Q3 に CALLS という名前の DO 表示を作成します。デバッガがプログラムから制御を受け取るたびに、SHOW CALLS コマンドが実行され、出力が CALLS 表示に表示され、もとの内容を置換します。

7. DBG> DISPLAY/MARK EXAM AT Q2 DO (EXAMINE A,B,C)

このコマンドはウィンドウ Q2 に EXAM という名前の DO 表示を作成します。デバッガが入力を求めるプロンプトを表示するたびに、表示には変数 A、B、C の現在の値が示されます。変更された値はどれも強調表示されます。

8. all> DISPLAY/PROCESS OUT\_X AT S4

このコマンドは表示 OUT\_X を可視プロセス (プロセス 3) に固有のものにし、ウィンドウ S4 に表示を置きます。

---

# DUMP

メモリの内容を表示します。

---

## フォーマット

DUMP *address-expression1* [:*address-expression2*]

---

## パラメータ

*address-expression1*

表示する最初の記憶位置を指定します。

*address-expression2*

表示する最後の記憶位置を指定します (省略時の設定は *address-expression1* です)。

---

## 修飾子

/BINARY

確認した個々の値を 2 進整数として表示します。

/BYTE

確認した個々の値をバイト整数 (1 バイト長) として表示します。

/DECIMAL

確認した個々の値を 10 進整数として表示します。

/HEXADECIMAL

確認した個々の値を 16 進整数として表示します。

/LONGWORD (省略時の設定)

確認した個々の値をロングワード整数型 (4 バイト長) で表示します。これは、コンパイラで生成されたデータ型のないプログラム記憶位置の省略時のデータ型です。

/OCTAL

確認した個々の値を 8 進整数として表示します。

/QUADWORD

確認した個々の値をクオードワード整数型 (8 バイト長) で表示します。

/WORD

確認した個々の値をワード整数型 (2 バイト長) で表示します。

---

説明

DUMP コマンドは、レジスタ、変数、配列も含めて、メモリの内容を表示します。DUMP コマンドは DCL の DUMP コマンドと同じ方法で出力をフォーマットします。デバugga の DUMP コマンドは集合体の構造体を解釈しません。

一般に、DUMP コマンドを入力すると、デバugga は address-expression1 を評価して、プログラム記憶位置を求めます。そのあと、デバugga は次の方法で、その記憶位置に格納されている値を表示します。

- 値にシンボル名が割り当てられている場合は、デバugga は値のサイズを使用して、表示するアドレスの範囲を確認します。
- 値にシンボル名がない場合には (したがって、コンパイラで生成されたデータ型が割り当てられていない場合)、デバugga は address-expression2 (指定されている場合) を通じて、address-expression1 を表示します。

どちらの場合も、DUMP コマンドはこれらの記憶位置の内容を現在の基数で、ロングワード (省略時の設定) 整数値として表示します。

表示のために使用する省略時の基数は、ほとんどの言語で 10 進数です。例外は BLISS と MACRO です。これらの言語での省略時の基数は 16 進数です。

他の基数でデータを表示するには、4 つの基数修飾子 (/BINARY, /DECIMAL, /HEXADECIMAL, /OCTAL) のいずれかを使用します。また、SET RADIX コマンドと SET RADIX/OVERRIDE コマンドを使用して、省略時の基数を変更することもできます。

表示の形式を変更するには、サイズ修飾子 (/BYTE, /WORD, /LONGWORD, /QUADWORD) のいずれかを使用します。

DUMP コマンドは、現在のエンティティ組み込みシンボル %CURLOC とピリオド (.) を、指定されたアドレス式によって示される記憶位置に設定します。論理的に前の記憶位置 (%PREVLOC またはサーカンフレックス文字 (^)) と後の記憶位置 (%NEXTLOC) は、現在のエンティティの値をもとに決定されます。

関連コマンド:

EXAMINE



## 例

```

1.  DBG> DUMP/QUAD R16:R25
      0000000000000078 0000000000030038 8.....x..... %R16
      000000202020786B 0000000000030041 A.....kx    ... %R18
      0000000000030140 0000000000007800 .x.....@..... %R20
      0000000000010038 0000000000000007 .....8..... %R22
      0000000000000006 0000000000000000 ..... %R24

```

DBG>

このコマンドは、汎用レジスタ R16 ~ R25 をクォードワード形式の 16 進数で表示します。

```

2.  DBG> DUMP APPLES
      00000000 00030048 00000000 00004220 B.....H..... 00000000000300B0
      63724F20 746E6F6D 646F6F57 000041B0 °A..Woodmont Orc 00000000000300C0
      20202020 20202020 20202073 64726168 hards          00000000000300D0
      6166202C 73646E61 6C747275 6F432020 Courtlands, fa 00000000000300E0
                                   00002020 2079636E ncy    .. 00000000000300F0

```

DBG>

このコマンドは、APPLES というエンティティの値をロングワード形式の 16 進数で表示します。

```

3.  DBG> DUMP/BYTE/DECIMAL 30000:30040
      0   0   0   0   0   3   0  -80 °..... 0000000000030000
      0   0   0   0   0   3   1   64 @..... 0000000000030008
      0   0   0   0   0   3   0   48 0..... 0000000000030010
      0   0   0   0   0   3   0   56 8..... 0000000000030018
      0   0   0   0   0   3   0  -64
                                   `A..... 0000000000030020
      0   0   0   0   0   3   0  -80 °..... 0000000000030028
      0   0   0   0   0   0   7  -50
                                   ^I..... 0000000000030030
      101 101 119  32 116 120 101 110 next wee 0000000000030038
                                   107 k          0000000000030040

```

DBG>

このコマンドは、記憶位置 30000 ~ 30040 をバイト形式の 10 進数で表示します。

---

## EDIT

SET EDITOR コマンドで設定されたエディタを起動します。ユーザが SET EDITOR コマンドを入力していなかった場合には、システムにインストールされていればランゲージ・センシティブ・エディタ (LSE) を起動します。

---

### フォーマット

EDIT *[[module-name\] line-number]*

---

### パラメータ

module-name

編集の対象となるソース・ファイルが含まれているモジュールの名前を指定します。モジュール名を指定する場合には、行番号も指定しなければなりません。module-name パラメータを省略すると、現在のソース表示に表示されているコードを持つソース・ファイルが編集の対象として選択されます。

line-number

初期状態でエディタのカーソルをどのソース行に置くかを指定する正の整数。このパラメータを省略すると、カーソルは、初期状態ではデバッガの現在のソース表示の中央のソース行の先頭に置かれます。エディタが/NOSTATRT\_POSITION (SET EDITOR コマンドを参照) に設定されていた場合には 1 行目の先頭に置かれます。

---

### 修飾子

/EXIT

/NOEXIT (省略時の設定)

エディタを起動する前にデバッグ・セッションを終了するかどうかを制御します。/EXIT を指定すると、デバッグ・セッションが終了されたあと、エディタが起動されます。/NOEXIT を指定すると、編集セッションが開始され、その編集セッションが終了したあと、デバッグ・セッションに戻ります。

---

### 説明

SET EDITOR コマンドでエディタが指定されていなかった場合には、EDIT コマンドはスポンされたサブプロセスで (ランゲージ・センシティブ・エディタ (LSE) がシステムにインストールされている場合には) LSE を起動します。EDIT コマンドの通常 (省略時) の使用法では、パラメータを全く指定しません。この場合、編集カーソ

ルは、初期状態では現在選択されているデバッガ・ソース表示 (現在のソース表示) の中央にある行の先頭に置かれます。

SET EDITOR コマンドには、サブプロセス内から各種のエディタを起動するためのオプションや、呼び出し可能インタフェースを使用して各種のエディタを起動するためのオプションがあります。

#### 関連コマンド

(SET,SHOW) EDITOR

(SET,SHOW,CANCEL) SOURCE

---

#### 例

1. DBG> EDIT

このコマンドは現在のソース・ディスプレイに表示されるコードを持つソース・ファイルを編集するために、サブプロセスとしてランゲージ・センシティブ・エディタ (LSE) をスポンします。編集カーソルはソース表示の中央にある行の先頭に置かれます。

2. DBG> EDIT SWAP\12

このコマンドは、SWAP というモジュールがあるソース・ファイルを編集するために、サブプロセスとしてランゲージ・センシティブ・エディタ (LSE) を作成します。編集カーソルはソース行の 12 行目の先頭に置かれます。

3. DBG> SET EDITOR/CALLABLE\_EDT  
DBG> EDIT

この例では、SET EDITOR/CALLABLE\_EDT コマンドは EDT が省略時のエディタであることと、それが (サブプロセスとして作成するのではなく) 呼び出し可能インタフェースを通じて起動されることを指定しています。EDIT コマンドは現在のソース表示に表示されるコードを持つソース・ファイルを編集するために EDT を起動します。省略時の修飾子である/NOSTART\_POSITION が EDT に対して有効であるため、編集カーソルはソース行の 1 行目の先頭に置かれます。

---

## ENABLE AST

プログラムでの非同期システム・トラップ (AST) の実行要求を許可します。

---

### フォーマット

ENABLE AST

---

### 説明

ENABLE AST コマンドはプログラムの実行中に AST の実行要求を許可します。これらの AST には保留中の AST d (配布されるのを待っている AST) も含みます。デバッガの動作中 (コマンドの処理中など) に AST が生成されると、AST はキューに登録され、制御がプログラムに戻ったとき配布されます。AST の実行要求は省略時の設定では許可されません。

---

#### 注意

AST を禁止する \$SETAST システム・サービスへの呼び出しを行うと、前回の ENABLE AST コマンドは上書きされます。

---

### 関連コマンド

(DISABLE,SHOW) AST

---

### 例

```
DBG> ENABLE AST
DBG> SHOW AST
ASTs are enabled
DBG>
```

ENABLE AST コマンドで AST の実行要求を許可します。それを SHOW AST コマンドによって確認します。

---

## EVALUATE

現在の言語 (省略時の設定では、メイン・プログラムを含むモジュールの言語) で言語式の値を表示します。

---

### フォーマット

EVALUATE *language-expression*[, ... ]

---

### パラメータ

*language-expression*

現在の言語での有効な式を指定します。

---

### 修飾子

/BINARY

結果を 2 進基数で表示することを指定します。

/CONDITION\_VALUE

式を状態値 (条件処理メカニズムを使用して指定する状態値の種類) として解釈することを指定します。その状態値に対応するメッセージ文が表示されます。指定する値は整数値でなければなりません。

/DECIMAL

結果を 10 進基数で表示することを指定します。

/HEXADECIMAL

結果を 16 進基数で表示することを指定します。

/OCTAL

結果を 8 進基数で表示することを指定します。

---

### 説明

デバッガは EVALUATE コマンドで指定された式を言語式として解釈し、現在の言語の構文と現在の基数で評価し、その値を現在の言語でリテラル (たとえば、整数値) として表示します。

現在の言語は前回 SET LANGUAGE コマンドで設定された言語です。SET LANGUAGE コマンドが入力されていなかった場合には、現在の言語は省略時の設定ではメイン・プログラムを含むモジュールの言語です。

式に種々のコンパイラ生成型のシンボルが含まれている場合には、デバッガは現在の言語の型変換規則を使用してその式を評価します。

デバッガは 2 進数, 10 進数, 16 進数, 8 進数の 4 つの基数のどれか 1 つで整数データを解釈したり表示したりできます。現在の基数は前回 SET RADIX コマンドで設定した基数です。

SET RADIX コマンドを入力しなかった場合、データの入力と表示のどちらに対しても、ほとんどの言語の省略時の基数は 10 進数です。例外は BLISS と MACRO です。これらの言語での省略時の基数は 16 進数です。

基数修飾子 (/BINARY, /OCTAL など) を使用すれば別の基数で整数データを表示できます。これらの修飾子はユーザが指定したデータをデバッガが解釈する方法には影響を及ぼしません。現在の出力基数は上書きされますが、入力基数はもとのままです。

EVALUATE コマンドは現在の値の組み込みシンボル %CURVAL と円記号 (\) を指定された式で示される値に設定します。

関数呼び出しが含まれている言語式を評価することはできません。たとえば、PRODUCT が 2 つの整数を乗算する関数である場合には、EVALUATE PRODUCT(3,5) というコマンドは使用できません。関数の返却値を変数に代入してその変数を調べれば、結果として関数呼び出しを含む式を評価できます。

Alpha プロセッサでは、EVALUATE *procedure-name* というコマンドは指定されたルーチン、エントリ・ポイントまたは Ada パッケージの (コード・アドレスではなく) プロシージャディスクリプタアドレスを表示します。

言語に固有の演算子と構造に対するデバッガ・サポートについての詳しい説明は、HELP Language をタイプしてください。

#### 関連コマンド

EVALUATE/ADDRESS  
MONITOR  
(SET,SHOW) LANGUAGE  
(SET,SHOW,CANCEL) RADIX  
(SET,SHOW) TYPE

---

例

1. DBG> EVALUATE 100.34 \* (14.2 + 7.9)  
2217.514  
DBG>

このコマンドは、デバッグを計算機として使用することにより、100.34 と (14.2 + 7.9) の乗算を行います。

2. DBG> EVALUATE/OCTAL X  
00000001512  
DBG>

このコマンドはシンボル X を評価し、その結果を 8 進基数で表示します。

3. DBG> EVALUATE TOTAL + CURR\_AMOUNT  
8247.20  
DBG>

このコマンドは、TOTAL と CURR\_AMOUNT という 2 つの実変数の値の合計を評価します。

4. DBG> DEPOSIT WILLING = TRUE  
DBG> DEPOSIT ABLE = FALSE  
DBG> EVALUATE WILLING AND ABLE  
False  
DBG>

この例では、EVALUATE コマンドは WILLING と ABLE という 2 つのブール変数の現在の値の論理積を評価します。

5. DBG> EVALUATE COLOR'FIRST  
RED  
DBG>

この Ada の例では、このコマンドは COLOR という列挙型の最初の要素を評価します。

---

## EVALUATE/ADDRESS

アドレス式を評価し，その結果をメモリ・アドレスまたはレジスタ名として表示します。

---

### フォーマット

EVALUATE/ADDRESS *address-expression*[, . . . ]

---

### パラメータ

*address-expression*

有効な形式のアドレス式 (たとえば，ルーチン名，変数名，ラベル，行番号など) を指定します。

---

### 修飾子

/BINARY

メモリ・アドレスを 2 進基数で表示します。

/DECIMAL

メモリ・アドレスを 10 進基数で表示します。

/HEXADECIMAL

メモリ・アドレスを 16 進基数で表示します。

/OCTAL

メモリ・アドレスを 8 進基数で表示します。

---

### 説明

EVALUATE/ADDRESS コマンドを使用すれば，アドレス式に対応するメモリ・アドレスまたはレジスタを求められます。

デバッガは 2 進数，10 進数，16 進数，8 進数の 4 つの基数のどれか 1 つで整数データを解釈したり表示したりできます。

ほとんどの言語の場合，データの入力と表示のどちらに対しても，省略時の基数は 10 進数です。例外は BLISS と MACRO です。これらの言語での省略時の基数は 16 進数です。



基数修飾子 (/BINARY , /OCTAL など) を使用すれば別の基数でアドレス・データを表示できます。これらの修飾子はユーザが指定したデータをデバッガが解釈する方法には影響を及ぼしません。現在の出力基数は上書きされますが、入力基数はもとのままです。

変数の値がメモリではなくレジスタに現在格納されている場合には、EVALUATE/ADDRESS コマンドはレジスタを識別します。この場合、基数修飾子は全く働きを持ちません。

EVALUATE/ADDRESS コマンドは、指定されたアドレス式で示される記憶位置に現在の要素組み込みシンボルの%CURLOC とピリオド(.)を設定します。論理的先行データ (%PREVLOC またはサーカンフレックス文字(^)) と論理的后続データ (%NEXTLOC) は現在の要素の値に基づきます。

Alpha プロセッサでは、EVALUATE/ADDRESS *procedure-name* というコマンドは指定されたルーチン、エントリ・ポイントまたは Ada パッケージの(コード・アドレスではなく) プロシージャディスクリプタアドレスを表示します。

関連コマンド

```
EVALUATE
(SET,SHOW,CANCEL) RADIX
SHOW SYMBOL/ADDRESS
SYMBOLIZE
```

---

## 例

1. DBG> EVALUATE/ADDRESS MODNAME\%LINE 110  
3942  
DBG>

このコマンドはアドレス式 MODNAME\%LINE 110 によって示されるメモリ・アドレスを表示します。

2. DBG> EVALUATE/ADDRESS/HEX A,B,C  
000004A4  
000004AC  
000004A0  
DBG>

このコマンドはアドレス式 A , B , C で示されるメモリ・アドレスを 16 進基数で表示します。

## EVALUATE/ADDRESS

```
3. DBG> EVALUATE/ADDRESS X
MOD3\%R1
DBG>
```

このコマンドは変数 X がレジスタ R1 に対応していることを示します。X は非静的 (レジスタ) 変数です。

---

## EXAMINE

プログラム変数の現在の値を表示します。より通常は、アドレス式で示される要素の値を表示します。

---

### フォーマット

EXAMINE *[address-expression[:address-expression] [, . . . ]]*

---

### パラメータ

address-expression

値を調べる要素を指定します。高級言語を使用する場合、これは通常、変数の名前になり、変数を一意に指定するためのパス名を含めることができます。より通常は、アドレス式はメモリ・アドレスまたはレジスタにすることもでき、1つまたは複数の演算子、オペランドまたは区切り文字だけでなく、数字(オフセット)とシンボルで構成することができます。レジスタのデバッグ・シンボルについてとアドレス式の中で使用できる演算子についての詳しい説明は、ヘルプ・トピックの Built\_in\_Symbols または Address\_Expressions を参照してください。

集合体変数(配列またはレコード構造体などの複合データ構造体)の名前を指定すると、デバッグはすべての要素の値を表示します。配列の場合、各配列要素の添字(インデックス)と値が示されます。レコードの場合、各レコードの構成要素の名前と値が示されます。

個々の配列要素、配列断面またはレコードの構成要素を指定するには、現在の言語の構文に従います。

要素の範囲を指定する場合、その範囲にある最初の要素を示すアドレス式の値は同じ範囲にある最後の要素を示すアドレス式の値よりも小さくしなければなりません。デバッグは最初のアドレス式で指定される値、そのアドレス式の論理的後続データ、その次の論理的後続データという順に、最後のアドレス式で指定される要素までを表示します。コンマで範囲と範囲を区切ることにより、複数の範囲からなるリストを指定できます。

ベクタ・レジスタとベクタ命令に特有の情報については、/TMASK、/FMASK、/VMR および/OPERANDS の修飾子を参照してください。

---

修飾子

/ASCIC

/AC

値を調べる個々の要素を，1 バイトのカウント・フィールドの後ろに続く，このカウント・フィールドにより長さを指定された ASCII 文字列と解釈します。

/ASCID

/AD

値を調べる個々の要素を ASCII 文字列を指す文字列ディスクリプタのアドレスと解釈します。ディスクリプタの CLASS フィールドと DTYPE フィールドは表示されませんが，LENGTH フィールドと POINTER フィールドが ASCII 文字列の文字長とアドレスを与えます。そのあと文字列が表示されます。

/ASCII:n

値を調べる個々の要素を長さ  $n$  バイト ( $n$  文字) の ASCII 文字列と解釈し，表示します。 $n$  を省略すると，デバッグはアドレス式の型をもとに長さを求めようとします。

/ASCIW

/AW

値を調べる個々の要素を，2 バイトのカウント・フィールドの後ろに続く，このカウント・フィールドにより長さを指定された ASCII 文字列と解釈します。それから文字列が表示されます。

/ASCIZ

/AZ

値を調べる個々の値を最後部に 0 のある ASCII 文字列と解釈します。最後部の 1 バイト分の 0 は文字列の終了を示します。それから文字列が表示されます。

/BINARY

値を調べる個々の値を 2 進整数として表示します。

/BYTE

値を調べる個々の値をバイト整数型 (1 バイト長) で表示します。

/CONDITION\_VALUE

値を調べる個々の値を状態値の戻り状態と解釈し，その戻り状態に対応するメッセージを表示します。

/D\_FLOAT

値を調べる個々の値を D 浮動小数点型 (8 バイト長) で表示します。

/DATE\_TIME

値を調べる個々の値を日付と時刻の内部表現を含むクォードワード整数 (8 バイト長) と解釈します。値は *dd-mmm-yyyy hh:mm:ss.cc* の形式で表示します。

/DECIMAL

値を調べる個々の値を 10 進整数として表示します。

**/DEFAULT**

値を調べる個々の値を省略時の基数で表示します。

この修飾子は/DEFA と短縮できます。

**/DEFINITIONS=n**

(Alpha のみ, 最適化コードがサポートされた場合は I64) コードが最適化されると, スプリット・ライフタイム変数用に *n* 定義ポイントを表示します。定義ポイントは, 変数が値を受け取る位置です。省略時の設定では, 最高 5 つまでの定義ポイントが表示されます。(明示的または省略時の設定で) 指定した数より多い定義ポイントが使用可能な場合は, 指定した数以上の定義ポイントも報告されます。スプリット・ライフタイム変数についての詳しい説明は, 『OpenVMS デバッガ説明書』を参照してください。

この修飾子は/DEFI と短縮できます。

**/EXTENDED\_FLOAT****/X\_FLOAT**

(Alpha および I64 のみ) 値を調べる個々の値を IEEE の X 浮動小数点型 (16 バイト長) で表示します。

**/FLOAT**

VAX プロセッサでは, 値を調べる個々の値を F 浮動小数点型 (4 バイト長) で表示します。

Alpha プロセッサでは, 値を調べる個々の値を IEEE T 浮動小数点型 (倍精度, 8 バイト長) で表示します。

**/F\_FLOAT**

(VAX のみ) 確認した個々の値を F 浮動小数点型 (4 バイト長) で表示します。

**/FPCR**

(Alpha のみ) 値を調べる個々の値を FPCR (浮動小数点制御レジスタ) 形式で表示します。

**/FMASK[=(mask-address-expression)]**

VAX のベクタ化されたプログラムの場合に指定できます。/TMASK 修飾子を参照してください。

**/G\_FLOAT**

VAX システムでは, 値を調べる個々の値を G 浮動小数点型 (8 バイト長) で表示します。

**/H\_FLOAT**

(VAX のみ) 値を調べる個々の値を H 浮動小数点型 (16 バイト長) で表示します。

**/HEXADECIMAL**

値を調べる個々の値を 16 進整数で表示します。

**/INSTRUCTION**

値を調べる個々の値をアセンブリ言語命令 (命令の長さは命令オペランドの数とアドレッシング・モードの種類によって変化) として表示します。/OPERANDS 修飾子も参照してください。

画面モードでは、EXAMINE/INSTRUCTION コマンドの出力は、現在機械語命令ディスプレイがあればそこに送られます。出力表示や DO 表示には送られません。機械語命令ディスプレイでの矢印は値を調べる命令を指します。

Alpha プロセッサでは、EXAMINE/INSTRUCTION *procedure-name* コマンドは指定されたルーチン、エントリ・ポイントまたは Ada パッケージのコード・アドレスにある最初の命令を表示します。

**/LINE (省略時の設定)****/NOLINE**

プログラム記憶位置を行番号 (%LINE *x*) で表示するか、*routine-name + byte-offset* として表示するかを制御します。省略時の設定 (/LINE) では、デバッガは行番号でプログラム記憶位置をシンボル化します。

**/LONG\_FLOAT****/S\_FLOAT**

(Alpha および I64 のみ) 値を調べる個々の値を IEEE S 浮動小数点型 (単一精度、4 バイト長) で表示します。

**/LONG\_LONG\_FLOAT****/T\_FLOAT**

(Alpha および I64 のみ) 値を調べる個々の値を IEEE T 浮動小数点型 (倍精度、8 バイト長) で表示します。

**/LONGWORD**

値を調べる個々の値をロングワード整数型 (4 バイト長) で表示します。これがコンパイラ生成型を持たないプログラム記憶位置の場合の省略時の型です。

**/OCTAL**

値を調べる個々の値を 8 進整数で表示します。

**/OCTAWORD**

個々の値をオクタワード整数型 (16 バイト長) で表示します。

**/OPERANDS[=keyword]**

(VAX のみ) 値を調べる命令に対応するオペランド情報を表示します (各オペランドのアドレスとその内容を、そのオペランドのデータ型を使用して表示します)。BRIEF と FULL のキーワードはレジスタ以外のオペランドに関して表示される情報の量を変えます。省略時の設定は/OPERANDS=BRIEF です。

現在の PC 値 (たとえば、EXAMINE/OPERANDS .0\%PC) での命令を値を調べる場合にだけ/OPERANDS を使用してください。現在の PC 値にない命令のオペランドの値を調べると、マシンの状態 (レジスタの内容) がその命令に合わせて設定されていないため、間違った結果が出力される恐れがあります。

画面モードでは、オペランド情報は現在の出力表示に出力されます。

ベクタ命令のオペランドの値を調べるときは、省略時の設定では、その命令に対応している可能性のあるオペランド要素のマスキングが実行されます。/TMASK 修飾子と/FMASK 修飾子を使用すると、他のいくつかのマスクを指定できます。ベクタ長レジスタ (VLR) の現在の値により、表示できるベクタ・レジスタの最上位要素が制限されます。

SET MODE [NO]OPERANDS=*keyword* コマンドも参照してください。このコマンドを使用すれば、命令を調べるときに表示されるオペランド情報の量に省略時のレベルを設定できます。

/PACKED:n

値を調べるそれぞれの値をパック 10 進数と解釈します。*n* の値は 10 進数の桁数です。1 つの桁は 1 ニブル (4 ビット) を占めます。

/PS

(Alpha のみ) 値を調べる個々の値を PS (プロセッサ・ステータス・レジスタ) 形式で表示します。

/PSL

(VAX のみ) 値を調べる個々の値を PSL (プロセッサ・ステータス・ロングワード) 形式で表示します。

/PSR

(I64 のみ) 値を調べる個々の値を PSR (プロセッサ・ステータス・レジスタ) 形式で表示します。

/PSW

(VAX のみ) 値を調べる個々の値を PSW (プロセッサ・ステータス・ワード) 形式で表示します。/PSW 修飾子は下位ワード (2 バイト) だけが表示されること以外は、/PSL に似ています。

/QUADWORD

値を調べる個々の値をクォドワード整数型 (8 バイト長) で表示します。

/S\_FLOAT

(Alpha のみ) 値を調べる個々の値を IEEE S 浮動小数点型 (単一精度, 4 バイト長) で表示します。

/SFPCR

Alpha プロセッサでは、値を調べる個々の値を SFPCR (ソフトウェア浮動小数点制御レジスタ) 形式で表示します。

/SOURCE

---

 注意
 

---

この修飾子は、デバッガへの HP DECwindows Motif for OpenVMS ユーザ・インタフェースでは使用できません。

---

値を調べる個々の値の記憶位置に対応するソース行を表示します。表示する値は機械語コード命令に対応していなければならない、したがって行番号、ラベル、ルーチン名または命令のメモリ・アドレスでなければなりません。表示する値はデータに対応する変数名やその他のアドレス式にすることはできません。

画面モードでは、EXAMINE/SOURCE コマンドの出力は、現在ソース表示があればそこに出力されます。出力表示や DO 表示には出力されません。ソース表示での矢印は指定された最後の要素 (または要素のリストで指定した最後の要素) に対応するソース行を指します。

Alpha プロセッサでは、EXAMINE/SOURCE *procedure-name* コマンドは指定されたルーチン、エントリ・ポイントまたは Ada パッケージのコード・アドレスにソース・コードを表示します。

/SYMBOLIC (省略時の設定)

/NOSYMBOLIC

シンボル化を行うかどうかを制御します。省略時の設定 (/SYMBOLIC) では、デバッガはできればすべてのアドレスをシンボル化します。つまり、数値アドレスをシンボル表現に変換します。/NOSYMBOLIC を指定すると、デバッガは絶対アドレスとして指定された要素のシンボル化を行わなくなります。要素を変数名として指定すると、シンボル化は行われます。/NOSYMBOLIC 修飾子は、(シンボリック名が数値アドレスに対して存在していても) シンボリック名ではなく、数値アドレスを表示した場合に役立ちます。/NOSYMBOLIC を使用すると、デバッガは数字を名前に変換する必要がないため、コマンド処理の速度が上がる可能性があります。

/TASK

タスキング (マルチスレッド) プログラムの場合に指定できます。値を調べる個々の値をタスク (スレッド)・オブジェクトとして解釈し、そのタスク・オブジェクトのタスク値 (名前またはタスク ID) を表示します。タスク・オブジェクトの値を調べるときは、プログラミング言語に組み込みタスキング・サービスが備っていない場合だけ /TASK を使用してください。

/TYPE=(name)

値を調べる個々の値を *name* (プログラムで宣言された変数またはデータ型の名前でなければならない) で指定された型に応じて解釈し、表示します。これによりユーザ宣言型を指定できます。型式は括弧で囲まなければならない。

/VARIANT=variant-selector address-expression

/VARIANT=(variant-selector,...) address-expression

デバッガが匿名バリエーションを発見したときに、正しい項目を表示できるようにします。



C プログラムの共用体に含まれているメンバは、個々の時点でいずれか 1 つのみが有効となっています。共用体を表示するとき、デバッガはその時点でどのメンバが有効なのかを知りません。

PASCAL プログラムのバリエーション部を含んでいるレコードでは、個々の時点でいずれか 1 つのバリエーションのみが有効となっています。匿名バリエーション部を含んでいるレコードを表示するとき、デバッガはその時点でどのバリエーションが有効なのかはわからず、省略時の設定ではすべてのバリエーションを表示します。

EXAMINE コマンドの/VARIANT 修飾子を使用すると、共用体 (C) または匿名バリエーション (PASCAL) のどのメンバを表示するかを選択することができます。

/WCHAR\_T[:n]

確認した個々の値を解釈し、n ロングワード (n 文字) のマルチバイト・ファイル・コード・シーケンスとして表示します。省略時の値は 1 ロングワードです。

確認した文字列を変換する場合、デバッガは、実行されているプロセスのロケール・データベースを使用します。省略時の設定は C ロケールです。

/WORD

値を調べる個々の要素をワード整数型 (2 バイト長) で表示します。

/X\_FLOAT

(Alpha および I64 のみ) 確認した個々の値を IEEE X 浮動小数点型 (16 バイト長) で表示します。

---

## 説明

EXAMINE コマンドはアドレス式で示す記憶位置に要素を表示します。このコマンドを使用すれば、プログラムでアクセスできるメモリ記憶位置またはレジスタの内容を表示できます。高級言語の場合、このコマンドは変数 (整数、実数、文字列、配列、レコードなど) の現在の値を獲得するために使用されることがほとんどです。

最適化されたコードをデバッグすると、EXAMINE コマンドは、スプリット・ライフタイム変数が値を受け取る定義ポイントを表示します。スプリット・ライフタイム変数についての詳しい説明は、『デバッガ説明書』を参照してください。省略時の設定では、EXAMINE コマンドは、最高 5 つまでの定義ポイントを表示します。/DEFINITIONS 修飾子を使用すると、定義ポイントの数を指定できます。

デバッガはシンボリック・アドレス式 (プログラムで宣言されたシンボリック名) に対応するコンパイラ生成型を認識します。シンボリック・アドレス式には次の要素が含まれます。

- 変数名。EXAMINE コマンドで変数を指定するときは、ソース・コードで使用される構文と同じ構文を使用する。

- ルーチン名，ラベル，行番号。これらは命令に対応している。変数の値を調べるときと同じ方法を使用して命令も表示できる。

通常，EXAMINE コマンドを入力すると，デバッガはプログラム記憶位置を求めるために指定されたアドレス式を評価します。そしてその位置に格納されている値を次のように表示します。

- その記憶位置にシンボリック名がある場合には，デバッガはそのシンボルに対応するコンパイラ生成型に従って（つまり，命令または特定の型の変数として）値を編集します。
- その記憶位置にシンボリック名がない場合（したがって，対応するコンパイラ生成型もない場合）には，デバッガは省略時の設定では *longword integer* 型で値を編集します。これは省略時の設定では，EXAMINE コマンドはロングワード（4 バイト）整数値としてこれらの記憶位置の内容を表示することを意味します。

プログラム記憶位置に対応する型を変更して，その記憶位置のデータを他のデータ形式で表示できるようにする方法はいくつもあります。

- シンボリック名を持たないすべての記憶位置に対する省略時の型を変更するには，SET TYPE コマンドを使用して新しい型を指定できます。
- すべての記憶位置（シンボリック名を持つものと持たないものの両方）に対して省略時の型を変更するには，SET TYPE/OVERRIDE コマンドを使用して新しい型を指定できます。
- 1 つの EXAMINE コマンドの継続中に特定の記憶位置に現在対応している型を上書きするには，修飾子 (/ASCII:*n*，/BYTE，/TYPE=(*name*など) を使用することにより新しい型を指定できます。EXAMINE コマンドのほとんどの修飾子は型修飾子です。

デバッガは 2 進数，10 進数，16 進数，8 進数の 4 つの基数のどれか 1 つで整数データを解釈したり表示したりできます。

ほとんどの言語の場合，データの入力と表示のどちらに対しても，省略時の基数は 10 進数です。例外は BLISS と MACRO です。これらの言語での省略時の基数は 16 進数です。

EXAMINE コマンドにはデータを他の基数で表示できるようにするための基数修飾子が 4 つあります (/BINARY，/DECIMAL，/HEXADECIMAL，/OCTAL)。SET RADIX コマンドと SET RADIX/OVERRIDE コマンドを使用して，省略時の基数を変更することもできます。

EXAMINE コマンドには，型修飾子と基数修飾子だけでなく，他の用途のための修飾子があります。

- /SOURCE 修飾子を使用すれば，データではなく命令に対応する行番号，ルーチン名，ラベルまたはその他のアドレス式に対応するソース・コードの行を表示できる。

- /**[NO]LINE** 修飾子と/**[NO]SYMBOLIC** 修飾子を使用すれば、アドレス式のシンボル化を制御できる。

EXAMINE コマンドは現在の値の組み込みシンボル%CURLOC とピリオド(.)を指定されたアドレス式で示される記憶位置に設定します。論理的先行データ (%PREVLOC またはサーカンフレックス文字(^)) と論理的后続データ (%NEXTLOC) は現在の値に基づきます。

/VARIANT 修飾子は、デバッガが匿名バリエーションを発見したときに、正しい項目を表示できるようにします。

C プログラムの共用体に含まれているメンバは、個々の時点でいずれか 1 つのみが有効となっています。共用体を表示するとき、デバッガはその時点でどのメンバが有効なのかを知りません。PASCAL プログラムのバリエーション部を含んでいるレコードでは、個々の時点でいずれか 1 つのバリエーションのみが有効となっています。匿名バリエーション部を含んでいるレコードを表示するとき、デバッガはその時点でどのバリエーションが有効なのかはわからず、省略時の設定ではすべてのバリエーションを表示します。

EXAMINE コマンドの/VARIANT 修飾子を使用すると、共用体 (C プログラム) または匿名バリエーション (PASCAL プログラム) のどのメンバを表示するかを選択することができます。フォーマットは次のとおりです。

```
DBG> EXAMINE /VARIANT=variant-selector address-expression
```

```
DBG> EXAMINE /VARIANT=(variant-selector,...) address-expression
```

バリエーション選択子の variant-selector は、名前、判別式 (PASCAL のみ)、または位置を指定します。つまり、以下のいずれかになります。

- NAME = name-string
- DISCRIMINANT = expression
- POSITION = expression

/VARIANT 修飾子はゼロ個以上のバリエーション選択子のリストを取ります。省略時の設定では、バリエーション選択子なしの/VARIANT となり、すべての匿名バリエーション・リストの最初のバリエーションが表示されます。

個々のバリエーション選択子は、表示するバリエーションの名前、判別式、または位置を指定します。

デバッガはバリエーション選択子を次のように使用します。

1. アドレス式を表示しているときに匿名変数リストに遭遇すると、デバッガはバリエーション選択子を使用して、どのバリエーションを表示するかを選択します。

2. 匿名バリエント・リストに遭遇するたびに、デバッガは次のバリエント選択子を使用して、どのバリエントを表示するかを選択します。バリエント選択子がバリエント・リスト (共用体) のいずれかのバリエントと一致した場合、デバッガはそのバリエントを表示します。
3. デバッガは構造体を上から下に、深さ優先で探索するので、子は兄弟よりも前に発見されます。
4. 匿名バリエント・リストに遭遇し、それにマッチするバリエント選択子がなかった場合、デバッガは最初のバリエントを表示します。
5. バリエント選択子が匿名バリエント・リストのどのバリエントともマッチしなかった場合、デバッガはその旨を示す行を表示します。これは、判別値が判別バリエント・リストのどのバリエントともマッチしなかったときにデバッガが表示する行に似ています。次に例を示します。

[Variant Record omitted - null or illegal Tag Value: 3]

**name** は名前文字列を指定します。名前がバリエントにマッチするのは、そのバリエントが **name** によって指定された名前のフィールドを含んでいる場合です。

判別式は、対象とするバリエント部のタグ型と互換性のある型の言語式を指定します。判別式がバリエントにマッチするのは、それがバリエントの **case** ラベル・リストの中の値に評価される場合です。C と C++ の共用体は判別式を持たないので、判別式は Pascal プログラムにのみ適用されます。

**positional-selector** は、バリエント・リストの中のバリエントの数を **N** として、1 から **N** までの整数に評価される言語式を指定します。I に評価される **positional-selector** は、I 番目のバリエントを表示するように指定します。

アスタリスク (\*) をワイルドカードとして使用すると、匿名バリエント・リストのすべてのバリエントにマッチします。

これらのバリエント選択子は、いずれもすべてのバリエントにマッチするような形で使用することができます。特に、以下に示すバリエント選択子は、最初の匿名バリエント・リストのすべてのバリエントを表示するよう指定します。

```
/VAR=D=*
/VAR=N=*
/VAR=P=*
```

バリエント選択子そのものに、選択子のリストを含めることができます。たとえば、以下のコマンドはいずれも同じ意味を持ちます。

```
EXAMINE /VARIANT=(DIS=3,DIS=1,DIS=54) x
EXAMINE /VARIANT=(DIS=(3,1,54)) x
EXAMINE /VARIANT=DIS=(3,1,54) x
```

値が単純な 10 進整数である場合には、1 つの判別式または位置の値を括弧なしで指定することができます。一般的な式を使って値を指定するときには、式を括弧で囲みます。以下のコマンドのリストでは、最初の 4 つのコマンドが有効であるのに対し、最後の 3 つのコマンドは無効です。

```
EXAMINE /VARIANT=POS=3
EXAMINE /VARIANT=POS=(3)      ! parentheses unnecessary
EXAMINE /VARIANT=(POS=(3))    ! parentheses unnecessary
EXAMINE /VARIANT=(POS=3)      ! parentheses unnecessary
EXAMINE /VARIANT=(POS=foo)    ! parentheses necessary
EXAMINE /VARIANT=POS=(foo)    ! parentheses necessary
EXAMINE /VARIANT=(POS=3-1)    ! parentheses necessary
```

#### 関連コマンド

**CANCEL TYPE/OVERRIDE**  
**DEPOSIT**  
**DUMP**  
**EVALUATE**  
**SET MODE [NO]OPERANDS**  
**SET MODE [NO]SYMBOLIC**  
**(SET,SHOW,CANCEL) RADIX**  
**(SET,SHOW) TYPE**

---

#### 例

1. DBG> EXAMINE COUNT  
SUB2\COUNT: 27  
DBG>

このコマンドはモジュール SUB2 の整数変数 COUNT の値を表示します。

2. DBG> EXAMINE PART\_NUMBER  
INVENTORY\PART\_NUMBER: "LP-3592.6-84"  
DBG>

このコマンドは文字列変数 PART\_NUMBER の値を表示します。

3. DBG> EXAMINE SUB1\ARR3  
SUB1\ARR3  
(1,1): 27.01000  
(1,2): 31.01000  
(1,3): 12.48000  
(2,1): 15.08000  
(2,2): 22.30000  
(2,3): 18.73000  
DBG>

このコマンドはモジュール SUB1 の配列 ARR3 にあるすべての要素の値を表示します。ARR3 は、要素数が 2 x 3 個の実数配列です。

## EXAMINE

```
4. DBG> EXAMINE SUB1\ARR3(2,1:3)
SUB1\ARR3
      (2,1):    15.08000
      (2,2):    22.30000
      (2,3):    18.73000
DBG>
```

このコマンドは、配列 SUB1\ARR3 の配列断面の要素の値を表示します。この配列断面には、"行" 2 の"列" 1 から 3 が含まれます。

```
5. DBG> EXAMINE VALVES.INTAKE.STATUS
MONITOR\VALVES.INTAKE.STATUS: OFF
DBG>
```

このコマンドはモジュール MONITOR 内のネストされたレコードの構成要素 VALVES.INTAKE.STATUS の値を表示します。

```
6. DBG> EXAMINE/SOURCE SWAP
module MAIN
      47: procedure SWAP(X,Y: in out INTEGER) is
DBG>
```

このコマンドはルーチン SWAP が宣言されているソース行 (ルーチン SWAP の記憶位置) を表示します。

```
7. DBG> DEPOSIT/ASCII:7 WORK+20 = 'abcdefg'
DBG> EXAMINE/ASCII:7 WORK+20
DETAT\WORK+20: "abcdefg"
DBG> EXAMINE/ASCII:5 WORK+20
DETAT\WORK+20: "abcde"
DBG>
```

この例では、DEPOSIT コマンドは要素 'abcdefg' を 7 バイト長の ASCII 文字列としてシンボル WORK で示される記憶位置から 20 バイト先の記憶位置に格納します。最初の EXAMINE コマンドはその記憶位置にある要素の値を 7 バイト長の ASCII 文字列 (abcdefg) として表示します。2 番目の EXAMINE コマンドはその記憶位置の要素の値を 5 バイト長の ASCII 文字列 (abcde) として表示します。

```
8. DBG> EXAMINE/OPERANDS=FULL .0\%PC
X\X$START+0C:  MOVL    B^04(R4),R7
               B^04(R4)  R4 contains X\X$START\M (address 00001054),
                       B^04(00001054) evaluates to X\X$START\K
                       (address 00001058), which contains 00000016
               R7       R7 contains 00000000
DBG>
```

VAX システムでは、このコマンドは現在の PC 値での命令 (MOVL) を表示します。/OPERANDS=FULL を使用すると、オペランド情報の最大レベルが表示されます。

```

9.  DBG> SET RADIX HEXADECIMAL
    DBG> EVALUATE/ADDRESS WORKDATA
    0000086F
    DBG> EXAMINE/SYMBOLIC 0000086F
    MOD3\WORKDATA: 03020100
    DBG> EXAMINE/NOSYMBOLIC 0000086F
    0000086F: 03020100
    DBG>

```

この例では、EVALUATE/ADDRESS コマンドは変数 WORKDATA のメモリ・アドレスが 16 進数の 0000086F であることを示します。2 つの EXAMINE コマンドはアドレスを WORKDATA にシンボル化するかどうかを制御するために、/[NO]SYMBOLIC を使用してそのアドレスに含まれている値を表示します。

```

10. DBG> EXAMINE/HEX FIDBLK
    FDEX1$MAIN\FIDBLK
    (1):      00000008
    (2):      00000100
    (3):      000000AB
    DBG>

```

このコマンドは 16 進基数で配列変数 FIDBLK の値を表示します。

```

11. DBG> EXAMINE/DECIMAL/WORD NEWDATA:NEWDATA+6
    SUB2\NEWDATA: 256
    SUB2\NEWDATA+2: 770
    SUB2\NEWDATA+4: 1284
    SUB2\NEWDATA+6: 1798
    DBG>

```

このコマンドは NEWDATA から NEWDATA + 6 バイトで示される記憶位置の範囲内にあるワード整数要素 (2 バイト要素) の値を 10 進基数で表示します。

```

12. DBG> EXAMINE/TASK SORT INPUT
    MOD3\SORT_INPUT: %TASK 12
    DBG>

```

このコマンドは SORT\_INPUT という名前のタスク・オブジェクトのタスク ID を表示します。

```

13. DBG> EXAMINE /VARIANT=(NAME=m,DIS=4,POS=1) x

```

このコマンドは、最初に遭遇した匿名バリエント・リストについては "m" という名前のフィールドを含んでいるバリエント部を表示し、2 番目の匿名バリエント・リストについては判別式の値が 4 の部分を表示し、3 番目の匿名バリエント・リストについては最初のバリエント部を表示します。

---

## EXIT

デバッグ・セッションを終了するか、またはマルチプロセス・プログラムの 1 つまたは複数のプロセスを終了し、アプリケーション宣言終了ハンドラを実行できるようにします。コマンド・プロシージャまたは DO 句の中で使用し、プロセスを全く指定しないと、その時点でそのコマンド・プロシージャまたは DO 句が終了されます。

---

### フォーマット

EXIT *[process-spec[, ... ]]*

---

### パラメータ

*process-spec*

現在デバッガの制御下にあるプロセスを指定します。次のいずれかの形式で指定します。

*[%PROCESS\_NAME] process-name*

スペースや小文字を含まないプロセス名。プロセス名にはワイルドカード文字(\*)を含めることができる。

*[%PROCESS\_NAME] "process-name"*

スペースまたは小文字を含むプロセス名。二重引用符(")の代わりに、一重引用符(') 使用することもできる。

*%PROCESS\_PID process\_id*

プロセス識別子 (PID, 16 進数)。

*[%PROCESS\_NUMBER] process-number*  
(または *%PROC process-number*)

デバッガの制御下に入ったときにプロセスに割り当てられた番号。新しい番号は、1 から順番に各プロセスに割り当てられる。EXIT コマンドまたは QUIT コマンドによってプロセスが終了した場合、そのデバッグ・セッション中にその番号が再割り当てされることがある。プロセス番号は SHOW PROCESS コマンドの実行で表示される。プロセスは、組み込みシンボル *%PREVIOUS\_PROCESS* および *%NEXT\_PROCESS* によってインデックスづけできるように、循環リスト内に順序づけされる。

*process-set-name*

DEFINE/PROCESS\_SET コマンドで定義された、プロセスのグループを表すシンボル。

*%NEXT\_PROCESS*

デバッガの循環プロセス・リスト中で可視プロセスの次のプロセス。

*%PREVIOUS\_PROCESS*

デバッガの循環プロセス・リスト中で可視プロセスの前のプロセス。



%VISIBLE\_PROCESS

シンボル、レジスタ値、ルーチン呼び出し、ブレークポイントなどの検索時に現在のコンテキストになっているスタック、レジスタ・セット、およびイメージを持つプロセス。

すべてのプロセスを指定するためにワイルドカード文字のアスタリスク(\*)を使用することもできます。

---

## 説明

EXIT コマンドはプログラムを実行するために使用できる 4 つのデバッグ・コマンドのうちの 1 つです (他の 3 つは CALL, GO, STEP です)。

### デバッグ・セッションの終了

デバッグ・セッションを終了するには、パラメータを全く指定しないでデバッグ・プロンプトに対し EXIT コマンドを入力します。これによりセッションが順序正しく終了されます。つまり、プログラムのアプリケーション宣言終了ハンドラ (もしあれば) が実行され、次にデバッグの終了ハンドラが実行され (ログ・ファイルをクローズし、画面とキーパッドの状態を復元するなど)、制御がコマンド・インタプリタに戻ります。そのあとは DCL コマンドの DEBUG や CONTINUE を入力してもプログラムのデバッグを続けることはできません (デバッグを再起動する必要があります)。

EXIT は任意のアプリケーション宣言終了ハンドラを実行するので、該当する終了ハンドラにブレークポイントを設定することができます。設定したブレークポイントは EXIT と入力すると検出されます。したがって、EXIT を使用して終了ハンドラをデバッグできます。

アプリケーション宣言終了ハンドラを実行しないでデバッグ・セッションを終了するには、EXIT の代わりに QUIT コマンドを使用します。

### コマンド・プロシージャや DO 句での EXIT コマンドの使用

デバッグがコマンド・プロシージャの中で (パラメータを全く指定しないで) EXIT コマンドを実行すると、制御はそのコマンド・プロシージャを起動したコマンド・ストリームに戻ります。コマンド・ストリームは、端末、外側 (当該コマンド・プロシージャを含んでいる側) のコマンド・プロシージャ、コマンドまたは画面表示定義での DO 句とすることができます。たとえば、コマンド・プロシージャが DO 句の中から起動された場合には、制御はその DO 句に戻り、そこでデバッグは (コマンド・シーケンスの中にあれば) 次のコマンドを実行します。

デバッグは、DO 句の中で (パラメータを全く指定しないで) EXIT コマンドを実行すると、その句にある残りのコマンドは無視し、プロンプトを表示します。

### 指定プロセスの終了

マルチプロセス・プログラムをデバッグする場合，EXIT コマンドを使用すればデバッグ・セッションを終了せずに，指定したプロセスを終了できます。プロンプトに対してEXIT コマンドを入力するか，それともコマンド・プロシージャまたはDO 句の中でEXIT コマンドを使用するかにかかわらず，同じ方法と動作が適用されます。

1 つまたは複数のプロセスを終了するには，それらのプロセスをパラメータとして指定してEXIT コマンドを入力します。この結果，指定されたプロセスのイメージが順序正しく終了され，それらのイメージに対応するアプリケーション宣言終了ハンドラが実行されます。その後は，指定したプロセスはSHOW PROCESS/ALL コマンドを実行しても表示されなくなります。SET PROCESS コマンドの結果，指定したプロセスの中に保留状態だったものがある場合，保留状態は無視されます。

指定したプロセスが終了処理を開始すると，保留されていない未指定のプロセスが実行を開始します。実行が開始されると，その実行がどのように継続されるかはユーザがSET MODE [NO]INTERRUPT コマンドを入力したかどうかによって異なります。省略時の設定 (SET MODE INTERRUPT) では，実行はプロセスのどれかで中断されるまで継続します。中断された時点で，イメージを実行していた他のどれかのプロセスで割り込みがかり，デバッガが入力を促します。

アプリケーション宣言終了ハンドラを実行したり，別の方法で実行を開始したりしないで指定のプロセスを終了するには，EXIT コマンドではなくQUIT コマンドを使用します。

### 関連コマンド

DISCONNECT  
@ (実行プロシージャ)  
Ctrl/C  
Ctrl/Y  
Ctrl/Z  
QUIT  
RERUN  
RUN  
SET ABORT\_KEY  
SET MODE [NO]INTERRUPT  
SET PROCESS

---

## 例

1. DBG> EXIT  
\$

このコマンドはデバッグ・セッションを終了し，DCL レベルに戻ります。

```
2. all> EXIT %NEXT_PROCESS, JONES_3, %PROC 5  
all>
```

このコマンドを実行すると、マルチプロセス・プログラムの3つのプロセスが順序正しく終了します。つまり、プロセス・リスト内の可視プロセスの次のプロセス、プロセス JONES\_3、プロセス 5 が終了します。指定されたプロセスが終了したあと、制御はデバッガに戻ります。

---

## EXITLOOP

中にこのコマンドが入っている 1 つまたは複数の FOR , REPEAT または WHILE のループを終了します。

---

### フォーマット

EXITLOOP *[integer]*

---

### パラメータ

integer

終了するネストされたループの数を指定する 10 進数の整数。省略時の設定は 1 です。

---

### 説明

1 つまたは複数の FOR , REPEAT または WHILE のループを終了するにはこれらのループの中で EXITLOOP コマンドを使用します。

関連コマンド

FOR  
REPEAT  
WHILE

---

### 例

```
DBG> WHILE 1 DO (STEP; IF X .GT. 3 THEN EXITLOOP)
```

WHILE 1 コマンドは繰り返すたびに STEP コマンドを実行する無限ループを作成します。STEP が 1 回終わるたびに、X の値がテストされます。X が 3 より大きい場合には、EXITLOOP コマンドはループを終了します (Fortran の例)。

---

## EXPAND

ディスプレイに対応するウィンドウを拡大または縮小します。

---

### 注意

---

このコマンドは、デバッガへの HP DECwindows Motif for OpenVMS ユーザ・インタフェースでは使用できません。

---



---

## フォーマット

EXPAND *[display-name[, ... ]]*

---

## パラメータ

display-name

拡大または縮小するディスプレイを指定します。次のいずれかを指定できます。

- 定義済みディスプレイ

SRC  
OUT  
PROMPT  
INST  
REG  
FREG (Alpha および I64 のみ)  
IREG

- DISPLAY コマンドで作成したディスプレイ
- ディスプレイの組み込みシンボル

%CURDISP  
%CURSCROLL  
%NEXTDISP  
%NEXTINST  
%NEXTOUTPUT  
%NEXTSCROLL  
%NEXTSOURCE

ディスプレイを指定しないと、SELECT コマンドの設定した現在のスクロール・ディスプレイが選択されます。

---

## 修飾子

/DOWN[:n]

ディスプレイの下の境界を $n$ 行分だけ下に ( $n$ が正の値の場合)、または $n$ 行分だけ上に ( $n$ が負の値の場合) 移動します。 $n$ を省略すると、境界は 1 行下に移動されます。

/LEFT[:n]

ディスプレイの左の境界を $n$ 行分だけ左に ( $n$ が正の値の場合)、または $n$ 行分だけ右に ( $n$ が負の値の場合) 移動します。 $n$ を省略すると、境界は左に 1 行移動されます。

/RIGHT[:n]

ディスプレイの右の境界を $n$ 行分だけ右に ( $n$ が正の値の場合)、または $n$ 行分だけ左に ( $n$ が負の値の場合) 移動します。 $n$ を省略すると、境界は右に 1 行移動されます。

/UP[:n]

ディスプレイの上の境界を $n$ 行分だけ上に ( $n$ が正の値の場合)、または $n$ 行分だけ下に ( $n$ が負の値の場合) 移動します。 $n$ を省略すると、境界は 1 行上に移動されます。

---

## 説明

少なくとも修飾子を 1 つ指定しなくてはなりません。

EXPAND コマンドは指定された修飾子 (/UP[:n]、/DOWN[:n]、RIGHT[:n]、/LEFT[:n]) に従ってディスプレイ・ウィンドウの 1 つまたは複数の境界を移動します。

EXPAND コマンドはディスプレイ・ペーストボード上でのディスプレイの順番には影響を及ぼしません。EXPAND コマンドを実行すると、指定したディスプレイで他のディスプレイが見えなくなったり、見えるようになったりします。また、指定したディスプレイが他のディスプレイによって部分的または全体的に見えなくなることもあります。どのようになるかはディスプレイ相互間の順序によります。

PROMPT ディスプレイを除き、どのディスプレイもそれが見えなくなるポイント (そのポイントで "removed" とマークされる) まで縮小できます。そこまで縮小されると、今度はそのポイントから拡大できます。見えなくなるポイントまでディスプレイを縮小すると、そのディスプレイに対して選択された属性を失うことになります。PROMPT ディスプレイは水平方向には縮小も拡大もできませんが、垂直方向には 2 行分縮小できます。

ウィンドウの境界は上方向には画面の辺までしか拡大できません。ウィンドウの左と上の境界はディスプレイのそれぞれ左辺と上辺を超えて拡大することはできません。右の境界はディスプレイの左辺から 255 列まで拡大できます。ソース・ディスプレイまたは機械語命令ディスプレイの下の境界は、下方向にはディスプレイの下辺まで (ソース・モジュールまたはルーチンの命令の端) しか拡大できません。レジスタ・ディスプレイはそのフル・サイズを超えて拡大することはできません。

EXPAND コマンドに対応するキー定義のリストについては、ヘルプ・トピックの `Keypad_Definitions_CI` を参照してください。また、現在のキー定義を調べるには `SHOW KEY` コマンドを使用してください。

#### 関連コマンド

DISPLAY  
MOVE  
SELECT/SCROLL  
(SET,SHOW) TERMINAL

---

#### 例

1. `DBG> EXPAND/RIGHT:6`  
このコマンドは現在のスクロール・ディスプレイの右の境界を右に 6 列移動します。
2. `DBG> EXPAND/UP/RIGHT:-12 OUT2`  
このコマンドは OUT2 ディスプレイの上の境界を上 1 行、右の境界を左に 12 列移動します。
3. `DBG> EXPAND/DOWN:99 SRC`  
このコマンドは SRC ディスプレイの下境界を画面の下辺まで降ろします。

---

## EXTRACT

画面表示の内容をファイルに保存するか、または現在の画面状態をあとで再構成するために必要なコマンドのすべてを盛り込んだデバッガ・コマンド・プロシージャを作成します。

---

### 注意

このコマンドは、デバッガへの HP DECwindows Motif for OpenVMS ユーザ・インタフェースでは使用できません。

---

---

## フォーマット

EXTRACT *[display-name[, ... ]]* *[file-spec]*

---

## パラメータ

*display-name*

書き出すディスプレイを指定します。次のいずれかを指定できます。

- 定義済みディスプレイ

SRC

OUT

PROMPT

INST

REG

FREG (Alpha および I64 のみ)

IREG

- DISPLAY コマンドで作成したディスプレイ

ディスプレイ名のワイルドカード文字のアスタリスク(\*)は使用できません。/ALL 修飾子を指定する場合、ディスプレイ名は指定できません。

*file-spec*

情報を書き込むファイルを指定します。論理名を指定できます。

/SCREEN\_LAYOUT を指定する場合、ファイルの省略時の指定は SYSSDISK:[J]DBGS CREEN.COM です。指定しない場合、省略時の指定は SYSSDISK:[J]DEBUG.TXT です。



---

## 修飾子

### /ALL

すべてのディスプレイを書き出します。この修飾子を指定する場合、/SCREEN\_LAYOUT は指定できません。

### /APPEND

新しいファイルを作成するのではなく、ファイルの終わりに情報を追加します。省略時の設定では、新しいファイルが作成されます。この修飾子を指定する場合、/SCREEN\_LAYOUT は指定できません。

### /SCREEN\_LAYOUT

画面の現在の状態を記述するデバッガ・コマンドを含むファイルを書き込みます。この情報には画面の幅と高さ、メッセージのラップの設定、既存のすべてのディスプレイの位置、表示対象、ディスプレイ属性が含まれます。あとで画面を再構成するためにこのファイルを実行プロシージャのコマンド(@) で実行することができます。

---

## 説明

ディスプレイの内容をファイルに保存するために EXTRACT コマンドを使用すると、そのディスプレイのメモリ・バッファに現在格納されている行だけ (DISPLAY コマンドの /SIZE 修飾子で求められるとおり) がファイルに書き込まれます。

PROMPT ディスプレイをファイルに書き出すことはできません。

### 関連コマンド

DISPLAY  
SAVE

---

## 例

1. DBG> EXTRACT SRC

このコマンドは SYS\$DISK:[ ]DEBUG.TXT ファイルに SRC ディスプレイのすべての行を書き込みます。

2. DBG> EXTRACT/APPEND OUT [JONES.WORK]MYFILE

このコマンドは [JONES.WORK]MYFILE.TXT ファイルの終わりに OUT ディスプレイのすべての行を追加します。

3. DBG> EXTRACT/SCREEN\_LAYOUT

このコマンドは SYS\$DISK:[ ]DBGSCREEN.COM ファイルに画面を再構成するために必要なデバッガ・コマンドを書き込みます。

---

# FOR

変数を増分しながら一連のコマンドを指定された回数実行します。

---

## フォーマット

```
FOR  name=expression1 TO expression2 [BY expression3]  
DO  (command[; . . . ])
```

---

## パラメータ

name

カウント変数の名前を指定します。

expression1

整数型または列挙型の値を指定します。expression1とexpression2のパラメータは同じ型のものでなければなりません。

expression2

整数型または列挙型の値を指定します。expression1とexpression2のパラメータは同じ型のものでなければなりません。

expression3

整数を指定します。

command

デバッガ・コマンドを指定します。複数のコマンドを指定する場合は、それぞれのコマンドをセミコロンで区切らなければなりません。各コマンドを実行すると、デバッガはそのコマンド内の式の構文をチェックして、評価します。

---

## 説明

FOR コマンドの動作は expression3 パラメータの値によって決まります。次の表をご覧ください。

expression3	FOR コマンドの動作
正の値	nameパラメータは、expression2 の値より大きくなるまで、expression1 の値から expression3 の値ずつ増分される。
負の値	nameパラメータは、expression2 の値より小さくなるまで、expression1 の値から expression3 の値ずつ減分される。
0	デバッガはエラー・メッセージを返す。

---

expression3	FOR コマンドの動作
Omitted	デバッガは、値が +1 であるものとみなす。

---

#### 関連コマンド

EXITLOOP

REPEAT

WHILE

---

#### 例

1. DBG> FOR I = 10 TO 1 BY -1 DO (EXAMINE A(I))  
このコマンドは配列を後ろから前に調べます。
2. DBG> FOR I = 1 TO 10 DO (DEPOSIT A(I) = 0)  
このコマンドは配列を 0 に初期化します。

---

## GO

プログラムの実行を開始または再開します。

---

### フォーマット

GO *[address-expression]*

---

### パラメータ

address-expression

プログラムの実行をアドレス式で示される記憶位置から再開することを指定します。アドレス式を指定しないと、実行は中断されたポイントから、またはデバッガ起動の場合にはイメージ遷移アドレスから再開されます。

---

### 説明

GO コマンドはプログラムの実行を開始するか、または実行が現在中断されているポイントから再開します。GO コマンドはプログラムを実行するために使用できる 4 つのデバッガ・コマンドの 1 つです (他の 3 つは CALL, EXIT, STEP です)。

GO コマンドでアドレス式を指定すると、それはプログラムの通常の制御の流れを変えるため、思いがけない結果が生じる可能性があります。たとえば、デバッグ・セッションの最中に GO %LINE 1 というコマンドを入力すればプログラムの先頭から実行を再開できますが、そのプログラムはすでに実行されているため、変数によっては、最初に同じプログラムを実行したときとは異なった内容に初期化されるものもあります。

例外ブレークポイントを検出すると (SET BREAK/EXCEPTION コマンドまたは STEP/EXCEPTION コマンドの結果)、実行はアプリケーション宣言条件ハンドラが起動される前に中断されます。そのあと GO コマンドで実行を再開すると、動作は次のようになります。

- 現在の記憶位置から実行を再開するために GO コマンドを入力すると、デバッガは例外を再度シグナル通知する。これにより、必要であればどのアプリケーション宣言ハンドラが次にその例外を処理するのかがわかる。
- 現在の記憶位置以外の記憶位置から実行を再開するために GO コマンドを入力すると、その例外に対してはアプリケーション宣言ハンドラを実行できなくなる。

マルチプロセス・プログラムをデバッグする場合，GO コマンドは現在のプロセス・セットのコンテキストで実行されます。また，マルチプロセス・プログラムをデバッグするときに，プロセス内で実行がどのように続行されるかは，入力したコマンドが SET MODE [NO]INTERRUPT コマンドか，それとも SET MODE [NO]WAIT コマンドかによって異なります。省略時の設定 (SET MODE NOINTERRUPT) では，1 つのプロセスが停止しても，デバッガは他のプロセスに関しては何のアクションも行いません。また，やはり省略時の設定 (SET MODE WAIT) では，デバッガは現在のプロセス・セットの中のすべてのプロセスが停止するのを待ってから，新たなコマンドを求めるプロンプトを表示します。詳細は，『OpenVMS デバッグ説明書』を参照してください。

#### 関連コマンド

```
CALL
EXIT
RERUN
SET BREAK
SET MODE [NO]INTERRUPT
SET MODE [NO]WAIT
SET PROCESS
SET STEP
SET TRACE
SET WATCH
STEP
WAIT
```

---

#### 例

```
1.  DBG> GO
      . . .
      'Normal successful completion'
      DBG>
```

このコマンドはプログラムの実行を開始します。その実行はやがて正常に完了します。

```
2.  DBG> SET BREAK RESTORE
      DBG> GO      ! 実行を開始
      . . .

      break at routine INVENTORY\RESTORE
      137: procedure RESTORE;
      DBG> GO      ! 実行を再開
      . . .
```

この例では，SET BREAK コマンドはブレークポイントをルーチン RESTORE に設定します。最初の GO コマンドはプログラムの実行を開始します。それはや

## GO

がルーチン RESTORE のブレークポイントで中断されます。2 番目の GO コマンドはブレークポイントから実行を再開します。

### 3. DBG> GO %LINE 42

このコマンドは実行が現在中断されているモジュールの 42 行目で実行を再開します。

---

## HELP

デバッガ・コマンドと選択されたトピックに関するオンライン・ヘルプを表示します。

---

### 注意

このコマンドは、デバッガへの HP DECwindows Motif for OpenVMS ユーザ・インタフェースでは使用できません。コマンドに関するヘルプは DECwindows デバッガ・ウィンドウの「Help」メニューから入手できます。

---

---

## フォーマット

HELP *topic* [*subtopic* [ . . . ]]

---

## パラメータ

*topic*

ヘルプ情報を得たいデバッガ・コマンドまたはトピックの名前を指定します。ワイルドカード文字のアスタリスク(\*)を単独で指定することも、名前の中に指定することもできます。

*subtopic*

さらに情報を必要とするサブトピック、修飾子またはパラメータを指定します。ワイルドカード文字のアスタリスク(\*)を単独で指定することも、名前の中に指定することもできます。

---

## 説明

デバッガのオンライン・ヘルプ機能は、コマンド記述、コマンド形式、コマンドで指定できるパラメータの説明、コマンドで指定できる修飾子の説明のほか、デバッガ・コマンドについて次の情報を提供します。

特定の修飾子またはパラメータに関する情報を獲得するには、それをサブトピックとして指定します。すべての修飾子に関する情報が必要な場合には、"qualifier"をサブトピックとして指定します。すべてのパラメータに関する情報が必要な場合には、"parameter"をサブトピックとして指定します。1つのコマンドに関連するすべてのパラメータ、修飾子、その他のサブトピックについての情報が必要な場合には、アスタリスク(\*)をサブトピックとして指定します。

## HELP

コマンドに関するヘルプ以外にも、画面機能、キーパッド・モードなどのさまざまなトピックについてオンライン・ヘルプを獲得できます。HELP と入力すると、コマンドといっしょにトピック・キーワードがリストされます。

デバッガの今回のリリースでの新機能の概略については、HELP New\_Features をタイプして参照してください。

定義済みキーパッド・キーの機能についてのヘルプは、ヘルプ・トピックの Keypad\_Definitions\_CI を参照してください。また、現在のキー定義を調べるには SHOW KEY コマンドを使用してください。

---

### 例

```
DBG> HELP GO
```

このコマンドは GO コマンドのヘルプを表示します。



---

## IF

言語式 (ブール式) が真として評価された場合に一連のコマンドを実行します。

---

### フォーマット

```
IF Boolean-expression THEN (command; . . . )  
  [ELSE (command; . . . )]
```

---

### パラメータ

*Boolean-expression*

現在設定されている言語でブール値 (真または偽) として評価される言語式を指定します。

*command*

デバッガ・コマンドを指定します。複数のコマンドを指定する場合には、それぞれをセミコロン(;)で区切らなければなりません。

---

### 説明

IF コマンドはブール式を評価します。値が真 (現在の言語での定義に従う) である場合には、THEN 句のコマンド・リストが実行されます。式が偽である場合には、ELSE 句のコマンド・リストがある場合実行されます。

関連コマンド

```
EXITLOOP  
FOR  
REPEAT  
WHILE
```

---

### 例

```
DBG> SET BREAK R DO (IF X .LT. 5 THEN (GO) ELSE (EXAMINE X))
```

このコマンドを実行すると、デバッガはXの値が5未満であれば、Rの記憶位置 (ブレークポイント) でプログラムの実行を中断します (Fortran の例)。Xの値が5以上の場合には、その値が表示されます。

---

## MONITOR

プログラム変数または言語式の現在の値を HP DECwindows Motif for OpenVMS ユーザ・インタフェースのモニタ・ビューに表示します。

---

### 注意

HP DECwindows Motif for OpenVMS ユーザ・インタフェースが必要です。

---

---

## フォーマット

MONITOR *expression*

---

## パラメータ

*expression*

モニタの対象とする要素を指定します。高級言語を使用する場合、これは通常、変数の名前です。現在、MONITOR は複合式 (演算子を含む言語式) は処理しません。

集合体変数 (配列またはレコード構造などのような複合データ構造) の名前を指定すると、モニタ・ビューはその変数の値として“Aggregate”と表示します。そのあと変数名をダブルクリックすれば、すべての要素の値を獲得できます。(『デバッグ説明書』を参照。)

個々の配列要素、配列断面またはレコードの構成要素を指定するには、現在の言語の構文に従います。

---

## 修飾子

/ASCIC

/AC

モニタする個々の要素を、1 バイトのカウント・フィールドの後ろに続く、このカウント・フィールドにより長さを指定された ASCII 文字列と解釈します。

/ASCID

/AD

モニタする個々の要素を ASCII 文字列を指す文字列ディスクリプタのアドレスと解釈します。ディスクリプタの CLASS フィールドと DTYPE フィールドはモニタされませんが、LENGTH フィールドと POINTER フィールドが ASCII 文字列の文字長とアドレスを与えます。そのあと文字列が表示されます。

/ASCII:n

モニタする個々の要素を長さ  $n$  バイト ( $n$  文字) の ASCII 文字列と解釈し、表示します。 $n$  を省略すると、デバッガはアドレス式の型をもとに長さを求めようとします。

/ASCIIW

/AW

モニタする個々の要素を、2 バイトのカウント・フィールドの後ろに続く、このカウント・フィールドにより長さを指定された ASCII 文字列と解釈します。それから文字列が表示されます。

/ASCIIZ

/AZ

モニタする個々の値を最後部に 0 のある ASCII 文字列と解釈します。最後部の 1 バイト分の 0 は文字列の終了を示します。それから文字列が表示されます。

/BINARY

モニタする個々の値を 2 進整数として表示します。

/BYTE

モニタする個々の値をバイト整数型 (1 バイト長) で表示します。

/D\_FLOAT

(VAX のみ) モニタする個々の値を D 浮動小数点型 (8 バイト長) で表示します。

/DATE\_TIME

モニタする個々の値を日付と時刻の内部表現を含むクォードワード整数 (8 バイト長) と解釈します。値は *dd-mmm-yyyy hh:mm:ss.cc* の形式で表示します。

/DECIMAL

モニタする個々の値を 10 進整数として表示します。

/DEFAULT

モニタする個々の値を省略時の基数で表示します。

/EXTENDED\_FLOAT

(Alpha および I64 のみ) モニタする個々の値を IEEE の X 浮動小数点型 (16 バイト長) で表示します。

/FLOAT

VAX プロセッサでは、モニタする個々の値を F 浮動小数点型 (4 バイト長) で表示します。

Alpha プロセッサでは、モニタする個々の値を IEEE T 浮動小数点型 (倍精度, 8 バイト長) で表示します。

/F\_FLOAT

(VAX のみ) モニタする個々の値を F 浮動小数点型 (4 バイト長) で表示します。

/G\_FLOAT

モニタする個々の値を G 浮動小数点型 (8 バイト長) で表示します。

**/H\_FLOAT**

(VAX のみ) モニタする個々の値を H 浮動小数点型 (16 バイト長) で表示します。

**/HEXADECIMAL**

モニタする個々の値を 16 進整数で表示します。

**/INSTRUCTION**

モニタする個々の値をアセンブリ言語命令 (命令の長さは命令オペランドの数とアドレッシング・モードの種類によって変化) として表示します。/OPERANDS 修飾子も参照してください。

**/INT**

/LONGWORD 修飾子と同じ。

**/LONG\_FLOAT**

(Alpha および I64) モニタする個々の値を IEEE S 浮動小数点型 (単精度, 4 バイト長) で表示します。

**/LONG\_LONG\_FLOAT**

(Alpha および I64) モニタする個々の値を IEEE T 浮動小数点型 (倍精度, 8 バイト長) で表示します。

**/LONGWORD****/INT****/LONG**

モニタする個々の値をロングワード整数型 (4 バイト長) で表示します。これがコンパイラ生成型を持たないプログラム記憶位置の場合の省略時の型です。

**/OCTAL**

モニタする個々の値を 8 進整数で表示します。

**/OCTAWORD**

モニタする個々の値をオクタワード整数型 (16 バイト長) で表示します。

**/QUADWORD**

モニタする個々の値をクォドワード整数型 (8 バイト長) で表示します。

**/REMOVE**

モニタ・ビューから指定されたアドレス式でモニタされた 1 つまたは複数の項目を削除します。

**/SHORT**

/WORD 修飾子と同じ。

**/TASK**

タスキング (マルチスレッド) プログラムの場合に指定できます。モニタする個々の値をタスク (スレッド) オブジェクトとして解釈し, そのタスク・オブジェクトのタスク値 (名前またはタスク ID) を表示します。タスク・オブジェクトをモニタするときには, プログラミング言語に組み込みタスキング・サービスが備っていない場合だけ/TASK を使用してください。

/WORD  
/SHORT

モニタする個々の要素をワード整数型 (2 バイト長) で表示します。

---

## 説明

コマンドの出力はモニタ・ビューに出力されるため、MONITOR コマンドはデバッガの HP DECwindows Motif for OpenVMS ユーザ・インタフェースがある場合にだけ使用できます。コマンド・インタフェースがある場合には、通常、代わりに EVALUATE、EXAMINE、SET WATCH の各コマンドを使用します。

MONITOR コマンドは次のことを行います。

1. モニタ・ビューがそれまでの MONITOR コマンドでまだ表示されていない場合には、モニタ・ビューを表示する。
2. 指定された変数または式の名前とその現在の値をモニタ・ビューに配置する。

ユーザがモニタしている変数または記憶位置の値が変更されたかどうかにかかわらず、プログラムはデバッガから制御が戻ると、モニタ・ビューを更新します。対称的に、ウォッチポイントはウォッチされている変数の値が変化すると実行を停止します。

モニタ・ビューと MONITOR コマンドについての詳しい説明は、『デバッガ説明書』を参照してください。

### 関連コマンド

DEPOSIT  
EVALUATE  
EXAMINE  
SET WATCH

---

## 例

DBG> MONITOR COUNT

このコマンドはデバッガの HP DECwindows Motif for OpenVMS ユーザ・インタフェースのモニタ・ビューに変数 COUNT の名前と現在の値を表示します。プログラムからデバッガに制御が戻ると、値は更新されます。

---

## MOVE

画面ディスプレイを画面上で垂直または水平に移動します。

---

### 注意

このコマンドは、デバッガへの HP DECwindows Motif for OpenVMS ユーザ・インタフェースには使用できません。

---

---

## フォーマット

MOVE *[display-name[, ... ]]*

---

## パラメータ

display-name

移動するディスプレイを指定します。次のいずれかを指定できます。

- 定義済みディスプレイ
  - SRC
  - OUT
  - PROMPT
  - INST
  - REG
  - FREG (Alpha および I64 のみ)
  - IREG
- DISPLAY コマンドで作成したディスプレイ
- ディスプレイの組み込みシンボル

- %CURDISP
  - %CURSCROLL
  - %NEXTDISP
  - %NEXTINST
  - %NEXTOUTPUT
  - %NEXTSCROLL
  - %NEXTSOURCE

ディスプレイを指定しないと、SELECT コマンドで設定した現在のスクロール・ディスプレイが選択されます。

---

## 修飾子

/DOWN[:*n*]

ディスプレイを*n*行分だけ下に (*n*が正の値の場合)、または*n*行分だけ上に (*n*が負の値の場合) 移動します。*n*を省略すると、表示は1行下に移動されます。

/LEFT[:*n*]

ディスプレイを*n*行分だけ左に (*n*が正の値の場合)、または*n*行分だけ右に (*n*が負の値の場合) 移動します。*n*を省略すると、表示は左に1行移動されます。

/RIGHT[:*n*]

ディスプレイを*n*行分だけ右に (*n*が正の値の場合)、または*n*行分だけ左に (*n*が負の値の場合) 移動します。*n*を省略すると、表示は右に1行移動されます。

/UP[:*n*]

修飾子は少なくとも1つ指定しなければなりません。

ディスプレイを*n*行分だけ上に (*n*が正の値の場合)、または*n*行分だけ下に (*n*が負の値の場合) 移動します。*n*を省略すると、ディスプレイは1行上に移動されます。

---

## 説明

少なくとも1つの修飾子を指定しなくてはなりません。

指定された表示ごとに、MOVE コマンドはウィンドウ内のテキストの相対位置を維持したまま、画面上の他の場所に同じ寸法のウィンドウを作成し、それにディスプレイをマップします。

MOVE コマンドはディスプレイ・ペーストボード上でのディスプレイの順番は変更しません。MOVE コマンドを実行すると、指定したディスプレイによって他のディスプレイが見えなくなったり、見えるようになっていきます。また、指定したディスプレイが他のディスプレイによって部分的または全体的に見えなくなることもあります。どのようになるかはディスプレイ相互間の順序によります。

ディスプレイは上には画面の上辺までしか移動できません。

MOVE コマンドに対応するキーパッド・キー定義のリストについては、ヘルプ・トピック Keypad\_Definitions\_CI を参照してください。また、現在のキー定義を調べるには SHOW KEY コマンドを使用してください。

### 関連コマンド

DISPLAY  
EXPAND  
SELECT/SCROLL

## MOVE

(SET,SHOW) TERMINAL

---

### 例

1. DBG> MOVE/LEFT

このコマンドは現在のスクロール・ディスプレイを左に 1 列移動します。

2. DBG> MOVE/UP:3/RIGHT:5 NEW\_OUT

このコマンドはNEW\_OUTディスプレイを上 3 行、右 5 列移動します。



---

## PTHREAD

実行のために、POSIX Threads デバッガにコマンドを渡します。

---

### 注意

---

このコマンドは、イベント機能が THREADS であり、プログラムが POSIX Threads 3.13 以降を実行している場合のみ有効です。

---

---

## フォーマット

PTHREAD *command*

---

## パラメータ

*command*

POSIX Threads デバッグ・コマンドです。

---

## 説明

実行のために、POSIX Threads デバッガにコマンドを渡します。実行結果は、「Command」ビューに表示されます。POSIX Threads デバッガ・コマンドが完了すると、OpenVMS デバッガに制御が戻ります。PTHREAD HELP と入力すると、POSIX Threads デバッガ・コマンドのヘルプを参照することができます。

POSIX Threads デバッガの使用方法的詳細については、『Guide to the POSIX Threads Library』を参照してください。

関連コマンドには、次のものがあります。

- SET EVENT FACILITY
- SET TASK | THREAD
- SHOW EVENT FACILITY
- SHOW TASK | THREAD

---

例

```
DBG_1> PTHREAD HELP
conditions [-afhwqrs] [-N <n>] [id]...: list condition variables
exit: exit from DECthreads debugger
help [topic]: display help information
keys [-v] [-N <n>] [id]...: list keys
mutexes [-afhilqrs] [-N <n>] [id]...: list mutexes
quit: exit from DECthreads debugger
show [-csuv]: show stuff
squeue [-c <n>] [-fhq] [-t <t>
] [a]: format queue
stacks [-fs] [sp]...: list stacks
system: show system information
threads [-l] [-N <n>] [-abcdfhklmnor] [-s
<v>] [-tz] [id]...: list threads
tset [-chna] [-s <v>] <id>
: set state of thread
versions: display versions
write <st>: write a string
All keywords may be abbreviated: if the abbreviation is ambiguous,
the first match will be used. For more help, type 'help <topic>'.
DBG_1>
```

このコマンドでは、POSIX Threads デバッガのヘルプ・ファイルが表示された後、OpenVMS デバッガに制御が戻っています。POSIX Threads デバッガのヘルプ・トピックについての情報を表示するには、PTHREAD HELP topic と入力します。

---

## QUIT

EXIT と同様にデバッグ・セッションを終了するか、またはマルチプロセス・プログラムの 1 つまたは複数のプロセスを終了しますが、アプリケーション宣言終了ハンドラは実行不可能にします。コマンド・プロシージャまたは DO 句の中で使用し、プロセスを全く指定しないと、その時点でそのコマンド・プロシージャまたは DO 句が終了します。

---

### フォーマット

QUIT [*process-spec* [, ... ]]

---

### パラメータ

*process-spec*

(保持デバッガのみ。) 現在デバッガの制御下にあるプロセスを指定します。次のいずれかの形式で指定します。

[%PROCESS\_NAME] *process-name*

スペースや小文字を含まないプロセス名。プロセス名にはワイルドカード文字(\*)を含めることができる。

[%PROCESS\_NAME] "*process-name*"

スペースまたは小文字を含むプロセス名。二重引用符(")の代わりに、一重引用符(') 使用することもできる。

%PROCESS\_PID *process\_id*

プロセス識別子 (PID, 16 進数)。

[%PROCESS\_NUMBER] *process-number*  
(または%PROC *process-number*)

デバッガの制御下に入ったときにプロセスに割り当てられた番号。新しい番号は、1 から順番に各プロセスに割り当てられる。EXIT コマンドまたは QUIT コマンドによってプロセスが終了した場合、そのデバッグ・セッション中にその番号が再割り当てされることがある。プロセス番号は SHOW PROCESS コマンドの実行で表示される。プロセスは、組み込みシンボル%PREVIOUS\_PROCESS および%NEXT\_PROCESS によってインデックスづけできるように、循環リスト内に順序づけされる。

*process-set-name*

DEFINE/PROCESS\_SET コマンドで定義された、プロセスのグループを表すシンボル。

%NEXT\_PROCESS

デバッガの循環プロセス・リスト中で可視プロセスの次のプロセス。

%PREVIOUS\_PROCESS

デバッガの循環プロセス・リスト中で可視プロセスの前のプロセス。

`%VISIBLE_PROCESS`

シンボル、レジスタ値、ルーチン呼び出し、ブレークポイントなどの検索時に現在のコンテキストになっているスタック、レジスタ・セット、およびイメージを持つプロセス。

すべてのプロセスを指定するためにワイルドカード文字のアスタリスク(\*)を使用することもできます。

---

## 説明

QUIT コマンドはプログラムの実行を引き起こさないこと、したがってプログラムでアプリケーション宣言終了ハンドラを実行しないこと以外は、EXIT コマンドと類似しています。

### デバッグ・セッションの終了

デバッグ・セッションを終了するには、パラメータを全く指定しないでデバッグ・プロンプトに対し QUIT コマンドを入力します。この結果、デバッグ・セッションが順序正しく終了されます。つまり、デバッグの終了ハンドラが実行され(ログ・ファイルをクローズし、画面とキーパッドの状態を復元するなど)、制御が DCL レベルに戻ります。そのあとは DCL コマンドの DEBUG や CONTINUE を入力してもプログラムのデバッグを続けることはできません(デバッグを再起動する必要があります)。

### コマンド・プロシージャや DO 句での QUIT コマンドの使用

デバッグがコマンド・プロシージャの中で(パラメータを全く指定しないで) QUIT コマンドを実行すると、制御はそのコマンド・プロシージャを起動したコマンド・ストリームに戻ります。コマンド・ストリームは、端末、外側(当該コマンド・プロシージャを含んでいる側)のコマンド・プロシージャ、コマンドまたは画面ディスプレイ定義での DO 句とすることができます。たとえば、コマンド・プロシージャが DO 句の中から起動された場合には、制御はその DO 句に戻り、そこでデバッグはコマンド・シーケンスの中にあれば次のコマンドを実行します。

デバッグは DO 句の中で(パラメータを全く指定しないで) QUIT コマンドを実行する場合、その句にある残りのコマンドは無視し、プロンプトを表示します。

### 指定プロセスの終了

マルチプロセス・プログラムをデバッグする場合、QUIT コマンドを使用すればデバッグ・セッションを終了せずに、指定したプロセスを終了できます。プロンプトに対して QUIT コマンドを入力するか、それともコマンド・プロシージャまたは DO 句の中で QUIT コマンドを使用するかにかかわらず、同じ方法と動作が適用されます。

1 つまたは複数のプロセスを終了するには、それらのプロセスをパラメータとして指定して QUIT コマンドを入力します。この結果、指定されたプロセスのイメージが順序正しく終了されますが、それらのイメージに対応するアプリケーション宣言終了ハンドラは実行されません。その後は、指定したプロセスは SHOW PROCESS/ALL コマンドを実行しても表示されなくなります。

EXIT コマンドとは対照的に、QUIT コマンドを実行しても、プロセスの実行は開始されません。

#### 関連コマンド

DISCONNECT  
@ (実行プロシージャ)  
Ctrl/C  
Ctrl/Y  
Ctrl/Z  
EXIT  
RERUN  
RUN  
SET ABORT\_KEY  
SET PROCESS

---

#### 例

1. DBG> QUIT  
\$

このコマンドは、デバッグ・セッションを終了し、DCL レベルに戻ります。

2. all> QUIT %NEXT\_PROCESS, JONES\_3, %PROC 5  
all>

このコマンドを実行すると、マルチプロセス・プログラムの3つのプロセスが順序正しく終了します。つまり、プロセス・リスト内の可視プロセスの次のプロセス、プロセス JONES\_3、プロセス 5 が終了します。指定されたプロセスが終了したあと、制御はデバッガに戻ります。

---

## REBOOT (Alpha および I64 のみ)

OpenVMS システム・コード・デバッガをオペレーティング・システム・コードのデバッグに使用している場合、オペレーティング・システム・コードを実行しているターゲット・マシンをリブートし、システム・プログラムを実行 (再実行) します。

つまり、OpenVMS システム・コード・デバッガ環境では、REBOOT コマンドは RUN コマンドや RERUN コマンドと同じ働きをします。OpenVMS システム・コード・デバッガは、OpenVMS デバッガから起動するカーネル・デバッガです。

このコマンドを実行する前に、Alpha または I64 のデバイス・ドライバの作成、OpenVMS システム・コード・デバッガの起動、デバッグを可能にする CONNECT コマンドの実行を行う必要があります。

また、OpenVMS デバッガを DEBUG/KEEP コマンドで起動していなければなりません。

---

### フォーマット

REBOOT

---

### 説明

OpenVMS システム・コード・デバッガの使用方法的詳細については、『OpenVMS System Analysis Tools Manual』を参照してください。

#### 関連コマンド

CONNECT  
DISCONNECT

---

### 例

1. DBG> REBOOT

このコマンドは、OpenVMS オペレーティング・システムをデバッグする予定のターゲット・マシンをリブートし、プログラムを再実行します。

---

# REPEAT

指定された回数一連のコマンドを実行します。

---

## フォーマット

```
REPEAT language-expression DO (command; . . . )
```

---

## パラメータ

*language-expression*

現在設定されている言語で評価すると正の整数になる式を示します。

*command*

デバッガ・コマンドを指定します。複数のコマンドを指定する場合、それぞれのコマンドをセミコロン(;)で区切らなければなりません。各コマンドを実行すると、デバッガはそのコマンド内の式の構文をチェックして、評価します。

---

## 説明

REPEAT コマンドは FOR コマンドの単純な形式です。REPEAT コマンドは一連のコマンドを指定回数繰り返し実行します。ただし、FOR コマンドとは異なり、カウント・パラメータを設定するためのオプションは提供しません。

関連コマンド

EXITLOOP

FOR

WHILE

---

## 例

```
DBG> REPEAT 10 DO (EXAMINE Y; STEP)
```

このコマンド行は 2 つのコマンドからなるシーケンス (EXAMINE Y のあと STEP) を 10 回実行するループを設定します。

---

## RERUN

現在デバッガの制御下にあるプログラムを再実行します。

---

### 注意

DCL コマンドの DEBUG/KEEP を使用してデバッグ・セッションを開始し、デバッガの RUN コマンドを実行しておく必要があります。DCL コマンドの RUN filespec でデバッグ・セッションを開始した場合は、デバッガの RERUN コマンドは使用できません。

---

---

## フォーマット

RERUN

---

## 修飾子

/ARGUMENTS="arg-list"

引数のリストを指定します。引用符を指定する場合には、デバッガがその引用符を解析するときに引用を取り除くため、二重引用符を追加する必要があるかもしれません。引数を指定しないと、以前そのプログラムを実行または再実行したときに指定した引数が省略時の値として使用されます。

/HEAP\_ANALYZER

ワークステーション・ユーザにだけ適用されます。アプリケーションのメモリ使用状況を知るための機能であるヒープ・アナライザを起動します。ヒープ・アナライザの使用法についての詳しい説明は、『デバッガ説明書』を参照してください。

/SAVE (省略時の設定)

/NOSAVE

次に実行するプログラムのすべてのブレークポイント、トレースポイント、静的ウォッチポイントの現在の状態 (有効か無効か) を保存するかどうかを制御します。/SAVE 修飾子を指定するとそれらの状態は保存され、/NOSAVE を指定すると保存されません。/SAVE が特定の非静的ウォッチポイントの状態を保存するか保存しないかは、メイン・プログラム・ユニット (実行が再開された場所) を基準とした、ウォッチ対象の変数の有効範囲によって決まります。



---

## 説明

DCL コマンドの DEBUG/KEEP でデバッガを呼び出し、その結果デバッガの RUN コマンドを使用してプログラムのデバッグを開始した場合は、RERUN コマンドを使用して、現在デバッガの制御下にあるプログラムを再実行できます。

RERUN コマンドはデバッグしていたイメージを終了したあと、デバッガの制御下のイメージを再開します。実行は、メイン・プログラム・ユニットが開始されると、デバッガの RUN コマンド、または DCL コマンドの RUN/DEBUG を実行した場合と同じように一時停止されます。

RERUN コマンドは現在デバッガの制御下にあるイメージと同じバージョンを使用します。デバッグ・セッションから同じプログラムの異なるバージョン、または別のプログラムをデバッグするには、RUN コマンドを使用します。

### 関連コマンド

RUN (debugger コマンド)  
RUN (DCL コマンド)  
(ACTIVATE,DEACTIVATE) BREAK  
(ACTIVATE,DEACTIVATE) TRACE  
(ACTIVATE,DEACTIVATE) WATCH

---

## 例

1. DBG> RERUN

このコマンドは現在のプログラムを再実行します。省略時の状態では、デバッガはすべてのブレークポイント、トレースポイント、静的ウォッチポイントの現在の状態 (有効が無効か) を保存します。

2. DBG> RERUN/NOSAVE

このコマンドはブレークポイント、トレースポイント、ウォッチポイントの現在の状態を保存しないで現在のプログラムを再実行します。これは、RUN コマンドを使用してイメージ名を指定することと同じです。

3. DBG> RERUN/ARGUMENTS="fee fii foo fum"

このコマンドは、現在のプログラムを新しい引数で返します。

---

## RUN

デバッガの制御下でプログラムを実行します。

---

### 注意

DCL コマンドの DEBUG/KEEP を使用してデバッグ・セッションを開始しておく必要があります。DCL コマンドの RUN filespec でデバッグ・セッションを開始した場合は、デバッガの RUN コマンドは使用できません。

---

---

## フォーマット

RUN *[program-image]*

---

## パラメータ

program-image

デバッグの対象となるプログラムの実行可能なイメージを指定します。

/COMMAND=*cmd-symbol* 修飾子を使用している場合、イメージは指定できません。

---

## 修飾子

/ARGUMENTS="*arg-list*"

引数のリストを指定します。引用符を指定する場合には、デバッガがその引用符を解析するときに引用を取り除くため、二重引用符を追加する必要があるかもしれません。

/COMMAND="*cmd-symbol*"

プログラムを実行するための DCL フォーリン・コマンドを指定します。

program-image パラメータを指定する場合、この修飾子は指定できません。

SET COMMAND コマンドで作成された DCL コマンド定義またはその他のコマンド定義は指定できません。

/HEAP\_ANALYZER

ワークステーション・ユーザにだけ適用される。アプリケーションのメモリ使用状況を知るための機能であるヒープ・アナライザを起動します。ヒープ・アナライザの使用法についての詳しい説明は、『デバッガ説明書』を参照してください。

/NEW

すでに実行しているプログラムを中断せずに、デバッガ制御下で新しいプログラムを実行します。

---

## 説明

DCL コマンドの DEBUG/KEEP で呼び出した場合のデバッグ・セッション中であればいつでもデバッガの RUN コマンドを使用して、デバッガの制御下でプログラムを起動できます。RUN コマンドを実行したのがプログラムのデバッグの最中であれば、/NEW 修飾子を使用しないかぎり、プログラムはまず停止します。

同じプログラムつまり、現在デバッガの制御下にあるプログラムの同じバージョンを再び実行するには、RERUN コマンドを使用します。RERUN コマンドを使用すれば、任意のブレークポイント、トレースポイント、静的ウォッチポイントの現在の状態 (有効か無効か) を保存できます。

RUN コマンドまたは RERUN コマンドを使用して引数を渡す方法については、『デバッガ説明書』を参照してください。

デバッガの RUN コマンドに関する次の制限事項に注意してください。

- RUN コマンドが使用できるのは DCL コマンドの DEBUG/KEEP でデバッガを起動した場合だけです。
- RUN コマンドを使用して、実行中のプログラムにデバッガを接続することはできません。Ctrl/Y の説明を参照してください。
- DECnet リンクを介してデバッガの制御下にあるプログラムを実行することはできません。デバッグの対象となるイメージとデバッガの両方が同じノードになくてはなりません。

## 関連コマンド

RERUN  
 RUN (DCL コマンド)  
 Ctrl/Y-DEBUG (DCL コマンド)  
 DEBUG (DCL コマンド)

---

## 例

1. DBG> RUN EIGHTQUEENS  
 Language: C, Module: EIGHTQUEENS  
 このコマンドは EIGHTQUEENS プログラムをデバッガの制御下に置きます。

## RUN

2. \$ RUNPROG == "\$ DISK3:[SMITH]MYPROG.EXE"  
\$ DEBUG/KEEP

...  
DBG> RUN/COMMAND="RUNPROG"/ARGUMENTS="X Y Z"

この例の最初の行は MYPROG.EXE という名前のイメージを実行するために (DCL レベルで) コマンド・シンボル RUNPROG を作成します。2 行目はデバッガを起動します。次にデバッガの RUN コマンドは MYPROG.EXE イメージをデバッガの制御下に置きます。/COMMAND 修飾子は以前に作成したコマンド・シンボル (この場合は RUNPROG) を指定し, /ARGUMENTS 修飾子は引数 X Y Z をイメージに引き渡します。

3. DBG> RUN/ARGUMENTS="X Y Z" MYPROG

このコマンドは, プログラム MYPROG.EXE をデバッガの制御下に置いて, 引数 X Y Z を引き渡します。

---

## SAVE

これまでの画面ディスプレイの内容を新しいディスプレイでも保持します。

---

### 注意

---

このコマンドは、デバッガへの HP DECwindows Motif for OpenVMS ユーザ・インタフェースでは使用できません。

---



---

## フォーマット

SAVE *old-display* AS *new-display* [, . . . ]

---

## パラメータ

*old-display*

内容を保存するディスプレイを指定します。次のいずれかを指定できます。

- 定義済みのディスプレイ

SRC

OUT

PROMPT

INST

REG

FREG (Alpha および I64 のみ)

IREG

- DISPLAY コマンドで作成したディスプレイ
- ディスプレイの組み込みシンボル

%CURDISP

%CURSCROLL

%NEXTDISP

%NEXTINST

%NEXTOUTPUT

%NEXTSCROLL

%NEXTSOURCE

*new-display*

作成する新しいディスプレイの名前を指定します。この新しいディスプレイがold-dispディスプレイの内容を受け取ります。

---

## 説明

SAVE コマンドを使用すると、既存のディスプレイのスナップショット・コピーを新しいディスプレイに保存し、あとで参照できます。新しいディスプレイのテキスト内容は、既存のディスプレイのものと同じです。通常の場合、新ディスプレイは画面から削除されたものを除いて旧ディスプレイの属性をすべて受け継ぎます。自動的に更新されることはありません。DISPLAY コマンドを実行すれば、保存したディスプレイを端末画面に再度呼び出すことができます。

SAVE コマンドを使用すると、ディスプレイのメモリ・バッファ (DISPLAY コマンドの /SIZE 修飾子を指定することによって決定されます) に現在格納されている行だけが保存されます。ただし、ソース・ディスプレイまたは機械語命令ディスプレイを保存した場合は、そのモジュールに関連したソース行は保存しなかったものも、またそのルーチンに関連した命令は保存しなかったものも、保存したディスプレイをスクロールすることにより表示できます。

PROMPT ディスプレイは保存できません。

関連コマンド

DISPLAY  
EXITLOOP

---

## 例

```
DBG> SAVE REG AS OLDREG
```

このコマンドは、REG というディスプレイの内容を、新規作成したディスプレイ OLDREG に保存します。

---

# SCROLL

画面ディスプレイをスクロールして、ディスプレイ・ウィンドウにテキストの他の部分を表示します。

---

## 注意

---

このコマンドは、デバッガへの HP DECwindows Motif for OpenVMS ユーザ・インタフェースでは使用できません。

---



---

## フォーマット

SCROLL *[display-name]*

---

## パラメータ

display-name

スクロールするディスプレイを指定します。次のいずれかを指定できます。

- 定義済みディスプレイ

SRC  
OUT  
PROMPT  
INST  
REG  
FREG (Alpha および I64 のみ)  
IREG

- DISPLAY コマンドで作成したディスプレイ
- ディスプレイの組み込みシンボル

%CURDISP  
%CURSCROLL  
%NEXTDISP  
%NEXTINST  
%NEXTOUTPUT  
%NEXTSCROLL  
%NEXTSOURCE

ディスプレイを指定しないと、SELECT コマンドで設定した現在のスクロール・ディスプレイが選択されます。

---

## 修飾子

/BOTTOM

表示テキストの最後までスクロールします。

/DOWN:[n]

表示テキストを  $n$  行だけ下方へスクロールして、テキストのさらに下の部分を表示します。 $n$  を指定しないと、表示はウィンドウ高さの約 3/4 ほどスクロールされます。

/LEFT:[n]

表示テキストを  $n$  で指定した欄数だけ左へスクロールし、ウィンドウの左側の境界より左にあるテキストを表示します。ただし、欄 1 を越えてスクロールすることはできません。 $n$  を指定しないと、8 欄だけ左側へスクロールします。

/RIGHT:[n]

表示テキストを  $n$  で指定した欄数だけ右へスクロールし、ウィンドウの右側の境界より右にあるテキストを表示します。ただし、欄 255 を越えてスクロールすることはできません。 $n$  を指定しないと、8 欄だけ右側へスクロールします。

/TOP

表示テキストの最上部までスクロールします。

/UP:[n]

表示テキストを  $n$  行だけ上方へスクロールし、それより上にあるテキストを表示します。 $n$  を指定しない場合、ディスプレイはウィンドウ高さの約 3/4 ほどスクロールされます。

---

## 説明

SCROLL コマンドは、ディスプレイをウィンドウの上、下、右または左の方向に移動することにより、表示テキストのいろいろな部分をウィンドウに表示できます。

SCROLL コマンド (現在のスクロール・ディスプレイ) のディスプレイを選択するときは、SELECT/SCROLL コマンドを使用します。

SCROLL コマンドに関連したキー定義については、ヘルプ・トピック Keypad\_Definitions\_CI を参照してください。また、現在のキー定義を調べるには SHOW KEY コマンドを使用してください。

関連コマンド SELECT.



---

## 例

1. `DBG> SCROLL/LEFT`

このコマンドは、現在のスクロール・ディスプレイを 8 欄だけ左へスクロールします。

2. `DBG> SCROLL/UP:4 ALPHA`

このコマンドはスクロール・ディスプレイ ALPHA の 4 行上へスクロールします。

---

## SEARCH

指定した文字列をソース・コードから検索し、その文字列を含んでいるソース行を表示します。

---

### フォーマット

SEARCH *[range] [string]*

---

### パラメータ

range

検索するプログラム領域を指定します。次のいずれかの書式を使用します。

<i>mod-name</i>	指定したモジュール内の行 0 からモジュールの最後までの間で検索します。
<i>mod-name</i> \ <i>line-num</i>	指定されたモジュール内の指定された行番号からモジュールの最後までの間で検索します。
<i>mod-name</i> \ <i>line-num:line-num</i>	指定されたモジュール内のコロンの左で指定した行番号からコロンの右で指定した行番号までの間で検索します。
<i>line-num</i>	現在の有効範囲を使用してモジュールを決定し、そのモジュール内の指定された行番号からモジュールの最後までの間で検索します。現在の有効範囲は、前の SET SCOPE コマンドで設定されています。SET SCOPE コマンドを指定しなかった場合は PC 有効範囲です。SET SCOPE コマンドで有効範囲検索リストを指定すると、デバッガは最初に指定された有効範囲に関連したモジュールだけを検索します。
<i>line-num:line-num</i>	現在の有効範囲を使用してモジュールを決定し、そのモジュール内のコロンの左の行番号からコロンの右の行番号までの間で検索します。現在の有効範囲は、前の SET SCOPE コマンドで設定されています。SET SCOPE コマンドを指定しなかった場合は PC 有効範囲です。SET SCOPE コマンドで有効範囲検索リストを指定すると、デバッガは最初に指定された有効範囲に関連したモジュールだけを検索します。
null (入力なし)	最後にソース行が表示された (たとえば、コマンド TYPE, EXAMINE/SOURCE, または SEARCH を実行した結果) モジュールの中のその最後に表示された行の直後からそのモジュールの最後までの間で検索します。
string	検索するソース・コード文字を指定します。文字列を指定しない場合、最後に実行した SEARCH コマンドで文字列を指定していればその文字列が使用されます。

次の場合は、文字列を二重引用符(")または一重引用符(')で囲まなければなりません。

- 文字列の前部か後部に、スペース文字かタブ文字が含まれている場合
- 文字列の中間にセミコロンが含まれている場合
- レンジ・パラメータが空の場合

文字列を二重引用符で囲む場合、文字列自体が二重引用符を含んでいればそれを示すために2つの二重引用符("")を並べて使用します。一重引用符で囲む場合は、2つの一重引用符('')を並べて使用して、文字列内の一重引用符であることを示します。

---

## 修飾子

/ALL

指定された範囲内の文字列をデバッガがすべて検索し、その文字列を含んでいる行ごとに表示することを指定します。

/IDENTIFIER

指定された範囲内で文字列を検索すると、さらに、その文字列が現在の言語で識別子の一部を構成する文字によってどちらの側にもつながっていない文字列なら、それを表示することを指定します。

/NEXT

省略時の設定。デバッガは指定された範囲内で文字列が次に出現するところを検索し、その文字列がある行だけ表示することを指定します。

/STRING

省略時の設定。デバッガは指定された文字列を検索して、表示するように、さらに文字列の出現箇所のコンテキストは解釈しないよう指定します。後者の点は/IDENTIFIERの場合と異なります。

---

## 説明

SEARCH コマンドは、指定された文字列を含んでいるソース・コード行を表示します。

SEARCH コマンドでモジュール名を指定したら、そのモジュールは設定されていなければなりません。特定のモジュールが設定されているかどうか確認するには、SHOW MODULE コマンドを使用します。設定されていなければSET MODULE コマンドを使用します。

SEARCH コマンドの修飾子は、デバッガが(1)その文字列のすべての出現を検索する (/ALL) かまたは次に出現するものを検索する (/NEXT) かを決定し、(2)出現文字列をすべて (/STRING) 表示するか、現在の言語で識別子を構成する文字によってどちら側にもつながっていない文字列だけ (/IDENTIFIER) を表示するかを決定します。

修飾子が同じ SEARCH コマンドを複数個入力する場合、まず、SET SEARCH コマンドを使用して、新しい省略時の修飾子を設定します。たとえば、SET SEARCH ALL と指定すると、SEARCH コマンドは SEARCH/ALL と同じように動作します。この結果、SEARCH コマンドにこの修飾子を使用しないで済みます。また SEARCH コマンドで省略時の設定とは異なる修飾子を指定すれば、その SEARCH コマンドの実行中は、現在の省略時の修飾子を上書きできます。

#### 関連コマンド

```
(SET,SHOW) LANGUAGE
(SET,SHOW) MODULE
(SET,SHOW) SCOPE
(SET,SHOW) SEARCH
```

---

#### 例

```
1. DBG> SEARCH/STRING/ALL 40:50 D
   module COBOLTEST
      40: 02      D2N      COMP-2 VALUE -234560000000.
      41: 02      D        COMP-2 VALUE  222222.33.
      42: 02      DN      COMP-2 VALUE -222222.333333.
      47: 02      DR0     COMP-2 VALUE   0.1.
      48: 02      DR5     COMP-2 VALUE  0.000001.
      49: 02      DR10    COMP-2 VALUE  0.000000000001.
      50: 02      DR15    COMP-2 VALUE  0.0000000000000001.
DBG>
```

このコマンドは、現在の有効範囲にあるモジュール COBOLTEST の行 40 ~ 50 で、文字 D をすべて検索します。

```
2. DBG> SEARCH/IDENTIFIER/ALL 40:50 D
   module COBOLTEST
      41: 02      D        COMP-2 VALUE  222222.33.
DBG>
```

このコマンドは、COBOLTEST モジュールの行 40 ~ 50 で文字 D をすべて検索します。デバッガは、現在の言語で識別子の一部を構成する文字によって文字 D(検索文字列) のどちらの側にもつながっていない行だけを表示します。

```
3.  DBG> SEARCH/NEXT 40:50 D
    module COBOLTEST
      40: 02      D2N      COMP-2 VALUE -234560000000.
DBG>
```

このコマンドは、COBOLTEST モジュールの行 40 ~ 50 の間で次の文字 D を検索します。

```
4.  DBG> SEARCH/NEXT
    module COBOLTEST
      41: 02      D      COMP-2 VALUE  222222.33.
DBG>
```

このコマンドは、次の文字 D を検索します。D が最後に入力したものであり、ほかに検索文字列を指定していないため、デバッガはこの D を検索文字列であるとみなします。

```
5.  DBG> SEARCH 43 D
    module COBOLTEST
      47: 02      DR0     COMP-2 VALUE  0.1.
DBG>
```

このコマンドは、行 43 から、次の省略時の設定文字 D を検索します。

---

## SDA

OpenVMS デバッガの中から、デバッグ・セッションを終了せずに、システム・ダンプ・アナライザ (SDA) を起動します。

---

### フォーマット

SDA *[sda-command]*

---

### パラメータ

sda-command

制御を OpenVMS デバッガに戻す前に実行される 1 つの SDA コマンド。

---

### 説明

SDA コマンドを使用すると、デバッガの中から以下のタスクのためにシステム・ダンプ・アナライザ (SDA) を使用することができます。

- システム・コード・デバッガ (SCD) によるシステム・コードのデバッグ (Alpha および I64 のみ)
- システム・ダンプ・デバッガ (SDD) によるシステム・ダンプ解析 (Alpha および I64 のみ)
- システム・デバッグ・アナライザ (SDA) によるプロセス・ダンプ解析 (Alpha および I64 のみ)

これにより、デバッグ・セッションの中からすべての SDA コマンドにアクセスすることができます。SDA を終了すると、同じデバッグ・セッションに戻ります。SDA セッションの中でデバッグ・コマンドにアクセスすることはできないことに注意してください。

---

#### 注意

ユーザ・モード・プログラムのデバッグ中は、SDA コマンドは使用できません。

---

### 関連コマンド

ANALYZE/CRASH\_DUMP  
ANALYZE/PROCESS\_DUMP

## CONNECT %NODE

---

例

1. 

```
DBG> SDA
OpenVMS (TM) Alpha process dump analyzer

SDA> ..
.
.
SDA> EXIT
DBG>
```

この例は、OpenVMS デバッガ内で SDA セッションをオープンし、何らかの解析を実行し、SDA セッションをクローズしてデバッガに制御を戻しています。

2. 

```
DBG> SDA SHOW PROCESS
.
.
DBG>
```

この例は、デバッガの中から 1 つの SDA コマンドを実行した後に、デバッガに制御を戻しています。

---

## SELECT

現在のエラー・ディスプレイ，入力ディスプレイ，機械語命令ディスプレイ，出力ディスプレイ，プログラム・ディスプレイ，プロンプト・ディスプレイ，スクロール・ディスプレイ，またはソース・ディスプレイを画面ディスプレイとして選択します。

---

### 注意

このコマンドは，デバッガへの HP DECwindows Motif for OpenVMS ユーザ・インタフェースでは使用できません。

---

---

## フォーマット

SELECT *[display-name]*

---

## パラメータ

display-name

選択するディスプレイを指定します。次のいずれかを指定できますが，修飾子の説明で述べるように制限事項があります。

- 定義済みディスプレイ

SRC  
OUT  
PROMPT  
INST  
REG  
FREG (Alpha および I64 のみ)  
IREG

- A display previously
- DISPLAY コマンドで作成したディスプレイ
- ディスプレイの組み込みシンボル

%CURDISP  
%CURSCROLL  
%NEXTDISP  
%NEXTINST  
%NEXTOUTPUT  
%NEXTSCROLL  
%NEXTSOURCE



このパラメータを省略し、修飾子を指定しないと、現在のスクロール・ディスプレイの"選択を解除する"ことになります。ディスプレイのスクロール属性がなくなります。このパラメータを省略したのに修飾子/INPUT、/SOURCEなどを指定すると、その属性を持つ現在のディスプレイの選択が解除されます。修飾子の説明を参照してください。

---

## 修飾子

### /ERROR

指定されたディスプレイを現在のエラー・ディスプレイとして選択します。この結果、すべてのデバッグ診断メッセージがそのディスプレイに送られます。指定するディスプレイは、出力ディスプレイか PROMPT ディスプレイのどちらかでなければなりません。ディスプレイを指定しないと、この修飾子は PROMPT 表示が現在のエラー・ディスプレイとして選択します。省略時には、PROMPT ディスプレイがエラー属性を持ちます。

### /INPUT

指定されたディスプレイを現在の入力ディスプレイとして選択します。この結果、このディスプレイにデバッグ入力 (PROMPT ディスプレイ内に表示されます) がエコーバックされます。指定するディスプレイは出力ディスプレイでなければなりません。

ディスプレイを指定しないと、現在の入力ディスプレイが選択解除され、デバッグ入力はどのディスプレイにもエコーバックされません。デバッグ入力は PROMPT ディスプレイにだけ表示されます。省略時には、どのディスプレイも入力属性を持ちません。

### /INSTRUCTION

指定されたディスプレイを現在の機械語命令ディスプレイとして選択します。この結果、すべての EXAMINE/INSTRUCTION コマンドの出力がそのディスプレイに送られます。指定されたディスプレイは、機械語命令ディスプレイでなければなりません。

ディスプレイを指定しないと、現在の機械語命令ディスプレイが選択解除され、どのディスプレイも命令属性を持ちません。

省略時には、MACRO-32 を除くどの言語にも命令属性を持つディスプレイはありません。言語を MACRO-32 に設定すると、INST ディスプレイは省略時に命令属性を持ちます。

### /OUTPUT

指定されたディスプレイを現在の出力ディスプレイとして選択します。この結果、デバッグは他のディスプレイに出力先が指定されていない出力をこのディスプレイに送ります。指定されたディスプレイは出力ディスプレイまたは PROMPT ディスプレイのどちらかでなければなりません。

ディスプレイを指定しないと、PROMPT ディスプレイが現在の出力ディスプレイとして選択されます。省略時には、OUT ディスプレイが出力属性を持ちます。

#### /PROGRAM

指定されたディスプレイを現在のプログラム・ディスプレイとして選択します。この結果、デバッガはプログラムの入出力をそのディスプレイに送り込もうとします。現在は、PROMPT ディスプレイだけを指定できます。

ディスプレイを指定しないと、現在のプログラム・ディスプレイは選択解除され、プログラムの入出力は指定されたディスプレイには送られなくなります。

省略時には、PROMPT ディスプレイがプログラム属性を持ちます。ただし、ワークステーションでは、プログラム属性が選択解除されます。

#### /PROMPT

指定されたディスプレイを現在のプロンプト・ディスプレイとして選択します。デバッガは、ここに入力を促すプロンプトをディスプレイします。現在のところ、PROMPT ディスプレイだけ指定できます。また、PROMPT ディスプレイの選択解除はできません (PROMPT ディスプレイは常にプロンプト属性を持っています)。

#### /SCROLL

省略時の設定。指定されたディスプレイが現在のスクロール・ディスプレイとして選択されます。これは、コマンド SCROLL, MOVE, EXPAND 用の省略時のディスプレイです。どのディスプレイもスクロール属性を持つことができますが、PROMPT ディスプレイで利用できるのは、MOVE コマンドと EXPAND コマンドだけです (SCROLL コマンドは使用できません)。

ディスプレイを指定しないと、現在のスクロール・ディスプレイは選択解除され、どのディスプレイもスクロール属性を持たなくなります。

省略時には、MACRO-32 を除くどの言語の場合にも、SRC ディスプレイがスクロール属性を持ちます。言語が MACRO-32 に設定されると、INST ディスプレイが省略時設定としてスクロール属性を持ちます。

#### /SOURCE

指定されたディスプレイを現在のソース・ディスプレイとして選択します。この結果、すべての TYPE コマンドと EXAMINE/SOURCE コマンドの出力がそのディスプレイに送られます。指定されたディスプレイはソース・ディスプレイでなければなりません。

ディスプレイを指定しないと、現在のソース・ディスプレイが選択解除され、どのディスプレイもソース属性を持たないことになります。

省略時には、MACRO-32 を除くどの言語の場合にも、SRC ディスプレイがソース属性を持ちます。言語が MACRO-32 に設定されると、どのディスプレイも省略時設定としては、ソース属性を持ちません。

---

## 説明

属性は、現在のスクロール・ディスプレイを選択し、各種のデバッグ出力を特定のディスプレイに出力することを指定するのに使用します。これにより、デバッグ入力、デバッグ出力、デバッグ診断メッセージなどの種類の異なる情報をスクロール可能なディスプレイ内に混ぜて示したり、別のディスプレイに示したりできます。

SELECT コマンドに 1 つまたは複数の修飾子 (/ERROR, /SOURCE など) を指定すると、1 つのディスプレイに相当する属性を 1 つまたは複数個割り当てることができます。修飾子を指定しないと、省略時の値として /SCROLL が指定されているものとみなされます。

ディスプレイ名を指定しないで SELECT コマンドを使用すると、修飾子が示している属性割り当ては取り消されます (選択解除されます)。表示属性を再割り当てするには、もう一度 SELECT コマンドを実行しなければなりません。詳しい説明は、各修飾子を参照してください。

SELECT コマンドに関連したキー定義は、ヘルプ・トピック `Keypad_Definitions_CI` を参照してください。また、現在のキー定義を調べるには、`SHOW KEY` コマンドを使用してください。

### 関連コマンド

DISPLAY  
EXPAND  
MOVE  
SCROLL  
SHOW SELECT

---

## 例

1. `DBG> SELECT/SOURCE/SCROLL SRC2`

このコマンドは、`SRC2` ディスプレイを現在のソース・ディスプレイおよびスクロール・ディスプレイとして選択します。

2. `DBG> SELECT/INPUT/ERROR OUT`

このコマンドは、`OUT` ディスプレイを現在の入力ディスプレイおよびエラー・ディスプレイとして選択します。この結果、デバッグ入力、デバッグ出力 (`OUT` を現在の出力ディスプレイとみなして)、デバッグ診断メッセージが `OUT` ディスプレイ内に正しい順に記録されます。

## SELECT

### 3. DBG> SELECT/SOURCE

このコマンドは、現在選択されているソース・ディスプレイを選択解除 (現在選択されているソース・ディスプレイからソース属性を削除) します。TYPE コマンドまたは EXAMINE/SOURCE コマンドの出力は、現在選択されている出力ディスプレイに送られます。

---

## SET ABORT\_KEY

デバッガの強制終了機能を他の Ctrl キー・シーケンスに割り当てます。省略時には、Ctrl/C が強制終了機能になります。

---

### 注意

このコマンドは、デバッガへの HP DECwindows Motif for OpenVMS ユーザ・インタフェースでは使用できません。

---

---

## フォーマット

SET ABORT\_KEY = *CTRL\_character*

---

## パラメータ

character

Ctrl キーを押している間に押すキーを指定します。どの英数字のキーでも指定できます。

---

## 説明

省略時の設定では、デバッグ・セッション中に Ctrl/C を入力すると、デバッグ・コマンドの実行が強制終了され、プログラムの実行に割り込みがかかります。SET ABORT\_KEY コマンドを使用して、この強制終了機能を他の Ctrl キー・シーケンスに割り当てることができます。これは、プログラムで Ctrl/C AST サービス・ルーチンを使用する場合に必要です。

Ctrl キー・シーケンスには定義済みの機能がすでに割り当てられている場合が多く、SET ABORT\_KEY コマンドを実行すると、このようなキー定義が上書きされます『OpenVMS ユーザーズ・マニュアル』を参照してください。オペレーティング・システムの Ctrl キー文字で使用されていないのは、G、K、N、および P です。

SHOW ABORT\_KEY コマンドは、現在有効な強制終了機能として Ctrl キーシーケンスを表示します。

デバッグ・セッションでは Ctrl/Y は使用できません。代わりに、Ctrl/C を使用するか、SET ABORT\_KEY コマンドで設定した同等の Ctrl キー・シーケンスを使用してください。

## SET ABORT\_KEY

### 関連コマンド

Ctrl/C  
Ctrl/Y  
SHOW ABORT\_KEY

---

### 例

```
DBG> SHOW ABORT_KEY
Abort Command Key is CTRL_C
DBG> GO
. . .
Ctrl/C
DBG> EXAMINE/BYTE 1000:101000 !should have typed 1000:1010
1000: 0
1004: 0
1008: 0
1012: 0
1016: 0
Ctrl/C
%DEBUG-W-ABORTED, command aborted by user request
DBG> SET ABORT_KEY = CTRL_P
DBG> GO
. . .
Ctrl/P
DBG> EXAMINE/BYTE 1000:101000 !should have typed 1000:1010
1000: 0
1004: 0
1008: 0
1012: 0
1016: 0
Ctrl/P
%DEBUG-W-ABORTED, command aborted by user request
DBG>
```

この例は、次のことを示しています。

- 強制終了機能には Ctrl/C を使用する (省略時)。
- Ctrl/P に強制終了機能を再割り当てするには、SET ABORT\_KEY コマンドを使用する。

---

## SET ATSIGN

デバッガがコマンド・プロシージャの検索に使用する省略時のファイル指定を設定します。

---

### フォーマット

SET ATSIGN *file-spec*

---

### パラメータ

*file-spec*

省略時にデバッガがコマンド・プロシージャの検索に使用するファイル指定の任意の部分 (たとえば、ディレクトリ名またはファイルの種類など) を指定します。完全なファイル指定を行わないと、デバッガは、欠落しているフィールドの省略時のファイル指定は SYS\$DISK:[ ]DEBUG.COM であるとみなします。

検索リストに変換する論理名を指定できます。この場合、デバッガは、目的のコマンド・プロシージャが見つかるまで、検索リストに並んでいる順にファイル指定を処理します。

---

### 説明

実行プロシージャ (@) コマンドでデバッガ・コマンド・プロシージャを起動すると、デバッガは、省略時の設定として、コマンド・プロシージャのファイル指定が SYS\$DISK:[ ]DEBUG.COM であるとみなします。SET ATSIGN コマンドを指定すると、この省略時の設定を上書きできます。

関連コマンド

@ (実行プロシージャ)  
SHOW ATSIGN

---

### 例

```
DBG> SET ATSIGN USER: [JONES.DEBUG] .DBG
DBG> @TEST
```

この例では、@TEST コマンドを実行するとデバッガが USER:[JONES.DEBUG]内で TEST.DBG ファイルを探します。

---

## SET BREAK

特定のクラスの命令または指定されたイベントの発生時に、アドレス式で示された記憶位置にブレークポイントを設定します。

---

### フォーマット

```
SET BREAK  [address-expression[, ... ]]
           [WHEN(conditional-expression)]
           [DO(command[: ... ])]
```

---

### パラメータ

*address-expression*

ブレークポイントを設定するアドレス式 (プログラム記憶位置) を指定します。高級言語の場合、これはたいてい、行番号、ルーチン名、またはラベルです。値を一意に指定するパス名をいれることもできます。また、アドレス式はメモリ・アドレスまたはレジスタの場合もあります。数字 (オフセット) とシンボルで構成したり、1 つまたは複数の演算子、オペランド、または区切り文字で構成したりできます。アドレス式で利用できる演算子についての詳しい説明は、ヘルプ・トピック `Address_Expressions` を参照してください。

ワイルドカード文字のアスタリスク (\*) は使用できません。また次の修飾子では、アドレス式を指定できません。

```
/ACTIVATING
/BRANCH
/CALL
/EXCEPTION
/HANDLER
/INSTRUCTION
/INSTRUCTION=(opcode[, ... ]) (VAX のみ)
/INTO
/[NO]JSB (VAX のみ)
/LINE
/OVER
/[NO]SHARE
/[NO]SYSTEM
/SYSEMULATE (Alpha のみ)
/TERMINATING
/UNALIGNED_DATA (Alpha および I64 のみ)
```



/MODIFY 修飾子と/RETURN 修飾子は、特定の種類のアドレス式に指定できます。

メモリ・アドレスを指定したり、値がシンボリック記憶位置でないアドレス式を指定する場合は、これにより指定されたメモリ・バイトで命令が実際に始まっているかを (EXAMINE コマンドを使用して) チェックしてください。命令がこのバイトで始まっていないと、そのバイトを持つ命令を実行したときに実行時エラーが発生します。値がシンボリック記憶位置でないアドレス式を指定してブレークポイントを設定しても、デバッグは指定された記憶位置が命令の開始位置を示しているかどうかをチェックしません。

たとえば、VAX システムの場合は、CALLS ルーチンと CALLG ルーチンは、エントリ・マスクで始まります。

#### conditional-expression

現在設定されている言語で条件式を指定します。この式は実行がブレークポイントに達したときに評価されます。(ブレークポイントが設定されたときではなく、実行がブレークポイントに達すると、デバッグは WHEN 句にある式の構文をチェックします。) 式が真であれば、デバッグは、ブレークポイントがトリガされたことを報告します。ブレーク・アクション (DO 句) がブレークポイントと関連付けられている場合は、同時に実行されます。式が偽の場合は報告されません。また、DO 句によって指定されたコマンドは実行されず、プログラムの実行は続行されます。

#### command

ブレーク・アクションが実行されたときに、DO 句の一部として実行されるデバッグ・コマンドを指定します。ブレークポイントが設定されたときではなく、DO 句が実行されると、デバッグは DO 句にある式の構文をチェックします。

---

## 修飾子

### /ACTIVATING

新しいプロセスがデバッグの制御下に置かれると、デバッグはブレークします。最初のプロセスがデバッグの制御下に置かれると、デバッグ・プロンプトが表示されます。この結果、プログラムが実行を開始する前にデバッグ・コマンドを入力できます。/TERMINATING 修飾子も参照してください。

### /AFTER:n

指定されたブレークポイントが  $n$  回検出されるまで、ブレーク・アクションを行わないことを指定します ( $n$  は 10 進整数です)。それ以降は、WHEN 句の条件 (指定された場合) が真ならば、検出されるたびにブレークポイントが発生します。SET BREAK/AFTER:1 コマンドは、SET BREAK と同じです。

### /BRANCH

プログラムの実行中に分岐命令を検出するたびに、デバッグがブレークします。/INTO 修飾子と/OVER 修飾子も参照してください。

**/CALL**

プログラムの実行中に呼び出し命令 (RET 命令を含む) を検出するたびにデバッガがブレークします。/INTO 修飾子と/OVER 修飾子も参照してください。

**/EVENT=event-name**

指定されたイベントが発生する (現在のイベント機能によってイベントが定義され、そのイベントが検出される) と、デバッガがブレークします。アドレス式に/EVENT を指定すると、そのアドレス式に対して指定されたイベントが発生するたびにデバッガがブレークします。アドレス式に特定のイベント名を指定することはできません。

イベント機能は、Ada ルーチンまたは SCAN ルーチンを呼び出すプログラムまたは POSIX Threads サービスを使用するプログラムで使用できます。現在のイベント機能および関連したイベント名を表示するには、SHOW EVENT\_FACILITY コマンドを使用します。

**/EXCEPTION**

例外がシグナル通知されるたびにデバッガがブレークします。ブレーク・アクションは、アプリケーションが宣言した例外ハンドラが起動される前に実行されます。

プログラムが例外を生成すると、SET BREAK/EXCEPTION コマンドの結果として、デバッガはプログラムの実行を中断して例外を報告し、そのプロンプトを表示します。例外ブレークポイントから実行を再開した場合、次のいずれかの動作が実行されます。

- address-expression パラメータを指定しないで GO コマンドを入力すると、例外が再度シグナル通知されます。その結果、アプリケーションで宣言した例外ハンドラを実行できます。
- address-expression パラメータを指定して GO コマンドを入力すると、指定された記憶位置からプログラムが続行されます。そのため、アプリケーションで宣言した例外ハンドラは実行できません。
- VAX 上で、STEP コマンドを入力すると、デバッガはアプリケーションで宣言した例外ハンドラ内の命令をステップ実行します。その例外に対してアプリケーションで宣言したハンドラがないと、デバッガは例外を再度シグナル通知します。

Alpha 上では、ハンドラ内でのデバッガの実行を一時停止するための STEP コマンドまたは GO コマンドを入力する前に、例外ハンドラにブレークポイントを明示的に設定しなければなりません。

- CALL コマンドを入力した場合、指定されたルーチンが実行されます。

Alpha プロセッサでは、例外が発生した命令実行の直後に、例外を (プログラムまたはデバッガに) 通知できないことがあります。したがって、デバッガは、実行の結果実際に例外を生成した命令よりあとの命令で実行を中断することがあります。

**/HANDLER**

デバッグしているプログラムに例外がある場合、デバッガは呼び出しスタックをスキャンし、設定されているフレーム・ベースの各ハンドラでブレークポイントを設定し

ようします。デバuggは、標準の RTL ハンドラとユーザ設定ハンドラを区別しません。

Alpha システムおよび I64 システムでは、ユーザ・プログラムが独自のハンドラを定義しているフレームで、多くの RTL がジャケット RTL ハンドラを設定します。この RTL ジャケットは、設定、引数の操作を行い、ユーザが作成したハンドラにディスパッチします。例外を処理する場合、ジャケット RTL ハンドラが呼び出しスタック上のアドレスであるため、デバuggはそこにだけブレークポイントを設定できます。デバuggがジャケット RTL ハンドラでプログラムの実行を中止した場合には、通常、STEP/CALL コマンドを数回と STEP/INTO コマンドを実行してディスパッチ・ポイントを見つけることにより、ユーザ定義ハンドラに到達することができます。

フレーム・ベースのハンドラの詳細については、OpenVMS 呼び出し規則を参照してください。

ジャケット RTL ハンドラが、ALPHA LIBOTS など、インストールされている共用イメージの一部である場合には、デバuggはそこにブレークポイントを設定できません (プライベート・ユーザ・モード書き込みアクセスなし)。この場合には、次の例のように論理名によってすべての RTL をプライベート・イメージとして有効にしてください。

```
$DEFINE LIBOTS SYS$SHARE:LIBOTS.EXE;
```

最後のセミコロン (;) は必ず指定してください。また、インストールされている共用 RTL は、すべてをプライベートで有効にするか、どれもプライベートで有効にしないかの、どちらかでなければなりません。SHOW IMAGE/FULL データを使用してシステム空間のコード・セクションのイメージ・リストを認識してから、すべてのセクション用の論理名を定義し、デバugg・セッションを再実行してください。

```
/INSTRUCTION
```

```
/INSTRUCTION[=(opcode[, . . . ])]
```

命令コードを指定しないと、プログラム実行中に命令を検出するたびにデバuggがブレークします。

VAX プロセッサの場合に限り、命令コードを 1 つまたは複数個選択して指定できます。これによって、リスト内で指定した命令コードが実行されるたびに、デバuggがブレークします。

/INTO 修飾子と/OVER 修飾子も参照してください。

```
/INTO
```

省略時の設定。次の修飾子で設定されているブレークポイント (すなわち、アドレス式が明示的に指定されていない場合) にだけ指定できます。

```
/BRANCH
```

```
/CALL
```

```
/INSTRUCTION
```

```

/INSTRUCTION=(opcode[, ... ]) (VAX のみ)
/LINE

```

これらの修飾子といっしょに使用すると、/INTO は、呼び出されたルーチン内 (実行が現在中断されているルーチン内だけでなく) の指定された地点でデバッガがブレークします。/INTO 修飾子は省略時の設定であり、/OVER の反対です。

/INTO を使用すると、/[NO]JSB、/[NO]SHARE、および/[NO]SYSTEM でブレーク・アクションをさらに修飾できます。

```

/JSB
/NOJSB

```

(VAX のみ) /INTO を修飾します。/INTO と次のいずれかの修飾子といっしょに使用します。

```

/BRANCH
/CALL
/INSTRUCTION
/INSTRUCTION=(opcode[, ... ])
/LINE

```

/JSB 修飾子は、DIBOL 以外のすべての言語における省略時の値です。デバッガは、JSB 命令または CALL 命令によって呼び出されるルーチン内でブレークします。/NOJSB 修飾子 (DIBOL のときの省略時の設定) は、JSB 命令によって呼び出されるルーチン内でブレークポイントを設定しないことを指定します。DIBOL の場合、アプリケーションで宣言したルーチンは CALL 命令によって呼び出され、DIBOL 実行時ライブラリ・ルーチンが JSB 命令によって呼び出されます。

```

/LINE

```

プログラムの実行中にソース行が検出されるたびにその行の先頭で、デバッガがブレークします。/INTO 修飾子と/OVER 修飾子も参照してください。

```

/MODIFY

```

アドレス式が示す記憶位置に値を書き込んで変更する命令を検出するたびに、デバッガがブレークします。アドレス式は、通常の場合変数名です。

SET BREAK/MODIFY コマンドは、SET WATCH コマンドと全く同じように動作し、同じ制限事項が適用されます。

アドレス式に絶対アドレスを指定すると、デバッガがアドレスを特定のデータ・オブジェクトに関連づけることができない場合があります。この場合、デバッガは省略時の長さとして 4 バイトを使用します。ただし、この長さは、入力を WORD (SET TYPE WORD で省略時の長さを 2 バイトに変更する) か BYTE (SET TYPE BYTE で省略時の長さを 1 バイトに変更する) に設定すれば変更できます。SET TYPE LONGWORD を指定すると、省略時の長さは 4 バイトに戻ります。

/OVER

次のいずれかの修飾子で設定されているブレークポイント (すなわち、アドレス式が明示的に指定されていない場合) だけに指定できます。

/BRANCH

/CALL

/INSTRUCTION

/INSTRUCTION=(*opcode* [, . . . ]) (VAX のみ)

/LINE

これらの修飾子といっしょに/OVER を使用すると、(呼び出されたルーチン内ではなく) 現在実行を中断しているルーチン内だけの指定された地点でデバuggはブレークします。/OVER 修飾子は、/INTO (省略時の設定) の反対です。

/RETURN

指定されたアドレス式 (ルーチン名、行番号など) に関連しているルーチンの復帰命令でデバuggがブレークします。復帰命令でブレークすると、ルーチンがアクティブである間ローカル環境を調べる (たとえば、ローカル変数の値を得るなど) ができます。ローカル環境のビューはアーキテクチャにより異なるので注意してください。

VAX プロセッサの場合、この修飾子は、CALLS 命令または CALLG 命令で呼び出したルーチンにだけ指定できます。JSB ルーチンでは使用できません。Alpha プロセッサの場合、この修飾子は、どのルーチンにも指定できます。

*address-expression*/パラメータは、ルーチン内の命令アドレスです。単なるルーチン名の場合もあります。この場合は、ルーチンの開始アドレスを指定します。ただし、ルーチン内の別の記憶位置を指定することもできます。こうすると、特定のプログラム部分を実行したあとに行われる戻りだけを表示できます。

SET BREAK/RETURN コマンドで SET BREAK と同じアドレス式を指定すると、前回の SET BREAK は取り消されます。

/SHARE (省略時の設定)

/NOSHARE

/INTO を修飾します。/INTO と次のいずれかの修飾子といっしょに使用します。

/BRANCH

/CALL

/INSTRUCTION

/INSTRUCTION=(*opcode* [, . . . ]) (VAX のみ)

/LINE

/SHARE 修飾子を使用すると、他のルーチンだけでなく共用可能イメージ・ルーチン内でもデバuggをブレークできます。/NOSHARE 修飾子は、共用可能イメージ内でブレークポイントを設定しないことを指定します。

/SILENT

/NOSILENT (省略時の設定)

"break . . . "メッセージと、現在の記憶位置のソース行をブレークポイントで表示するかどうかを制御します。/NOSILENT 修飾子を指定すると、メッセージが表示されます。/SILENT 修飾子を指定すると、メッセージとソース行は表示されません。/SILENT 修飾子を指定すると、/SOURCE は上書きされます。SET STEP [NO]SOURCE コマンドも参照してください。

/SOURCE (省略時の設定)

/NOSOURCE

現在の記憶位置のソース行をブレークポイントで表示するかどうかを制御します。/SOURCE 修飾子を指定すると、ソース行が表示されます。/NOSOURCE 修飾子を指定すると、ソース行は表示されません。/SILENT 修飾子を指定すると、/SOURCE は上書きされます。SET STEP [NO]SOURCE コマンドも参照してください。

/SYSEMULATE[=mask]

(Alpha のみ) オペレーティング・システムが命令をエミュレートした後、プログラムの実行を停止し、制御をデバッガに戻します。mask は省略可能な引数であり、エミュレートされたどの命令グループがブレークポイントを発生させるかを指定するためのビットを設定した符号なしクォードワードです。現在定義されているエミュレート済み命令グループは、BYTE 命令と WORD 命令です。この命令グループは、mask のビット 0 を 1 に設定することにより選択します。

mask が指定されていない場合や、mask = FFFFFFFFFFFFFFFF の場合には、オペレーティング・システムが任意の命令をエミュレートしたときに、デバッガはプログラムの実行を停止します。

/SYSTEM (省略時の設定)

/NOSYSTEM

/INTO を修飾します。/INTO と次のいずれかの修飾子と一しょに使用します。

/BRANCH

/CALL

/INSTRUCTION

/INSTRUCTION=(opcode[, . . . ]) (VAX のみ)

/LINE

/SYSTEM 修飾子を指定すると、他のルーチンだけでなくシステム・ルーチン (P 1 空間) 内でもデバッガがブレークできます。/NOSYSTEM 修飾子を指定すると、システム・ルーチン内ではブレークポイントが設定されません。

/TEMPORARY

ブレークポイントを検出したあとでブレークポイントを消去します (ブレークポイントを一時的に設定するときを使用します)。

**/TERMINATING**

プロセスがイメージを終了したときにデバッガがブレークします。デバッガに制御が戻り、単一プロセス・プログラムまたはマルチプロセス・プログラムの最後のイメージが終了すると、そのプロンプトを表示します。イメージが\$EXIT システム・サービスを実行し、その終了ハンドラがすべて実行されると、プロセスは終了します。/ACTIVATING 修飾子も参照してください。

**/UNALIGNED\_DATA**

(Alpha および I64 のみ) 境界に合っていないデータにアクセスすると、その命令の直後 (たとえば、ワード境界にないデータにアクセスするワード・ロード命令のあとで) デバッガがブレークします。

---

**説明**

ブレークポイントが検出されると、デバッガは次のいずれかの動作を行います。

1. ブレークポイント設定位置でプログラムの実行を中断する。
2. ブレークポイントの設定時に/AFTER を指定した場合、AFTER 回数をチェックする。指定された回数に達していないと実行が再開され、デバッガは残りのステップを実行しない。
3. ブレークポイントの設定時に WHEN 句を指定した場合、WHEN 句の式を評価する。式の値が偽であれば実行が再開され、デバッガは残りのステップに実行を移さない。
4. /SILENT が指定されていない場合、"break ... "メッセージを発行して、プログラム制御がブレークポイント設定位置にきたことを報告する。
5. ブレークポイントの設定時に/NOSOURCE も/SILENT も指定しないか、または SET STEP NOSOURCE を入力していない場合、実行を中断したソース・コード行を表示する。
6. ブレークポイントの設定時に DO 句を指定していれば、その DO 句内のコマンドを実行する。DO 句に GO コマンドが含まれていれば実行を続行し、デバッガは次のステップに移らない。
7. プロンプトを表示する。

プログラムの特定の記憶位置にブレークポイントを設定するには、SET BREAK コマンドでアドレス式を指定します。連続したソース行、命令クラス、またはイベントにブレークポイントを設定するには、SET BREAK コマンドで修飾子を指定します。通常はアドレス式か修飾子のどちらかを指定するだけでよく、両方を指定する必要はありません。ただし、/EVENT と/RETURN の場合は両方指定しなければなりません。

/LINE 修飾子は各ソース・コード行ごとにブレークポイントを設定します。

## SET BREAK

次の修飾子は命令クラスにブレークポイントを設定します。これらの修飾子と/LINEをいっしょに使用すると、デバッガはプログラムの実行中に各命令をトレースするので、実行速度が著しく遅くなります。

```
/BRANCH  
/CALL  
/INSTRUCTION  
/INSTRUCTION=(opcode, . . . ) (VAX のみ)  
/RETURN
```

次の修飾子は、ルーチンを呼び出したときに何が起こるかを決定します。

```
/INTO  
/[NO]JSB (VAX のみ)  
/OVER  
/[NO]SHARE  
/[NO]SYSTEM
```

次の修飾子は、ブレークポイントに達したときにどんな出力を表示するかを決定します。

```
/[NO]SILENT  
/[NO]SOURCE
```

次の修飾子は、ブレークポイントのタイミングと期間を決定します。

```
/AFTER:n  
/TEMPORARY
```

プログラム記憶位置の変更 (通常は変数値の変更) をモニタするには、/MODIFY 修飾子を使用します。

現在トレースポイントとして使用されている記憶位置をブレークポイントとして設定すると、トレースポイントは取り消されます。また、逆も同様です。

OpenVMS Alpha システムおよび I64 システムの場合、SET BREAK /UNALIGNED\_DATA コマンドは\$START\_ALIGN\_FAULT\_REPORT システム・サービス・ルーチンを呼び出します。デバッグしているプログラムに、同じ\$START\_ALIGN\_FAULT\_REPORT ルーチンへの呼び出しがある場合は、このコマンドを実行しないでください。プログラムを呼び出す前にこのコマンドを実行すると、この呼び出しは異常終了します。このコマンドを実行する前にプログラムが呼び出されると、境界に合っていないデータに対するブレークは設定されません。

ブレークポイントには、ユーザが定義するものと定義済みのものがあります。ユーザ定義のブレークポイントは、ユーザが SET BREAK コマンドで明示的に設定したブレークポイントです。定義済みのブレークポイントは、デバッグするプログラムの種類 (Ada あるいはマルチプロセスなど) によって異なりますが、デバッガの起動時に



自動的に設定されます。現在設定されているすべてのブレークポイントを表示するには、SHOW BREAK コマンドを使用します。定義済みのブレークポイントは定義済みのものとして表示されます。

ユーザ定義ブレークポイントと定義済みブレークポイントは、それぞれ別々に設定したり取り消したりします。たとえば、1つの記憶位置またはイベントに、ユーザ定義ブレークポイントと定義済みブレークポイントの両方を設定することができます。ユーザ定義ブレークポイントを取り消しても、定義済みブレークポイントは影響を受けません。逆も同様です。

#### 関連コマンド

```
(ACTIVATE,DEACTIVATE,SHOW,CANCEL) BREAK
CANCEL ALL
GO
(SET,SHOW) EVENT_FACILITY
SET STEP [NO]SOURCE
SET TRACE
SET WATCH
STEP
```

---

#### 例

1. DBG> SET BREAK SWAP\%LINE 12

このコマンドの場合、SWAP モジュールの行 12 でデバッガがブレークします。

2. DBG> SET BREAK/AFTER:3 SUB2

このコマンドの場合、SUB2 (ルーチン) が 3 度目以降に実行されたときにデバッガがブレークします。

3. DBG> SET BREAK/NOSOURCE LOOP1 DO (EXAMINE D; STEP; EXAMINE Y; GO)

このコマンドの場合、LOOP1 のアドレスでデバッガがブレークします。ブレークポイントでは、次のコマンドが順に実行されます。(1) EXAMINE D、(2) STEP、(3) EXAMINE Y、(4) GO。/NOSOURCE 修飾子を指定したので、ブレークポイントではソース・コードが表示されません。

4. DBG> SET BREAK ROUT3 WHEN (X > 4) DO (EXAMINE Y)

このコマンドの場合、X が 4 より大きい場合に ROUT3 ルーチンでデバッガがブレークします。ブレークポイントでは、EXAMINE Y コマンドが実行されます。WHEN 句の条件式の構文は、言語によって異なります。

5. 

```
DBG> SET BREAK/TEMPORARY 1440
DBG> SHOW BREAK
breakpoint at 1440 [temporary]
DBG>
```

このコマンドの場合、メモリ・アドレス 1440 にブレークポイントが一時的に設定されます。検出されるとこのブレークポイントは無効になります。

6. 

```
DBG> SET BREAK/LINE
```

このコマンドの場合、プログラムの実行中にソース行を検出するたびにその行の先頭でデバグがブレークします。

7. 

```
DBG> SET BREAK/LINE WHEN (X .NE. 0)
DBG> SET BREAK/INSTRUCTION WHEN (X .NE. 0)
```

この2つのコマンドの場合、X が 0 でないときにデバグがブレークします。最初のコマンドは、実行中に検出されたソース行の先頭で条件を満足しているかどうか調べます。2つ目のコマンドは、各命令で、条件を満足しているかどうかを調べます。WHEN 句の条件式の構文は、言語によって異なります。

8. 

```
DBG> SET BREAK/INSTRUCTION=ADDL3
```

(VAX のみ) このコマンドの場合、ADDL3 命令が実行される直前にデバグがブレークします。

9. 

```
DBG> SET BREAK/LINE/INTO/NOSHARE/NOSYSTEM
```

このコマンドの場合、各ソース行の先頭でデバグがブレークします。この行には、呼び出されたルーチン (/INTO) 内の行を含み、共用可能イメージ・ルーチン (/NOSHARE) 内とシステム・ルーチン (/NOSYSTEM) 内の行は含まれません。

10. 

```
DBG> SET BREAK/RETURN ROUT4
```

このコマンドの場合、ROUT4 ルーチンの復帰命令を実行する直前にデバグがブレークします。

11. 

```
DBG> SET BREAK/RETURN %LINE 14
```

このコマンドの場合、行 14 を含んでいるルーチンの復帰命令を実行する直前に、デバグがブレークします。このコマンド書式は、ルーチン内で現在実行が中断しており、そのルーチンの復帰命令でブレークポイントを設定したい場合に便利です。

12. 

```
DBG> SET BREAK/EXCEPTION DO (SET MODULE/CALLS; SHOW CALLS)
```

このコマンドの場合、例外がシグナル通知されるたびにデバグがブレークします。このブレークポイントでは、SET MODULE/CALLS コマンドと SHOW CALLS コマンドが実行されます。

13. 

```
DBG> SET BREAK/EVENT=RUN RESERVE, %TASK 3
```

このコマンドは2つのブレークポイントを設定します。それぞれ RESERVE タスクとタスク 3(タスク ID=3)に関連しています。関連したタスクが RUN 状態に移行するたびに、各ブレークポイントが検出されます。

14. all> SET BREAK/ACTIVATING

このコマンドの場合、マルチプロセス・プログラムのプロセスがデバッガの制御下に置かれるたびにデバッガがブレイクします。

---

## SET DEFINE

DEFINE コマンドの省略時の修飾子 (/ADDRESS , /COMMAND , /PROCESS\_SET , または /VALUE) を設定します。

---

### フォーマット

SET DEFINE *define-default*

---

### パラメータ

*define-default*

DEFINE コマンドに設定する省略時の値を指定します。次のいずれかのキーワード (それぞれ DEFINE コマンド修飾子に対応しています) を指定できます。

ADDRESS	それ以降の DEFINE コマンドは DEFINE/ADDRESS として扱われる。 これは省略時の設定である。
COMMAND	それ以降の DEFINE コマンドは DEFINE/COMMAND として扱われる。
PROCESS_SET	それ以降の DEFINE コマンドは DEFINE/PROCESS_SET として扱われる。
VALUE	それ以降の DEFINE コマンドは DEFINE/VALUE として扱われる。

---

### 説明

SET DEFINE コマンドは、それ以降の DEFINE コマンドの省略時の修飾子を設定します。SET DEFINE コマンドで指定するパラメータは、DEFINE コマンドの修飾子と同じ名前です。修飾子は、DEFINE コマンドがシンボルをアドレス、コマンド文字列、プロセス・リスト、または値のどれにバインドするかを決定します。

別の修飾子を指定すれば、1 つの DEFINE コマンドの実行中に現在の DEFINE コマンドの省略時の設定を上書きできます。現在の DEFINE コマンドの省略時の設定を表示するには、SHOW DEFINE コマンドを使用します。

#### 関連コマンド

DEFINE  
DEFINE/PROCESS\_SET  
DELETE  
SHOW DEFINE

## SHOW SYMBOL/DEFINED

---

例

```
DBG> SET DEFINE VALUE
```

SET DEFINE VALUE コマンドは、それ以降の DEFINE コマンドを DEFINE /VALUE として扱うことを指定します。

---

# SET EDITOR

EDIT コマンドによって起動するエディタを設定します。

---

## フォーマット

SET EDITOR *[command-line]*

---

## パラメータ

command-line

EDIT コマンドを使用したときに、指定したエディタをユーザのシステムで開始するためのコマンド行を指定します。

/CALLABLE\_EDT、/CALLABLE\_LSEDIT、または/CALLABLE\_TPU を使用した場合は、コマンド行を指定する必要はありません。これらの修飾子を使用しない場合は、EDIT コマンドを入力したときに SET EDITOR コマンド行で指定したエディタがサブプロセスとして作成されます。

コマンド行に/CALLABLE\_LSEDIT や/CALLABLE\_TPU は指定できますが、/CALLABLE\_EDT は指定できません。

---

## 修飾子

/CALLABLE\_EDT

EDIT コマンドを使用するときに EDT エディタの呼び出し可能なバージョンを起動することを指定します。コマンド行にこの修飾子を指定することはできません ("EDT" というコマンド行を使用します)。

/CALLABLE\_LSEDIT

(VAX のみ) EDIT コマンドを使用するときに DEC ランゲージ・センシティブ・エディタ (LSEDIT) の呼び出し可能なバージョンを起動することを指定します。コマンド行も指定すると、コマンド行は呼び出し可能な LSEDIT に渡されます。コマンド行を指定しないと、省略時のコマンド行は "LSEDIT" となります。

/CALLABLE\_TPU

EDIT コマンドを使用するときに DEC Text Processing ユーティリティ (DECTPU) の呼び出し可能なバージョンを起動することを指定します。コマンド行も指定すると、コマンド行は呼び出し可能な DECTPU に渡されます。コマンド行を指定しないと、省略時のコマンド行は "TPU" となります。

/START\_POSITION

/NOSTART\_POSITION (省略時の設定)

EDIT コマンドを入力したときに、指定されたコマンド行または省略時のコマンド行に/START\_POSITION 修飾子を付けるかどうかを制御します。現在は、DECTPU と DEC ランゲージ・センシティブ・エディタ (TPU または/CALLABLE\_TPU、および LSEDIT または/CALLABLE\_LSEDIT としてそれぞれ指定されます) だけがこの修飾子を指定できます。

/START\_POSITION 修飾子は、エディタのカーソルの最初の位置に影響を及ぼします。省略時の設定 (/NOSTART\_POSITION) では、エディタのカーソルがソース行 1 の先頭に置かれます。デバッガのソース表示でどの行が中央にあるか、また EDIT コマンドで行番号を指定するかどうかは関係ありません。/START\_POSITION を指定すると、カーソルは EDIT コマンドで番号で指定した行または現在のソース表示の中央にある行 (行番号を指定しない場合) に置かれます。

---

## 説明

SET EDITOR コマンドを使用すると、システムにインストールされているエディタを指定できます。通常、SET EDITOR コマンドのパラメータとして指定されたコマンド行は、サブプロセスとして作成され、実行されます。

VAX プロセッサの場合、EDT、LSEDIT、または DECTPU を使用すると、これらのエディタを効率よく起動できます。また、/CALLABLE\_EDT、/CALLABLE\_LSEDIT、または/CALLABLE\_TPU を指定すると、それぞれ EDT、LSEDIT、または DECTPU の呼び出し可能なバージョンを EDIT コマンドによって起動できます。LSEDIT と DECTPU の場合は、呼び出し可能なエディタが実行するコマンド行も指定できます。

Alpha プロセッサの場合、/CALLABLE\_EDT または/CALLABLE\_TPU は使用できませんが、/CALLABLE\_LSEDIT は使用できません。

## 関連コマンド

EDIT  
(SET,SHOW,CANCEL) SOURCE  
SHOW DEFINE

---

例

1. DBG> SET EDITOR '@MAIL\$EDIT ""'

このコマンドを指定しておくと、EDIT コマンドを入力したときに、コマンド行 '@MAIL\$EDIT ""' を作成し、その結果 MAIL で使用するのと同じエディタが開始されます。

2. DBG> SET EDITOR/CALLABLE\_TPU

このコマンドを指定しておくと、EDIT コマンドを入力したときに、TPU の省略時のコマンド行で呼び出し可能な DECTPU を開始します。

3. DBG> SET EDITOR/CALLABLE\_TPU TPU/SECTION=MYSECINI.TPU\$SECTION

このコマンドを指定しておくと、EDIT コマンドを入力したときに、コマンド行 TPU/SECTION=MYSECINI.TPU\$SECTION で呼び出し可能な DECTPU を開始します。

4. DBG> SET EDITOR/CALLABLE\_EDT/START\_POSITION

このコマンドを指定しておくと、EDIT コマンドを入力したときに、呼び出し可能な EDT を省略時のコマンド行 EDT で開始します。また、このコマンド行には /START\_POSITION 修飾子が付いていますから、編集はデバッガの現在のソース表示の中央にあるソース行から始まります。



---

# SET EVENT\_FACILITY

現在のイベント機能を設定します。

イベント機能は、Ada ルーチンか SCAN ルーチンを呼び出すプログラムまたは POSIX Threads サービスを使用するプログラムで使用できます。

---

## フォーマット

SET EVENT\_FACILITY *facility-name*

---

## パラメータ

*facility-name*

イベント機能を指定します。次のいずれかの *facility-name* キーワードを指定できます。

ADA	<p>イベント機能を ADA に設定すると、(SET, CANCEL) BREAK コマンドと (SET, CANCEL) TRACE コマンドは、汎用の低レベルのタスク・イベントだけでなく Ada 固有のイベントも認識できます (Ada イベントは、タスク・イベントと例外イベントから構成されます)。</p> <p>ただし、イベント機能を ADA に設定できるのは、メイン・プログラムが Ada で作成されている場合かプログラムが Ada ルーチンを呼び出す場合だけです。</p>
THREADS	<p>イベント機能を THREADS に設定すると、(SET, CANCEL) BREAK コマンドと (SET, CANCEL) TRACE コマンドは、汎用の低レベルのタスク・イベントだけでなく POSIX Threads 固有のイベントも認識できます。POSIX Threads イベントはすべてタスク (スレッド)・イベントです。</p> <p>イベント機能を THREADS に設定できるのは、共用可能イメージ CMASRTL が現在プログラム・プロセスの一部である場合 (そのイメージが SHOW IMAGE 表示に並んでいる場合) だけです。</p>
SCAN	<p>(VAX のみ) イベント機能を SCAN に設定すると、(SET, CANCEL) BREAK コマンドと (SET, CANCEL) TRACE コマンドが SCAN (パターン照合) イベントを認識します。</p> <p>イベント機能を SCAN に設定できるのは、メイン・プログラムが SCAN で作成されている場合か、プログラムが SCAN ルーチンを呼び出す場合だけです。</p>

---

## 説明

現在のイベント機能 (ADA, THREADS, または SCAN) は、SET BREAK/EVENT コマンドと SET TRACE/EVENT コマンドで設定できるイベントポイントを定義します。

イベント機能にリンクされているプログラムを開始すると、デバッガはプログラムの種類に適した方法でイベント機能を自動的に設定します。たとえば、メイン・プログラムが Ada(または SCAN) で作成されている場合、イベント機能は ADA(または SCAN) に設定されます。

SET EVENT\_FACILITY コマンドを指定するとイベント機能を変更できるので、デバッグ・コンテキストを変更できます。これは、複数言語プログラムを使用していて、現在設定されていないイベント機能に関連したルーチンをデバッグしたい場合に便利です。

VAX プロセッサでは、同じプログラムで Ada タスティング・サービスと POSIX Threads タスティング・サービスの両方を使用することはできません。すなわち、イベント機能は、ADA から SCAN に変換するか、POSIX Threads から SCAN に変換するか、またはその逆に変換するしかできません。

現在のイベント機能に関連しているイベント名を表示するには、SHOW EVENT\_FACILITY コマンドを使用します。これらのイベント名は、(SET, CANCEL)BREAK/EVENT コマンドと (SET, CANCEL)TRACE/EVENT コマンドで指定できるキーワードです。

#### 関連コマンド

```
(SET,CANCEL) BREAK/EVENT
(SET,CANCEL) TRACE/EVENT
SHOW BREAK
SHOW EVENT_FACILITY
SHOW IMAGE
SHOW TASK
SHOW TRACE
```

---

#### 例

```
DBG> SET EVENT_FACILITY THREADS
```

このコマンドは、THREADS (POSIX Threads) を現在のイベント機能として設定します。

---

## SET IMAGE

1 つまたは複数の共用可能イメージのシンボル情報をロードし、現在のイメージを設定します。

---

### フォーマット

```
SET IMAGE  [image-name[ . . . ]]
```

---

### パラメータ

*image-name*

共用可能イメージを指定します。ワイルドカード文字のアスタリスク(\*)は使用できません。代わりに/ALL 修飾子を使用してください。/ALL を指定する場合は、イメージ名は指定できません。

---

### 修飾子

/ALL

すべての共用可能イメージを設定することを指定します。

---

### 説明

SET IMAGE コマンドは指定された 1 つまたは複数のイメージのデータ構造を作成しますが、指定されたイメージ内のモジュールは設定しません。

現在のイメージとは現在のデバッグ・コンテキストです。アクセスするのは現在のイメージ内のシンボルです。SET IMAGE コマンドで 1 つのイメージだけを指定すると、そのイメージが現在のイメージになります。複数のイメージを並べて指定すると、そのリストで最後のイメージが現在のイメージになります。/ALL を指定すると、現在のイメージは変化しません。

SET IMAGE コマンドでイメージを設定する前に、DCL コマンドの LINK に /DEBUG 修飾子または/TRACEBACK 修飾子を指定してイメージをリンクしなければなりません。/NOTRACEBACK でリンクするとそのイメージのシンボル情報は使用できません。また、SET IMAGE コマンドで指定することもできなくなります。

DEFINE/ADDRESS コマンドと DEFINE/VALUE コマンドで作成した定義は、それらを作成したコンテキストのイメージが現在のイメージである場合だけ使用できます。SET IMAGE コマンドを使用して新しく現在のイメージを設定すると、これらの定義は一時的に使用できなくなります。ただし、DEFINE/COMMAND コマンドまたは DEFINE/KEY コマンドで作成した定義はすべてのイメージで使用できます。

### 関連コマンド

```
SET MODE [NO]DYNAMIC
(SET,SHOW,CANCEL) MODULE
(SHOW,CANCEL) IMAGE
```

---

### 例

```
DBG> SET IMAGE SHARE1
DBG> SET MODULE SUBR
DBG> SET BREAK SUBR
```

このコマンド列は、共用可能イメージ SHARE1 の SUBR モジュール内の SUBR ルーチンにブレークポイントを設定する方法を示しています。SET IMAGE コマンドはデバッグ・コンテキストを SHARE1 に設定します。SET MODULE コマンドは SUBR モジュールのシンボル・レコードを実行時シンボル・テーブル (RST) にロードします。SET BREAK コマンドは SUBR ルーチン内にブレークポイントを設定します。

---

# SET KEY

現在のキーの状態を設定します。

---

## 注意

このコマンドは、デバッガへの HP DECwindows Motif for OpenVMS ユーザ・インタフェースでは使用できません。

---

---

## フォーマット

SET KEY

---

## 修飾子

/LOG (省略時の設定)

/NOLOG

キー状態が設定されていることを示すメッセージを表示するかどうかを制御します。/LOG 修飾子を指定すると、メッセージが表示されます。/NOLOG 修飾子を指定すると、メッセージは表示されません。

/STATE[=state-name]

/NOSTATE (省略時の設定)

現在の状態として設定するキー状態を指定します。GOLD などの定義済みのキー状態や利用者定義状態を指定できます。状態名は英数字文字列です。/NOSTATE 修飾子を指定すると現在の状態は変わりません。

---

## 説明

このコマンドを使用する前に、キーパッド・モードにしなければなりません (SET MODE KEYPAD)。キーパッド・モードは省略時の設定です。

省略時の設定では、現在のキー状態は DEFAULT 状態です。ファンクション・キーを定義する場合、DEFINE/KEY/IF\_STATE コマンドを使用してキー定義に特定の状態名を割り当てることができます。キーを押したときにその状態が設定されていないと、その定義は処理されません。SET KEY/STATE コマンドを使用すれば、現在の状態を適切な状態に変更することができます。

状態を変更するキー (DEFINE/KEY/LOCK\_STATE/SET\_STATE で定義されたキー) を押しても、現在の状態を変更できます。

## SET KEY

### 関連コマンド

DELETE/KEY

DEFINE/KEY

SHOW KEY

---

### 例

```
DBG> SET KEY/STATE=PROG3
```

このコマンドはキー状態を PROG3 状態に変更します。これで、このキー状態に関連したキー定義を使用できるようになります。

---

# SET LANGUAGE

現在の言語を設定します。

---

## フォーマット

SET LANGUAGE *language-name*

---

## パラメータ

*language-name*

言語を指定します。

VAX プロセッサでは、次のいずれかのキーワードを指定できます。

ADA	BASIC	BLISS	C
C_PLUS_PLUS	COBOL	DIBOL	FORTRAN
MACRO	PASCAL	PLI	RPG
SCAN	UNKNOWN		

Alpha プロセッサでは、次のいずれかのキーワードを指定できます。

ADA	AMACRO	BASIC	BLISS
C	C_PLUS_PLUS	COBOL	FORTRAN
MACRO	MACRO64	PASCAL	PLI
UNKNOWN			

Intel Itanium プロセッサでは、次のいずれかのキーワードを指定できます。

AMACRO	BASIC	BLISS	C
C++	COBOL	Fortran	PASCAL
UNKNOWN			

MACRO-32 は、AMACRO コンパイラでコンパイルしなければなりません。

---

## 説明

デバッガを起動すると、メイン・プログラムを含んでいるモジュールを作成した言語が現在の言語になります。通常、このモジュールはイメージ遷移アドレスを含んでいます。メイン・プログラムと異なるソース言語で作成されたモジュールをデバッグする場合は、SET LANGUAGE コマンドを使用して言語を変更します。

現在の言語設定により、デバッガ・コマンド内で指定した名前、演算子、式の解析方法と解釈方法が決まります。さらに、変数の型、配列、およびレコード構文の種類、整数データの入力と表示の省略時の基数、大文字小文字の区別などの解析方法と解釈方法も決まります。また、言語設定により、ユーザ・プログラムに関連したデータをデバッガがどのように形式化し表示するかも決まります。

データ入力と表示の省略時の基数はほとんどの言語の場合、10 進数です。例外は BLISS と MACRO です。これらの言語での省略時の基数は 16 進数です。

コンパイラ生成型を持っていないプログラム記憶位置の省略時の型はロングワード整数です。これは、32 ビットのアプリケーションをデバッグするのに適しています。

64 ビット・アドレス空間を使用するアプリケーションをデバッグするには、省略時の型をクォードワードに変更することをお勧めします (OpenVMS I64 システムでは、省略時の型はクォードワードです)。SET TYPE QUADWORD コマンドを実行してください。

サポートされない言語でコーディングしたプログラムをデバッグするには SET LANGUAGE UNKNOWN コマンドを使用します。サポートされない言語に対してデバッガの有用性を最大にするために、SET LANGUAGE UNKNOWN は、一部のサポート言語に固有なものを含め大量のデータ形式と演算子をデバッガが受け入れるようにします。

SET LANGUAGE UNKNOWN は、「最も穏やかな」規則を使用しているため、簡単に迅速な予備手段となります。

言語に固有な演算子と構造に関するデバッガ・サポートについての詳しい説明は、ヘルプ・トピック Language\_Support を参照してください。

#### 関連コマンド

EVALUATE  
EXAMINE  
DEPOSIT  
SET MODE  
SET RADIX  
SET TYPE  
SHOW LANGUAGE

---

#### 例

1. DBG> SET LANGUAGE COBOL

このコマンドは現在の言語として COBOL を設定します。



2. DBG> SET LANGUAGE PASCAL

このコマンドは現在の言語として Pascal を設定します。

---

## SET LANGUAGE/DYNAMIC

自動言語設定の状態を切り替えます。

---

### フォーマット

SET LANGUAGE/DYNAMIC

---

### 説明

デバッガを起動したとき、現在の言語は、メイン・プログラムを含んでいるモジュールが書かれている言語に設定されています。これは通常はイメージ転送アドレスを含んでいるモジュールです。省略時の設定では、実行されているプログラムの有効範囲が別の言語で書かれているモジュールに変更されると、デバッガは現在の言語をそのモジュールの言語に変更します。

SET LANGUAGE/NODYNAMICコマンドを使用すると、デバッガが現在の言語を自動的に変更するのを禁止することができます。

#### 関連コマンド

SET LANGUAGE  
SHOW LANGUAGE

---

### 例

1. DBG> SET LANGUAGE/NODYNAMIC

このコマンドは、SET LANGUAGEまたはSET LANGUAGE/DYNAMICコマンドが入力されるまで、デバッガが現在の言語を変更するのを禁止します。

---

## SET LOG

SET OUTPUT LOG コマンドを入力したときに、デバッガがログを書き込むログ・ファイルを指定します。

---

### フォーマット

SET LOG *file-spec*

---

### パラメータ

*file-spec*

ログ・ファイルのファイル指定を示します。完全なファイル指定を行わないと、デバッガは、欠落しているフィールドの省略時のファイル指定は SYSSDISK:[ ]DEBUG.LOG であるとみなします。

指定したバージョン番号を持つファイルがすでに存在していると、デバッガは、そのファイルの最後にデバッグ・セッションのログを書き込みます。

---

### 説明

SET LOG コマンドはログ・ファイルの名前だけを決定します。すなわち、デバッガは指定されたファイルを作成したり書き込んだりしません。これを指定するのは SET OUTPUT LOG コマンドです。

SET OUTPUT LOG コマンドだけを入力して SET LOG コマンドを入力しないと、省略時の設定によってデバッガは SYSSDISK:[ ]DEBUG.LOG ファイルに書き込みます。

あるログ・ファイルにデバッガが書き込んでいるときに SET LOG コマンドで別のログ・ファイルを指定すると、デバッガはそれまで書き込んでいたファイルをクローズして、SET LOG コマンドで指定したファイルに書き込みを始めます。

#### 関連コマンド

SET OUTPUT LOG  
SET OUTPUT SCREEN\_LOG  
SHOW LOG

---

### 例

1. `DBG> SET LOG CALC`  
`DBG> SET OUTPUT LOG`

この例では、`SET LOG` コマンドが `SYSSDISK:[]CALC.LOG` をデバッグのログ・ファイルとして指定します。`SET OUTPUT LOG` を指定すると、ユーザの入力とデバッガの出力はそのファイルに書き込まれます。

2. `DBG> SET LOG [CODEPROJ]FEB29.TMP`  
`DBG> SET OUTPUT LOG`

この例では、`SET LOG` コマンドが `[CODEPROJ]FEB29.TMP` をログ・ファイルとして指定します。`SET OUTPUT LOG` コマンドを指定すると、ユーザの入力とデバッガの出力はそのファイルに書き込まれます。

---

## SET MARGINS

ソース行文字の左端位置と右端位置を指定します。この位置はソース行の表示が開始する位置と終了する位置です。

---

### 注意

このコマンドは、デバッガへの HP DECwindows Motif for OpenVMS ユーザ・インタフェースでは使用できません。

---

---

## フォーマット

```
SET MARGINS  rm
               lm:rm
               lm:
               :rm
```

---

## パラメータ

*lm*  
ソース・コードの行の表示を開始するソース行文字位置です (左マージン)。

*rm*  
ソース・コードの行の表示を終了するソース行文字位置です (右マージン)。

---

## 説明

SET MARGINS コマンドはソース行の表示だけを指定します。EXAMINE コマンドなどによる他のデバッガ出力の表示には影響しません。

SET MARGINS コマンドはソース・コードの表示を制御するのに便利です。たとえば、大きくインデントされている場合や長い行を右マージンで自動改行する場合などに使用できます。このような場合、左マージンを設定すればソース・ディスプレイ内にインデント用のスペースが入らなくなります。また右マージンの設定値を (省略時の値 255 から) 減らして行を短くし、自動改行されるのを防ぐことができます。

SET MARGINS コマンドは主に行 (非画面) モードのときに便利です。行モードでは、SET MARGINS コマンドはコマンド TYPE、EXAMINE/SOURCE、SEARCH、STEP を実行した結果や、ブレークポイント、トレースポイント、ウォッチポイントが検出された場合のソース行の表示を制御します。

画面モードでは、SET MARGINS コマンドは定義済みディスプレイ SRC などのソース・ディスプレイのソース行の表示には影響しません。TYPE コマンドや EXAMINE/SOURCE コマンドの出力はソース・ディスプレイに直接出力されるので、SET MARGINS コマンドはこれらの出力には影響しません。SET MARGINS コマンドは、出力または DO ディスプレイ内に示されるソース・コードの表示 (たとえば、STEP コマンドを実行したあと) にだけ影響します。ただし、PF1-PF3 を押して画面モードにした場合、このようなソース・コードは表示されません。このシーケンスは SET MODE SCREEN コマンドだけでなく SET STEP NOSOURCE コマンドも実行して冗長なソース・ディスプレイを削除するからです。

省略時には、デバッガがソース行の文字位置 1 から表示を開始します。これは、ユーザ端末画面では文字位置 9 に相当します。画面の最初の 8 文字は行番号用に予約されており、SET MARGINS コマンドでは操作できません。

番号を 1 つだけ指定すると、デバッガは左マージンを 1 に、右マージンを指定された数に設定します。

2 つの数をコロンで区切って指定すると、デバッガはコロンの左側の数を左マージンに、右側の数を右マージンに設定します。

数を 1 つだけ指定してそのあとにコロンを入力すると、デバッガはその数を左マージンに設定し、右マージンは変更しません。

コロンを入力してそのあとに 1 つの数を指定すると、デバッガはその数を右マージンに設定し、左マージンは変更しません。

#### 関連コマンド

```
SET STEP [NO]SOURCE
SHOW MARGINS
```

---

#### 例

```
1. DBG> SHOW MARGINS
   left margin: 1 , right margin: 255
DBG> TYPE 14
module FORARRAY
    14:          DIMENSION IARRAY(4:5,5), VECTOR(10), I3D(3,3,4)
DBG>
```

この例はソース・コード行の省略時のマージン設定を表示します (1 と 255)。

```
2.  DBG> SET MARGINS 39
    DBG> SHOW MARGINS
    left margin: 1 , right margin: 39
    DBG> TYPE 14
    module FORARRAY
      14:      DIMENSION IARRAY(4:5,5), VECTOR
    DBG>
```

この例は、右マージンの設定を 255 から 39 に変更するとソース・コード行の表示がどうなるかを示します。

```
3.  DBG> SET MARGINS 10:45
    DBG> SHOW MARGINS
    left margin: 10 , right margin: 45
    DBG> TYPE 14
    module FORARRAY
      14:  IMENSION IARRAY(4:5,5), VECTOR(10),
    DBG>
```

この例は、左右両方のマージンを変更後のソース・コード行の表示を示します。

```
4.  DBG> SET MARGINS :100
    DBG> SHOW MARGINS
    left margin: 10 , right margin: 100
    DBG>
```

この例は、左マージン設定はそのままにして右マージン設定だけを変更する方法を示します。

```
5.  DBG> SET MARGINS 5:
    DBG> SHOW MARGINS
    left margin: 5 , right margin: 100
    DBG>
```

この例は、右マージン設定はそのままにして左マージン設定だけを変更する方法を示します。

---

## SET MODE

デバッガ・モードを使用可能または使用不可能にします。

---

### フォーマット

SET MODE *mode*[, ... ]

---

### パラメータ

mode

デバッガ・モードを使用可能にするか使用不可能にするかを指定します。次のいずれかのキーワードを指定できます。

DYNAMIC

省略時の設定。動的モードを使用可能にします。動的モードを使用可能にすると、デバッガはプログラム実行中にモジュールとイメージを自動的に設定するので、SET MODULE または SET IMAGE コマンドを入力する必要がなくなります。特に、デバッガが実行に割り込みをかける (デバッガ・プロンプトが表示される) 場合は、実行が現在中断されているルーチンを含んでいるモジュールとイメージを必ず自動的に設定します。モジュールまたはイメージがすでに設定されている場合、動的モードはそのモジュールとイメージには何の影響も与えません。デバッガはモジュールまたはイメージを自動的に設定するときに情報メッセージを発行します。

NODYNAMIC

動的モードを使用不可能にします。モジュールまたはイメージが設定されると追加メモリが割り当てられるので、性能が下がらないよう動的モードを使用不可能にしたいことがあります (CANCEL MODULE コマンドと CANCEL IMAGE コマンドでモジュールとイメージを取り消すことによってメモリを解放することもできます)。動的モードを使用不可能にした場合、SET MODULE コマンドと SET IMAGE コマンドで明示的にモジュールとイメージを設定しなければなりません。

G\_FLOAT

式で指定した倍精度浮動小数点定数をデバッガが G\_FLOAT として解釈することを指定します (プログラム内で宣言された変数の解釈には影響しません)。

NOG\_FLOAT

省略時の設定。式で指定した倍精度浮動小数点定数をデバッガが D\_FLOAT として解釈することを指定します (プログラム内で宣言された変数の解釈には影響しません)。

INTERRUPT

マルチプロセス・プログラムのデバッグの際に有効です。いずれかのプロセスでプログラムの実行が停止されたとき、デバッガはイメージを実行していた他のすべてのプロセスの実行に割り込み、入力を求めるプロンプトを表示します。詳細は、『OpenVMS デバッガ説明書』を参照してください。

NOINTERRUPT

省略時の設定。マルチプロセス・プログラムのデバッグの際に有効です。いずれかのプロセスでプログラムの実行が停止されたとき、デバッガは他のプロセスに関しては何のアクションも行いません。



KEYPAD	省略時の設定。キーパッド・モードを使用可能にします。 このパラメータは、デバッグの HP DECwindows Motif for OpenVMS ユーザ・インタフェースでは使用できません。キーパッド・モードを使用可能にすると、数値キーパッドのキーを使用して定義済みの機能を実行できます。デバッグ・コマンド (画面モードで便利なもの)の中にはキーパッド・キーにバインドされているものもあります。(ヘルプ・トピック Keypad_Definitions_CI を参照してください。また、現在のキー定義を調べるには SHOW KEY コマンドを使用してください。) DEFINE/KEY コマンドを使用してキー機能を再定義することもできます。
NOKEYPAD	キーパッド・モードを使用不可能にします。このパラメータは、デバッグの HP DECwindows Motif for OpenVMS ユーザ・インタフェースでは使用できません。使用不可能にすると、数値キーパッドのキーは定義済み機能を果たさなくなります。また、DEFINE/KEY コマンドを使用してデバッグ機能をキーに割り当てることもできなくなります。
LINE	省略時の設定。可能ならば、プログラム記憶位置を行番号で表示することを指定します。
NOLINE	デバッグが、プログラム記憶位置を行番号ではなく、 <i>routine-name + byte-offSet</i> で表示することを指定します。
OPERANDS[=keyword]	(VAX のみ) 命令を調べるために EXAMINE コマンドを使用したとき、命令、オペランド、オペランドのアドレス、オペランドの内容を表示するように指定します。オペランドがレジスタでない場合に表示する情報のレベルは BRIEF キーワードと FULL キーワードのどちらを使用するかで異なります。省略時の設定は OPERANDS=BRIEF です。
NOOPERANDS	(VAX のみ) 省略時の設定。命令を調べるために EXAMINE コマンドを使用したときに、命令とオペランドを表示するように指定します。
SCREEN	画面モードを使用可能にします。このパラメータは、デバッグの HP DECwindows Motif for OpenVMS ユーザ・インタフェースでは使用できません。画面モードを使用可能にすると、端末の画面を長方形の領域に分割して、各領域にそれぞれ別のデータを表示することができます。画面モードでは、省略時の設定 (行単位の非画面モード) より多くの情報をもっと都合よく表示できます。また、定義済みのディスプレイを使用することも、自分で定義することもできます。
NOSCREEN	省略時の設定。画面モードを使用不可能にします。このパラメータは、デバッグの HP DECwindows Motif for OpenVMS ユーザ・インタフェースでは使用できません。
SCROLL	省略時の設定。スクロール・モードを使用可能にします。 このパラメータは、デバッグの HP DECwindows Motif for OpenVMS ユーザ・インタフェースでは使用できません。スクロール・モードを使用可能にすると、画面モード出力や DO ディスプレイは、生成されるたびに行ごとにスクロールすることによって更新されます。
NOSCROLL	スクロール・モードを使用不可能にします。このパラメータは、デバッグの HP DECwindows Motif for OpenVMS ユーザ・インタフェースでは使用できません。スクロール・モードを使用不可能にすると、画面モード出力や DO ディスプレイは、コマンド 1 つにつき 1 回だけ更新されます (出力が生成されるたびに 1 行ずつ更新されるわけではありません)。スクロール・モードを使用不可能にすると、発生する画面更新回数が少なくなり、処理速度が遅い端末では便利です。

## SET MODE

SYMBOLIC	省略時の設定。シンボリック・モードを使用可能にします。シンボリック・モードを使用可能にすると、(可能ならば) アドレス式で示した記憶位置をシンボルで表示し、(可能ならば) 命令オペランドをシンボルで表示します。EXAMINE/NOSYMBOLIC を指定すると、EXAMINE コマンドの実行中、SET MODE SYMBOLIC を上書きできます。
NOSYMBOLIC	シンボリック・モードを使用不可能にします。シンボリック・モードを使用不可能にすると、デバッガは数値アドレスをシンボル化しません (デバッガは数字を名前に変換しません)。これは、シンボリック名ではなく数値アドレスを表示するときに便利です (これらのアドレスに対応するシンボリック名が存在する場合)。シンボリック・モードを使用不可能にするとデバッガは数値から名前に変換する必要がないので、コマンド処理はいくらも速くなります。EXAMINE/SYMBOLIC を使用すると、EXAMINE コマンドの実行中、SET MODE NOSYMBOLIC を上書きできます。
WAIT	省略時の設定。WAIT モードを使用可能にします。WAIT モードでは、デバッガは制御下にあるすべてのプロセスが停止するのを待ってから、新しいコマンドの入力を求めるプロンプトを表示します。詳細は、『OpenVMS デバッガ説明書』を参照してください。
NOWAIT	WAIT モードを使用不可能にします。NOWAIT モードでは、デバッガは一部またはすべてのプロセスが実行中であっても、ただちに新しいコマンドの入力を求めるプロンプトを表示します。

---

## 説明

SET MODE コマンドについて詳しくは、パラメータの説明を参照してください。これらのモードの省略時の値はどの言語の場合も同じです。

### 関連コマンド

EVALUATE  
EXAMINE  
DEFINE/KEY  
DEPOSIT  
DISPLAY  
(SET,SHOW,CANCEL) IMAGE  
(SET,SHOW,CANCEL) MODULE  
SET PROMPT  
(SET,SHOW,CANCEL) RADIX  
(SET,SHOW) TYPE  
(SHOW,CANCEL) MODE  
SYMBOLIZE

---

例

```
DBG> SET MODE SCREEN
```

このコマンドはデバッガを画面モードにします。

---

## SET MODULE

現在のイメージ内のモジュールのシンボル・レコードをそのイメージの実行時シンボル・テーブル (RST) にロードします。

---

### 注意

現在のイメージはメイン・イメージ (省略時の設定) か、または SET IMAGE コマンドで現在のイメージとして設定されたイメージのどちらかです。

---

---

## フォーマット

```
SET MODULE  [module-name], ... ]]
```

---

## パラメータ

*module-name*

シンボル・レコードを RST にロードする現在のイメージのモジュールを指定します。ワイルドカード文字のアスタリスク(\*)は使用できません。代わりに /ALL 修飾子を使用してください。/ALL または /CALLS を指定する場合は、モジュール名は指定できません。

---

## 修飾子

/ALL

現在のイメージ内のすべてのモジュールのシンボル・レコードを RST にロードすることを指定します。

/CALLS

呼び出しスタック上に現在ルーチンを持っているすべてのモジュールを設定します。モジュールがすでに設定されている場合、/CALLS はそのモジュールに何の影響も与えません。

/RELATED (省略時の設定)

/NORELATED

(Ada プログラムに適用される。) 指定されたモジュールに WITH 句またはサブユニット関係によって関連づけられたモジュールのシンボル・レコードを RST にロードするかどうかを制御します。ロードをすると、Ada プログラムのソース・コード内で参照したのと全く同じデバッガ・コマンド内の関連するモジュールで宣言されている名前を参照できます。

---

説明

プログラム内で宣言したシンボルをデバッガが認識し正しく解釈するには、シンボル・レコードが RST 内に存在しなければなりません。モジュールのシンボル・レコードを RST にロードするプロセスは、モジュールの設定といえます。

デバッガは、その起動時に遷移アドレスを含んでいるモジュール (メイン・プログラム) を設定します。省略時の設定では、動的モードが使用可能です (SET MODE DYNAMIC)。このため、デバッガはプログラムが実行するときにモジュール (およびイメージ) を自動的に設定して、あとで必要なときにシンボルを参照できるようにします。特に、実行が中断している場合は必ず、実行を中断しているルーチンを含んでいるモジュールとイメージをデバッガは設定します。Ada プログラムの場合は、モジュールが動的に設定されると、省略時には関係するモジュールも自動的に設定され、正しいシンボルにアクセスできるよう (可視状態) にします。

動的モードは、参照する必要のあるシンボルのほとんどにアクセスできるようにします。まだ設定されていないモジュール内のシンボルを参照しなければならないときは、次のようにします。

- モジュールが現在のイメージ内にあるときは、SET MODULE コマンドを使用して、シンボルが定義されているモジュールを設定します。
- モジュールが別のイメージ内にあるときには、SET IMAGE コマンドを使用してそのイメージを現在のイメージにし、次に SET MODULE コマンドを使用してシンボルが定義されているモジュールを設定します。

動的モードが使用不可能にしてあるとき (SET MODE NODYNAMIC) には、遷移アドレスを含んでいるモジュールだけが自動的に設定されます。それ以外のモジュールは明示的に設定しなければなりません。

SET IMAGE コマンドを使用して新しいイメージを設定する場合、すでに設定されているモジュールはすべて設定されたままです。ただし、アクセスできるのは、現在のイメージで設定されているモジュール内のシンボルだけです。他のイメージで設定されているモジュール内のシンボルに一時的にアクセスすることはできません。

動的モードが使用可能な場合、RST の増加するサイズに対応できるように、メモリは自動的に割り当てられます。動的モードが使用不可能な場合、ユーザがモジュールまたはイメージを設定したために必要になった余分なメモリをデバッガが自動的に割り当てます。

SET SCOPE コマンド内のパラメータがまだ設定されていないモジュール内のプログラム記憶位置を指定すると、SET SCOPE コマンドがそのモジュールを設定します。

Ada プログラムに固有な情報については、ヘルプ・トピック Language\_Support Ada を参照してください。

## 関連コマンド

(SET,SHOW,CANCEL) IMAGE  
 SET MODE [NO]DYNAMIC  
 (SHOW) MODULE

## 例

1. DBG> SET MODULE SUB1

このコマンドは SUB1 モジュールを設定します (SUB1 モジュールのシンボル・レコードを RST にロードします)。

2. DBG> SET IMAGE SHARE3  
 DBG> SET MODULE MATH  
 DBG> SET BREAK %LINE 31

この例では、SET IMAGE コマンドが共用可能イメージ SHARE3 を現在のイメージにします。SET MODULE コマンドは SHARE3 イメージ内の MATH モジュールを設定します。SET BREAK コマンドは MATH モジュールの行 31 にブレークポイントを設定します。

3. DBG> SHOW MODULE/SHARE

module name	symbols	language	size
FOO	yes	MACRO	432
MAIN	no	FORTTRAN	280
...			
SHARE\$DEBUG	no	Image	0
SHARE\$LIBRTL	no	Image	0
SHARE\$MTHRTL	no	Image	0
SHARE\$SHARE1	no	Image	0
SHARE\$SHARE2	no	Image	0
total modules: 17.		bytes allocated: 162280.	

DBG> SET MODULE SHARE\$SHARE2  
 DBG> SHOW SYMBOL \* IN SHARE\$SHARE2

この例では、SHOW MODULE/SHARE コマンドが、現在のイメージとすべての共用可能イメージ (共用可能イメージの名前の前には"SHARE\$"が付きます) 内のすべてのモジュールを表示します。SET MODULE SHARE\$SHARE2 コマンドは共用可能イメージ・モジュール SHARE\$SHARE2 を設定します。SHOW SYMBOL コマンドは共用可能イメージ SHARE2 内で定義された任意のユニバーサル・シンボルを表示します。詳しい説明は SHOW MODULE/SHARE コマンドを参照してください。

---

# SET OUTPUT

デバッグ出力オプションを使用可能または使用不可能にします。

---

## フォーマット

SET OUTPUT *output-option*[, ... ]

---

## パラメータ

*output-option*

出力オプションを使用可能にするか使用不可能にするかを指定します。次のいずれかのキーワードを指定できます。

LOG	デバッグの入力と出力をログ・ファイルに記録することを指定します。SET LOG コマンドでログ・ファイルを指定すると、デバッグはそのファイルに書き込みます。指定しない場合には、省略時の設定により、デバッグは SYS\$DISK[:]DEBUG.LOG に書き込みます。
NOLOG	省略時の設定。デバッグの入力と出力をログ・ファイルに記録しないことを指定します。
SCREEN_LOG	画面モードの場合、画面が更新されたときに画面の内容をログ・ファイルに記録することを指定します。画面の内容を記録するには、SET OUTPUT LOG も指定しなければなりません。ログ・ファイルの指定については LOG オプションの説明を参照してください。
NOSCREEN_LOG	省略時の設定。画面モードの場合に、画面の内容をログ・ファイルに記録しないことを指定します。
TERMINAL	

---

### 注意

このパラメータは、デバッグへの HP DECwindows Motif for OpenVMS ユーザ・インタフェースでは使用できません。

---

省略時の設定。デバッグ出力を端末に表示することを指定します。

## SET OUTPUT

NOTERMINAL

---

### 注意

---

このパラメータは、デバッガへの HP  
DECwindows Motif for OpenVMS ユー  
ザ・インタフェースでは使用できませ  
ん。

---

VERIFY

デバッガ出力を診断メッセージを除いて端末に表示しないことを指定  
します。

コマンド・プロシージャまたは DO 句から実行している各入力コマン  
ド文字列を現在の出力装置へエコーバックすることを指定します。現  
在の出力装置は省略時の設定では SYS\$OUTPUT (ユーザの端末) で  
すが、論理名 DBG\$OUTPUT によって再定義できます。

NOVERIFY

省略時の設定。コマンド・プロシージャまたは DO 句から実行してい  
る各入力コマンド文字列をデバッガが表示しないように指定します。

---

## 説明

デバッガ出力オプションは、コマンドへのデバッガの応答を表示し記録する方法を制  
御します。SET OUTPUT コマンドについて詳しくは、パラメータの説明を参照して  
ください。

### 関連コマンド

@ (実行プロシージャ)  
(SET,SHOW) ATSIGN  
(SET,SHOW) LOG  
SET MODE SCREEN  
SHOW OUTPUT

---

## 例

DBG> SET OUTPUT VERIFY,LOG,NOTERMINAL

このコマンドはデバッガが次のいずれかのアクションをとることを指定します。

- コマンド・プロシージャまたは DO 句から実行している各コマンド文字列を出力  
する (VERIFY)
- デバッガ出力とユーザ入力をログ・ファイルに記録する (LOG)
- 診断メッセージを除き、端末には出力を表示しない (NOTERMINAL)



---

# SET PROCESS

可視プロセスの設定，または動的プロセスの設定の許可/禁止を行います。

マルチプロセス・プログラムをデバッグする場合のみ使用 (保持デバッガのみ)。

---

## フォーマット

SET PROCESS [*process-spec*[, ... ]]

---

## パラメータ

*process-spec*

現在デバッガの制御下にあるプロセスを指定します。次のいずれかの形式で指定します。

[%PROCESS\_NAME] *process-name*

スペースや小文字を含まないプロセス名。プロセス名にはワイルドカード文字(\*)を含めることができる。

[%PROCESS\_NAME] "*process-name*"

スペースまたは小文字を含むプロセス名。二重引用符(")の代わりに、一重引用符(') 使用することもできる。

%PROCESS\_PID *process\_id*

プロセス識別子 (PID，16 進数)。

[%PROCESS\_NUMBER] *process-number*  
(または%PROC *process-number*)

デバッガの制御下に入ったときにプロセスに割り当てられた番号。新しい番号は、1 から順番に各プロセスに割り当てられる。EXIT コマンドまたは QUIT コマンドによってプロセスが終了した場合、そのデバッグ・セッション中にその番号が再割り当てされることがある。プロセス番号は SHOW PROCESS コマンドの実行で表示される。プロセスは、組み込みシンボル%PREVIOUS\_PROCESS および%NEXT\_PROCESS によってインデックスづけできるように、循環リスト内に順序づけされる。

*process-set-name*

DEFINE/PROCESS\_SET コマンドで定義された、プロセスのグループを表すシンボル。

%NEXT\_PROCESS

デバッガの循環プロセス・リスト中で可視プロセスの次のプロセス。

%PREVIOUS\_PROCESS

デバッガの循環プロセス・リスト中で可視プロセスの前のプロセス。

%VISIBLE\_PROCESS

シンボル、レジスタ値、ルーチン呼び出し、ブレークポイントなどの検索時に現在のコンテキストになっているスタック、レジスタ・セット、およびイメージを持つプロセス。

すべてのプロセスを指定するためにワイルドカード文字のアスタリスク(\*)を使用することもできます。/[NO]DYNAMIC 修飾子とともに、プロセスを指定しないでください。

---

## 修飾子

/DYNAMIC (省略時の設定)

/NODYNAMIC

動的プロセス設定を許可するか禁止するかを制御します。動的プロセス設定を許可する (/DYNAMIC) と、デバッガが実行を中断しそれに対するプロンプトを表示するたびに、実行が中断されたプロセスが自動的に可視プロセスになります。動的プロセス設定を禁止する (/NODYNAMIC) と可視プロセスはそのまま、別のプロセスを SET PROCESS/VISIBLE コマンドで指定するまで変わりません。

/VISIBLE

指定されたプロセスを可視プロセスにします。この結果、デバッグ・コンテキストは指定されたプロセスに切り換えられ、シンボルの検索やブレークポイントの設定などはそのプロセスのコンテキスト内で実行されます。/VISIBLE を使用する場合は、プロセスを 1 つだけ指定しなければなりません。

---

## 説明

SET PROCESS コマンドは、可視プロセスの設定、現在のプロセス・セットの定義、または可視プロセスの定義を行います。

省略時の設定では、コマンドは可視プロセスのコンテキストで実行されます。(現在コンテキストをデバッグ中であるプロセス)。シンボルの検索やブレークポイントの設定などは、可視プロセスのコンテキスト内で実行されます。

省略時の設定では、動的プロセス設定が許可され、/[NO]DYNAMIC で制御されます。また、動的プロセス設定を許可すると、デバッガがプログラムの実行を中断し、それに対するプロンプトを表示するたびに、実行が中断されたプロセスが自動的に可視プロセスになります。

関連コマンド

CALL  
EXIT  
GO  
QUIT  
SHOW PROCESS  
STEP

---

**例**

```
1. all> SET PROCESS TEST_Y
all> SHOW PROCESS
  Number  Name  Hold  State  Current PC
*    2 TEST_Y   YES  break  PROG\%LINE 71
all>
```

この SET PROCESS TEST\_Y コマンドは、TEST\_Y プロセスを可視プロセスにします。省略時の設定により、SHOW PROCESS コマンドは可視プロセスに関する情報を表示します。

---

## SET PROMPT

デバッガ・プロンプト文字列をユーザが指定するものに変更します。

---

### フォーマット

SET PROMPT *[prompt-parameter]*

---

### パラメータ

prompt-parameter

新しいプロンプト文字列を指定します。文字列にスペース，セミコロン(;)，または小文字が含まれる場合，それを二重引用符(")または一重引用符(')で囲まなければなりません。文字列を指定しないと，現在のプロンプト文字列のままです。

省略時の設定では，シングル・プロセス・プログラムをデバッグしている場合，プロンプト文字列はDBG>です。

省略時の設定では，マルチプロセス・プログラムをデバッグしている場合，プロンプト文字列は現在のプロセス・セットの名前の後に右山括弧(>)を続けたものになります。マルチプロセス・プログラムをデバッグしているときは，SET PROMPT コマンドを使用するべきではありません。

---

### 修飾子

/POP

/NOPOP (省略時の設定)

(VWS を実行するワークステーションにのみ適用されます。) /POP 修飾子を使用すると，デバッガが入力促すプロンプトを表示したときにデバッガ・ウィンドウは他のウィンドウより上にポップアップし，キーボードに接続されます。/NOPOP 修飾子はこの動作を禁止します。デバッガ・ウィンドウは他のウィンドウにポップアップされず，デバッガが入力促すプロンプトを表示してもキーボードには自動的に接続されません。

---

### 説明

SET PROMPT コマンドを使用すると，デバッガのプロンプト文字列をユーザが指定するものに変更できます。

マルチプロセス・プログラムをデバッグしている場合， SET PROMPT コマンドを使用しないでください。

ワークステーションでデバッガを使用している場合，/[NO]POP を指定すると，デバッガが入力を促すプロンプトを表示したときにデバッガ・ウィンドウを他のウィンドウより上にポップアップするかどうかを制御します。

関連コマンド

(SET,SHOW) PROCESS

---

## 例

```
1. DBG> SET PROMPT "$ "  
   $ SET PROMPT "d b g : "  
   d b g : SET PROMPT "DBG> "  
   DBG>
```

この例では， SET PROMPT コマンドを連続して実行した結果，デバッガのプロンプトが“DBG>”から“\$”へ，次に“d b g :”へ変わったあとに，さらに“DBG>”へ戻ります。

---

## SET RADIX

整数データを入力するときと表示するときの基数を設定します。/OVERRIDEとともに指定すると、すべてのデータが、指定された基数の整数データとして表示されます。

---

### フォーマット

SET RADIX *radix*

---

### パラメータ

*radix*

設定する基数を指定します。次のいずれかのキーワードを指定できます。

BINARY	基数を 2 進数に設定する。
DECIMAL	基数を 10 進数に設定する。これは、BLISS、MACRO-32、および MACRO-64 (Alpha および I64 のみ) を除くすべての言語における省略時の設定である。
DEFAULT	基数を言語の省略時の設定にする。
OCTAL	基数を 8 進数に設定する。
HEXADECIMAL	省略時の基数を 16 進数に設定する。これは、BLISS、MACRO-32、および MACRO-64 (Alpha および I64 のみ) の省略時の設定である。

---

### 修飾子

/INPUT

入力基数 (整数データを入力するときの基数) だけを、指定された基数に設定します。

/OUTPUT

出力基数 (整数データを表示するときの基数) だけを、指定された基数に設定します。

/OVERRIDE

すべてのデータを指定された基数の整数データとして表示します。

---

### 説明

現在の基数設定は、デバッガが次の状況で整数データを解釈し、表示する方法に影響を与えます。

- アドレス式または言語式で指定する整数データ

- EXAMINE コマンドおよび EVALUATE コマンドによって表示される整数データ

データの入力と表示の省略時の基数はほとんどの言語の場合どちらも 10 進数です。例外は BLISS と MACRO です。これらの言語での省略時の基数は 16 進数です。

SET RADIX コマンドを使用して、データの入力または表示の新しい基数 (入力基数と出力基数それぞれ) を指定できます。

修飾子を指定しないと、SET RADIX コマンドは入力基数と出力基数の両方を変更します。/INPUT または /OUTPUT を指定すると、このコマンドは、それぞれ入力基数または出力基数を変更します。

SET RADIX/OVERRIDE を使用すると、変更されるのは出力基数だけですが、すべてのデータ (整数型のデータだけでなく) が、指定された基数の整数データとして表示されます。

SET RADIX コマンドは、/OVERRIDE とともに使用する以外は、非整数型の値 (実数型の値や列挙型の値など) の解釈や表示には影響しません。

コマンド EVALUATE、EXAMINE、DEPOSIT には、基数修飾子 (/BINARY、/HEXADECIMAL など) があります。これらを指定すると、そのコマンドの実行中は SET RADIX または SET RADIX/OVERRIDE で設定した基数を上書きできます。

また、組み込みシンボル %BIN、%DEC、%HEX、%OCT をアドレス式と言語式で使用する、整数リテラルを 2 進数、10 進数、16 進数または 8 進数の基数で解釈するように指定できます。

#### 関連コマンド

DEPOSIT  
EVALUATE  
EXAMINE  
(SET,SHOW,CANCEL) MODE  
(SHOW,CANCEL) RADIX

---

例

1. DBG> SET RADIX HEX

このコマンドは、基数を 16 進数に設定します。これは、省略時の設定では、整数データが 16 進数の基数で解釈され表示されることを意味しています。

2. DBG> SET RADIX/INPUT OCT

このコマンドは、入力基数を 8 進数に設定します。これは、省略時の設定では、入力された整数データは 8 進数の基数で解釈されることを意味しています。

3. DBG> SET RADIX/OUTPUT BIN

このコマンドは、出力基数を 2 進数に設定します。これは、省略時の設定では、整数データが 2 進数の基数で表示されることを意味しています。

4. DBG> SET RADIX/OVERRIDE DECIMAL

このコマンドは、上書き基数を 10 進数に設定します。これは、省略時の設定では、すべてのデータ (整数型のデータに限らず) が 10 進整数データとして表示されることを意味しています。



---

## SET SCOPE

パス名接頭識別子を指定しない場合に、デバッガがシンボル (変数名, ルーチン, 行番号など) を検索する方法を設定します。

---

### フォーマット

SET SCOPE *location*[, ... ]

---

### パラメータ

*location*

パス名接頭識別子を使用しないで指定したシンボルを解釈するときに使用するプログラム領域 (有効範囲) を示します。/CURRENT または /MODULE を指定しない場合、記憶位置は次のいずれかを指定できます。

*path-name prefix*

パス名接頭識別子で示される有効範囲を指定します。パス名接頭識別子は、1 つまたは複数のネストしているプログラムの要素 (モジュール, ルーチン, ブロックなど) の名前で構成されます。これらの名前はバックスラッシュ(\)で区切られています。パス名接頭識別子が 2 つ以上の名前で構成される場合、(\)の左にはネスト要素を入れ、右にはネストされた要素を入れます。共通パス名接頭識別子の形式は、*module(\)routine(\)block(\)*です。

モジュール名だけを指定し、その名前がルーチン名と同じである場合は、/MODULE を使用します。そうしないと、そのルーチンを指定しているものとデバッガがみなします。

*n*

呼び出しスタックの *n* レベル下にあるルーチンによって示される有効範囲を指定します (*n* は 10 進整数です)。整数によって指定される有効範囲は、プログラムの実行に伴い動的に変化します。値 0 は、現在実行中のルーチンを示し、値 1 は、そのルーチンの呼び出し元を示すというように呼び出しスタックを下っていきます。省略時の有効範囲検索リストは、0, 1, 2, ..., *n* です。ここで *n* は呼び出しスタック内の呼び出し数です。

\ (バックスラッシュ)

グローバルな有効範囲 — すなわち、グローバル・シンボルの存在が認識されているすべてのプログラム記憶位置の集合を指定します。グローバル・シンボルの定義と宣言方法は、言語によって異なります。

記憶位置パラメータを 2 個以上指定すると、有効範囲検索リストが設定されます。デバッガは最初のパラメータを使用してシンボルを解釈できないと、次のパラメータを使用し、正しくシンボルを解釈するか指定されたパラメータがなくなるまで、リストに並んでいる順にパラメータを使用します。

---

修飾子**/CURRENT**

省略時の検索リストに似た有効範囲検索リスト  $(0, 1, 2, \dots, n)$  を設定します。ただし、コマンド・パラメータとして指定された数値の有効範囲で開始します。有効範囲 0 は PC 有効範囲であり、 $n$  は、呼び出しスタック内の呼び出し数です。

SET SCOPE/CURRENT を使用する場合、次の規則と動作に注意してください。

- コマンドを入力するときに省略時の有効範囲検索リストが有効でなければならない。省略時の有効範囲検索リストを復元するには、CANCEL SCOPE コマンドを入力する。
- 指定するコマンド・パラメータは、1 つだけであり、0 から  $n$  までの 10 進整数でなければならない。
- 画面モードの場合、コマンドは定義済みのソース・ディスプレイ SRC、定義済みの機械語命令ディスプレイ INST、定義済みのレジスタ・ディスプレイ REG をそれぞれ更新して、シンボル検索を開始する呼び出しスタック上のルーチンを表示する。
- プログラムの実行を再開すると、省略時の有効範囲検索リストが復元される。

**/MODULE**

コマンド・パラメータとして指定された名前がモジュール名であり、ルーチン名ではないことを指定します。モジュール名をコマンド・パラメータとして指定し、そのモジュール名がルーチン名と同じであることを指定する場合は、/MODULE を使用しなければなりません。

---

説明

省略時の設定では、デバッガは、有効範囲検索リスト  $0, 1, 2, \dots, n$  に従い、パス名接頭識別子が指定されていないシンボルを検索します。 $n$  は、呼び出しスタック内に存在する呼び出し数です。この有効範囲検索リストは、現在の PC 値に基づいており、プログラムの実行に伴い動的に変化します。省略時の有効範囲検索リストは次のように指定します。すなわち、EXAMINE X などのシンボル検索は最初に、現在実行中のルーチン (有効範囲 0 すなわち PC 有効範囲ともいいます) で X を検索し、そこで X が見つからなければ、そのルーチンの呼び出し元 (有効範囲 1) で検索するというように呼び出しスタックのレベルを下げていくことを指定します。有効範囲  $n$  内で X が見つからなければ、実行時シンボル・テーブル (RST) の残り、すなわちすべての設定されたモジュールとグローバル・シンボル・テーブル (GST) を検索します。

ほとんどの場合は、この省略時の有効範囲検索リストを使用すれば、言語規則に準拠した自然な方法であいまいさを解消できます。ただし、何回実行しても定義されているシンボルにアクセスできないときは、次のいずれかの方法を使用します。

- パス名接頭識別子を付けてシンボルを指定する。パス名接頭識別子は、シンボルを別々のものとして指定するのに必要な、ネストしているプログラム・ユニット (たとえば、*module\routine\block*) で構成される。例を次に示す。

```
DBG> EXAMINE MOD4\ROUT3\X
DBG> TYPE MOD4\27
```

- SET SCOPE コマンドを使用してシンボル検索用の新しい省略時の有効範囲 (すなわち有効範囲検索リスト) を設定する。この結果、次の例に示すように、パス名接頭識別子を付けずにシンボルだけを指定できる。

```
DBG> SET SCOPE MOD4\ROUT3
DBG> EXAMINE X
DBG> TYPE 27
```

SET SCOPE コマンドは、シンボルを指定するたびにパス名を使用しなければならない場合に便利です。

省略時の有効範囲検索リストを復元するには、CANCEL SCOPE コマンドを使用します。

省略時の有効範囲検索リストが有効であれば、SET SCOPE/CURRENT コマンドを使用して、シンボル検索を有効範囲 0 以外の数値 (呼び出しスタックから数えた数値) の有効範囲 (たとえば、有効範囲 2) で開始することを指定できます。

SET SCOPE コマンドを使用し、/CURRENT を指定しないと、デバッガは明示的に指定したプログラム記憶位置だけを検索します。また、SET SCOPE コマンドで設定した有効範囲または有効範囲検索リストは、省略時の有効範囲検索リストを復元するか新たに SET SCOPE コマンドを指定するまで有効です。ただし、/CURRENT を指定すれば、プログラムの実行が再開されるたびに省略時の有効範囲検索リストが復元されます。

SET SCOPE コマンドは、/CURRENT が指定されたときだけ画面モード・ソース・ディスプレイまたは機械語命令ディスプレイを更新します。

SET SCOPE コマンドで指定した名前がモジュールとルーチンの両方の名前である場合、デバッガは有効範囲をルーチンに設定します。このような場合は、SET SCOPE/MODULE コマンドを使用すれば、有効範囲をモジュールに設定できます。

SET SCOPE コマンドで指定したモジュール名を持つモジュールがまだ設定されていないと、デバッガはそのモジュールを設定します。ただし、モジュールだけを設定する場合は、SET SCOPE コマンドではなく SET MODULE コマンドを使用してください。SET SCOPE コマンドを使用すると、現在の有効範囲検索リストを妨害する可能性があるためです。

## 関連コマンド

CANCEL ALL  
 SEARCH  
 SET MODULE  
 (SHOW,CANCEL) SCOPE  
 SHOW SYMBOL  
 SYMBOLIZE  
 TYPE

## 例

1. DBG> EXAMINE Y  
    %DEBUG-W-NOUNIQUE, symbol 'Y' is not unique  
 DBG> SHOW SYMBOL Y  
    data CHECK IN\Y  
    data INVENTORY\COUNT\Y  
 DBG> SET SCOPE INVENTORY\COUNT  
 DBG> EXAMINE Y  
    INVENTORY\COUNT\Y: 347.15  
 DBG>

この例では、最初の EXAMINE Y コマンドは、シンボル Y が何回も定義されており、現在の有効範囲検索リストでは解消できないことを示しています。SHOW SYMBOL コマンドは、宣言されているシンボル Y をすべて表示します。SET SCOPE コマンドは、INVENTORY モジュールの COUNT ルーチンでパス名接頭識別子のないシンボルを検索するようにデバッガに指示します。この結果、その次の EXAMINE コマンドは、Y を明確に解釈できます。

2. DBG> CANCEL SCOPE  
    DBG> SET SCOPE/CURRENT 1

この例では、CANCEL SCOPE コマンドが省略時の有効範囲検索リスト (0,1,2, ..., n) を復元します。次に SET SCOPE/CURRENT コマンドが有効範囲検索リストを 1,2, ..., n に変更し、シンボル検索が有効範囲 1 (すなわち、実行が現在中断しているルーチンの呼び出し元) で開始するようにします。定義済みソース表示 SRC と機械語命令ディスプレイ INST が更新され、実行が中断しているルーチンの呼び出し元のソースと命令を表示するようになります。

3. DBG> SET SCOPE 1  
    DBG> EXAMINE %R5

この例では、SET SCOPE コマンドは、有効範囲 1 (すなわち、実行が中断しているルーチンの呼び出し元) でパス名接頭識別子のないシンボルを検索するようにデバッガに指示します。EXAMINE コマンドは、呼び出し元のルーチンのレジスタ R5 の値を表示します。SET SCOPE コマンドとともに /CURRENT を指定しないと、ソース表示または機械語命令ディスプレイは更新されません。

## 4. DBG&gt; SET SCOPE 0, STACKS\R2, SCREEN

このコマンドは、次の有効範囲検索リストに従ってパス名接頭識別子のないシンボルを検索するようにデバッガに指示します。これにより、デバッガは最初に PC 有効範囲 (0 で示される) で検索します。ここで指定されたシンボルを見つけないと、STACKS モジュールの R2 ルーチンで探します。それでも見つからない場合は、SCREEN モジュール内を検索します。ここでも指定されたシンボルを見つけないと、デバッガはそれ以上は検索しません。

```
5. DBG> SHOW SYMBOL X
data ALPHA\X                ! global X
data ALPHA\BETA\X           ! local X
data X (global)             ! same as ALPHA\X
DBG> SHOW SCOPE
scope: 0 [ = ALPHA\BETA ]
DBG> SYMBOLIZE X
address ALPHA\BETA\%R0:
    ALPHA\BETA\X
DBG> SET SCOPE \
DBG> SYMBOLIZE X
address 00000200:
    ALPHA\X
address 00000200: (global)
    X
DBG>
```

この例では、SHOW SYMBOL コマンドが、シンボル X に対して 2 つの宣言 (グローバル ALPHA\X (2 回定義されている) とローカル ALPHA\BETA\X) があることを示します。現在の有効範囲では、X (ALPHA\BETA\X) のローカル宣言が可視状態にあります。有効範囲がグローバルな有効範囲 (SET SCOPE \) に設定されたあとで、X のグローバル宣言が可視状態になります。

---

## SET SEARCH

SEARCH コマンドの省略時の修飾子 (/ALL または /NEXT , /IDENTIFIER または /STRING) を設定します。

---

### フォーマット

```
SET SEARCH search-default[, . . . ]
```

---

### パラメータ

*search-default*

SEARCH コマンドの省略時の修飾子を指定します。次のキーワード (SEARCH コマンド修飾子と同じです) を指定できます。

ALL	それ以降の SEARCH コマンドが SEARCH/NEXT ではなく SEARCH /ALL として扱われます。
IDENTIFIER	それ以降の SEARCH コマンドが SEARCH/STRING ではなく SEARCH /IDENTIFIER として扱われます。
NEXT	省略時の設定。それ以降の SEARCH コマンドが SEARCH/ALL ではなく SEARCH/NEXT として扱われます。
STRING	省略時の設定。それ以降の SEARCH コマンドが SEARCH/IDENTIFIER ではなく SEARCH/STRING として扱われます。

---

### 説明

SET SEARCH コマンドは、それ以降の SEARCH コマンドの省略時の修飾子を設定します。SET SEARCH に指定するパラメータは、SEARCH コマンドの修飾子と同じ名前です。これらの修飾子は、SEARCH コマンドが(1)その文字列のすべての出現箇所を検索するか (ALL) または次の出現箇所だけを検索するか (NEXT) を決定し、さらに(2)その文字列の任意の出現箇所を表示するか (STRING) または現在の言語で識別子の一部を構成できる文字によってどちら側にもつながっていない文字列の出現箇所だけを表示するか (IDENTIFIER) を決定します。

1 つの SEARCH コマンドの実行中に、別の SEARCH 修飾子を指定すれば、現在の SEARCH の省略時の修飾子上書きできます。現在の SEARCH の省略時の修飾子を表示するには、SHOW SEARCH コマンドを使用します。

#### 関連コマンド

SEARCH  
(SET,SHOW) LANGUAGE

## SHOW SEARCH

---

例

```
DBG> SHOW SEARCH
search settings: search for next occurrence, as a string

DBG> SET SEARCH IDENTIFIER
DBG> SHOW SEARCH
search settings: search for next occurrence, as an identifier

DBG> SET SEARCH ALL
DBG> SHOW SEARCH
search settings: search for all occurrences, as an identifier
DBG>
```

この例では、SET SEARCH IDENTIFIER コマンドが、指定された範囲内でその文字列が出現するところを検索するが、現在の言語の識別子の一部になれる文字によってどちらの側にもつながっていない場合だけ、その文字列を表示するようにデバッガに指示します。

SET SEARCH ALL コマンドは、指定された範囲内でその文字列が出現するところをすべて検索し表示します。

---

## SET SOURCE

ソース・ファイルのディレクトリの検索リストと検索方法のどちらか一方，あるいは両方を指定します。

---

### フォーマット

```
SET SOURCE  directory-spec[, . . . ]
```

---

### パラメータ

*directory-spec*

ソース・ファイルを検索する場合に，省略時の設定としてデバグが使用する OpenVMS ファイル指定部分 (通常は装置 / ディレクトリ) を指定します。ユーザが指定しない全ファイル指定の任意の部分については，モジュールのシンボル・レコードに格納されているファイル指定 (すなわち，ソース・ファイルがコンパイル時に持っていたファイル指定) が使用されます。

1 つの SET SOURCE コマンドでディレクトリを 2 つ以上指定すると，ソース・ディレクトリ検索リストが作成されます (プロセス・レベルで定義した検索リスト論理名を指定することもできます)。この場合，デバグは指定された最初のディレクトリから順に検索し，ソース・ファイルを検索するかディレクトリ・リストが終わるまで検索を続けます。

---

### 修飾子

/DISPLAY

デバグがソース・コードを使用するときに表示するディレクトリ検索リストを指定します。省略時の表示検索ディレクトリはコンパイル・ディレクトリです。

/EDIT

デバグが EDIT コマンドの実行中に使用するディレクトリ検索リストを指定します。省略時の編集検索ディレクトリはコンパイル・ディレクトリです。

/EXACT (省略時の設定)

使用するディレクトリ検索方法を指定します。この修飾子を使用した場合，デバグはデバグ・シンボル・テーブルにあるバージョンと正確に一致するバージョンのソース・ファイルを検索します。



**/LATEST**

使用するディレクトリ検索方法を指定します。この修飾子を使用した場合、デバッガはソース・ファイルの最新バージョン、つまりディレクトリ内の一番大きい番号のバージョンを検索します。

**/MODULE=module-name**

指定されたモジュールだけに使用するディレクトリ検索リストを指定します。SET SOURCE/MODULE コマンドには、上に挙げた修飾子を追加することができます。

**/ORIGINAL**

STDL プログラムにだけ適用される。別売レイヤード・プロダクトのコリレーション・ファシリティ (Correlation Facility) をインストールする必要があり、保持デバッガ (Kept Debugger) を起動する必要があります。デバッガが、STDL のコンパイル時に生成される中間ファイルではなく、元の STDL ソース・ファイルを表示するように指定します。

---

**説明**

省略時の設定では、デバッガはソース・ファイルがコンパイル時に入っていたディレクトリ内にあると予想します。ソース・ファイルがコンパイルのあとに別のディレクトリに移動していた場合は、SET SOURCE コマンドを使用して、ファイルを見つけるためのディレクトリ検索リストとディレクトリ検索方法を指定します。

**ディレクトリ検索リストの指定**

ODS-2 OpenVMS ファイルの完全指定は、次の形式です。

```
node::device:[directory]file-name.file-type;version-number
```

この形式は、OpenVMS オペレーティング・システムに同梱の DECnet Phase IV で使用される DECnet ノード名機能を反映しています。詳細は、『DECnet for OpenVMS Networking Manual』を参照してください。

OpenVMS バージョン 6.1 と DECnet-Plus for OpenVMS を稼動している OpenVMS システムでは、ファイルの完全指定に完全名という拡張ノード指定を含めることができます。完全名は、DECdns 命名サービスに格納することができる階層構造の DECnet-Plus for OpenVMS ノード名です。完全名は次の形式で最高 255 バイトまで可能です。

```
namespace:.directory ... .directory.node-name
```

この構文では、namespaceにグローバル命名サービスを、directory ... .directoryに命名サービス内の階層ディレクトリパスを、そしてnode-nameに DECnet ノードを定義する特定のオブジェクトを指定します。

名前の体系を設定するための完全名と提案事項については、『OpenVMS システム管理者マニュアル』を参照してください。DECnet-Plus for OpenVMS については、『DECnet-Plus for OpenVMS Introduction and User's Guide』を参照してください。

ソース・ファイルのファイル完全指定が 255 文字を超えると、デバグはファイルを検索できません。この問題を解決するには、まず論理名 "X" (DCL レベルで) を長いファイル指定に合わせて拡大するよう定義してから SET SOURCE X コマンドを使用します。

/DISPLAY 修飾子または/EDIT 修飾子のどちらもない SET SOURCE コマンドはディスプレイおよび編集検索ディレクトリの両方を変更します。

/DEBUG 修飾子を使用してプログラムをコンパイルする場合、ルート・ディレクトリ論理名を使用してソース・ファイルの記憶位置を指定するときは、それが隠しルート・ディレクトリ論理名でなければなりません。隠しルート・ディレクトリ論理名でない場合ソース・ファイルを他のディレクトリに移動すると、デバグの SET SOURCE コマンドを使用してソース・ファイルの新しい記憶位置を指定することができません。

隠しルート・ディレクトリ論理名を作成するには、DCL の DEFINE コマンドに /TRANSLATION\_ATTR=CONCEALED 修飾子を指定します。

#### ディレクトリ検索方法の指定

SET SOURCE コマンドを実行する場合、2 つの修飾子/LATEST と/EXACT のいずれかが必ず有効でなければなりません。これらの修飾子は、デバグの検索方法に影響を与えます。/LATEST 修飾子は、最後に作成されたバージョン (ディレクトリ内の一番大きい番号のバージョン) を検索するようデバグに指示します。/EXACT 修飾子は、最後にコンパイルされたバージョン (コンパイル時に作成されたデバグ・シンボル・テーブルに記録されているバージョン) を検索するようデバグに指示します。たとえば、SET SOURCE/LATEST コマンドは SORT.FOR;3 を検索し、SET SOURCE/EXACT コマンドは SORT.FOR;1 を検索するという具合です。

デバグはディレクトリ検索リストを使用してバージョンを見つけると、作成日時、更新日時、ファイル・サイズ、レコード形式、およびファイル構成がコンパイル時の元のソース・ファイルと同じかどうかを調べます。これらがすべて一致した場合、デバグは元のソース・ファイルが新しいディレクトリにそのままのバージョンで存在していると結論づけます。

デバグはディレクトリ検索リストを使用してバージョンを見つけられないと、ソース・コードを初めて表示するときに、更新日時が最も近いファイル (そのようなファイルがディレクトリ内に存在している場合) を識別し、NOTORIGSRC メッセージ ("original version of source file not found") を発行します。

#### /EDIT 修飾子の指定

ソース・コードの表示に使用するファイルが EDIT コマンドを使用して編集しようとしているファイルと異なる場合は、/EDIT 修飾子を指定しなければなりません。これは、Ada プログラムの場合に当てはまります。Ada プログラムの場合、(SET, SHOW, CANCEL) SOURCE コマンドは、ソース・ディスプレイに使用するファイル (Ada プログラム・ライブラリ内の"コピーされた"ソース・ファイル) の検索に影響を与えます。(SET, SHOW, CANCEL) SOURCE/EDIT コマンドは、EDIT コマンドを使用して編集するソース・ファイルの検索に影響を与えます。/EDIT とともに/MODULE を使用すると、/EDIT の効果を/MODULE で修飾することができます。

Ada プログラムについての詳しい説明は、ヘルプ・トピック Language\_Support Ada を参照してください。

#### /ORIGINAL 修飾子の指定

システムに別売レイヤード・プロダクトのコリレーション・ファシリティ (Correlation Facility) をインストールしなければ、SET SOURCE コマンドに /ORIGINAL を修飾子を使用することはできません。デバッグ前にコリレーション・ライブラリを作成する方法については、コリレーション・ファシリティのドキュメントを参照してください。

次に、保持デバッガ (Kept Debugger) を起動して SET SOURCE/ORIGINAL コマンドを次のように実行します。

```
$  DEBUG/KEEP
DBG> SET SOURCE/ORIGINAL
DBG> RUN filename.EXE
```

以上のコマンドを実行すると、他のサポート言語のプログラムと同じように STDL ソース・コードをデバッグできるようになります。

#### 関連コマンド

(SHOW,CANCEL) SOURCE

---

例

1. 

```
DBG> SHOW SOURCE
no directory search list in effect
DBG> SET SOURCE [PROJA],[PROJB],[PETER.PROJC]
DBG> SHOW SOURCE
source directory list for all modules,
match the latest source file version:
[PROJA]
[PROJB]
[PETER.PROJC]
```

この例ではSET SOURCE コマンドによって、デバッガがディレクトリ[PROJA]、[PROJB]、[PETER.PROJC]をこの順に検索してソース・ファイルの最新バージョンを検索するようにしています。

2. 

```
DBG> SET SOURCE/MODULE=CTEST/EXACT [],SYSTEM::DEVICE:[PROJD]
DBG> SHOW SOURCE
source directory search list for CTEST,
match the exact source file version:
[]
SYSTEM::DEVICE:[PROJD]
source directory list for all other modules,
match the latest source file version:
[PROJA]
[PROJB]
[PETER.PROJC]
```

前の例の続きのこの例では、SET SOURCE/MODULE=CTEST コマンドによって、デバッガが現在の省略時のディレクトリ ([ ] )、SYSTEM::DEVICE:[PROJD]の順にソース・ファイルを検索し、CTEST モジュールで使用できるようにしています。/EXACT 修飾子は、デバッグ・シンボル・テーブルにあるバージョンと正確に一致するバージョンの CTEST ソース・ファイルを検索するように指定しています。

3. 

```
DBG> SET SOURCE /EXACT
DBG> SHOW SOURCE
no directory search list in effect,
match the exact source file
DBG> SET SOURCE [JONES]
DBG> SHOW SOURCE
source directory list for all modules,
match the exact source file version:
[JONES]
DBG> CANCEL SOURCE /EXACT
DBG> SHOW SOURCE
source directory list for all modules,
match the latest source file version:
[JONES]
```

この例では、最初の SET SOURCE/EXACT コマンドで設定した検索方法 (一致バージョンの検索) は、SET SOURCE [JONES] コマンドでもそのまま有効です。

CANCEL SOURCE/EXACT コマンドは、SET SOURCE/EXACT コマンドを取り消し、さらに SET SOURCE [JONES]に影響を与えています。

---

## SET STEP

STEP コマンドの省略時の修飾子 (/LINE , /INTO など) を設定します。

---

### フォーマット

SET STEP *step-default*[, ... ]

---

### パラメータ

#### step-default

STEP コマンドの省略時の設定を指定します。次のキーワード (STEP コマンドの修飾子と同じ) を指定できます。

BRANCH	それ以降の STEP コマンドが STEP/BRANCH (次の分岐命令までステップ実行する) として扱われます。
CALL	それ以降の STEP コマンドが STEP/CALL (次の呼び出し命令までステップ実行する) として扱われます。
EXCEPTION	それ以降の STEP コマンドが STEP/EXCEPTION (次の例外までステップ実行する) として扱われます。
INSTRUCTION	それ以降の STEP コマンドが STEP/INSTRUCTION (次の命令までステップ実行する) として扱われます。  VAX プロセッサでは、1 つまたは複数の命令( <i>opcode</i> [, ... ])も指定できます。デバッグは指定されたりスト中で次の命令までステップ実行します。  VAX プロセッサでは、ベクタ命令を指定する場合、命令修飾子 (/UNALIGNED_DATA, /MODIFY, /0, または/1) と命令ニーモニックをいっしょには指定できません。
INTO	それ以降の STEP コマンドが STEP/OVER(呼び出されたルーチンを 1 ステップ実行する) ではなく STEP/INTO (呼び出されたルーチン内の命令をステップ実行する) として扱われます。INTO を指定した場合、パラメータ [NO]JSB, [NO]SHARE, [NO]SYSTEM を使用するか、または STEP/[NO]JSB, STEP/[NO]SHARE, STEP/[NO]SYSTEM のコマンドと修飾子の組み合わせ (当 STEP コマンドにだけ影響を与えます) を使用して、ステップ実行したいルーチンの型を修飾できます。
JSB	(VAX のみ) INTO が指定されている場合、それ以降の STEP コマンドは STEP/INTO/JSB (JSB 命令によって呼び出されたルーチンと CALL 命令によって呼び出されたルーチン内の命令をステップ実行する) として扱われます。DIBOL を除くすべての言語では、これが省略時の設定です。
NOJSB	(VAX のみ) INTO が指定されている場合、それ以降の STEP コマンドは STEP/INTO/NOJSB (JSB 命令によって呼び出されたルーチンを 1 ステップとして実行するが、CALL 命令によって呼び出されたルーチンは、その中の命令をステップ実行する) として扱われます。これは、DIBOL における省略時の設定です。

LINE	省略時の設定。それ以降の STEP コマンドは、STEP/LINE (次の行までステップ実行する) として扱われます。
OVER	省略時の設定。それ以降の STEP コマンドは、STEP/INTO (呼び出されたルーチン内の命令をステップ実行する) ではなく STEP/OVER (呼び出されたルーチンを 1 ステップとして実行する) として扱われます。
RETURN	それ以降の STEP コマンドは、STEP/RETURN (現在実行中のルーチンの戻り命令をステップ実行する。すなわち、制御が呼び出し元のルーチンに戻る前の地点までステップ実行する) として扱われます。
LINE	省略時の設定。それ以降の STEP コマンドは、STEP/LINE (次の行までステップ実行する) として扱われます。
SEMANTIC_EVENT	(Alpha のみ) それ以降の STEP コマンドは、STEP/SEMANTIC_EVENT (次のセマンティック・イベントまでステップ実行する) として扱われます。これによって、最適化されたプログラムのデバッグが簡単になります。(詳細は、『デバッグ説明書』を参照してください。)
SHARE	省略時の設定。INTO が指定されている場合、それ以降の STEP コマンドは STEP/INTO/SHARE (共用可能イメージ内の呼び出されたルーチンとそれ以外の呼び出されたルーチン内の命令をステップ実行する) として扱われます。
NOSHARE	INTO が指定されている場合、それ以降の STEP コマンドが、STEP/INTO/NOSHARE (共用可能イメージ内の呼び出されたルーチンを 1 ステップとして実行するが、それ以外のルーチンでは、その中の命令をステップ実行する) として扱われます。
SILENT	それ以降の STEP コマンドが STEP/SILENT (1 ステップあとに、"stepped to ..."メッセージまたは現在の記憶位置のソース行を表示しない) として扱われます。
NOSILENT	省略時の設定。それ以降の STEP コマンドが STEP/NOSILENT (1 ステップあとに、"stepped to ..."メッセージを表示する) として扱われます。
SOURCE	省略時の設定。それ以降の STEP コマンドが STEP/SOURCE(1 ステップあとに、現在の記憶位置のソース行を表示する) として扱われます。またそれ以降のコマンド SET BREAK, SET TRACE, SET WATCH は、それぞれコマンド SET BREAK/SOURCE, SET TRACE/SOURCE, SET WATCH/SOURCE(ブレークポイント、トレースポイント、ウォッチポイントで現在の記憶位置のソース行を表示する) として扱われます。
NOSOURCE	それ以降の STEP コマンドが STEP/NOSOURCE(1 ストップあとに、現在の記憶位置のソース行を表示しない) として扱われます。また、それ以降のコマンド SET BREAK, SET TRACE, SET WATCH は、それぞれ SET BREAK/NOSOURCE, SET TRACE/NOSOURCE, SET WATCH/NOSOURCE(ブレークポイント、トレースポイント、ウォッチポイントで現在の記憶位置を表示しない) として扱われます。
SYSTEM	省略時の設定。INTO が指定されていると、それ以降の STEP コマンドが STEP/INTO/SYSTEM (システム空間 (P1 空間) 内の呼び出されたルーチンとそれ以外の呼び出されたルーチン内の命令をステップ実行する) として扱われます。
NOSYSTEM	INTO が指定されている場合、それ以降の STEP コマンドが STEP/INTO/NOSYSTEM(システム空間内の呼び出されたルーチンを 1 ステップとして実行するが、他のルーチンの場合はそのルーチン内の命令をステップ実行する) として扱われます。

---

説明

SET STEP コマンドは、それ以降の STEP コマンドの省略時の修飾子を設定します。SET STEP コマンドで指定するパラメータは、STEP コマンドの修飾子と同じ名前です。次のパラメータは、STEP コマンドが 1 ステップあとでどこで実行を中断するのかを決定します。

BRANCH  
CALL  
EXCEPTION  
INSTRUCTION  
INSTRUCTION=(*opcode*[, . . . ]) (VAX のみ)  
LINE  
RETURN  
SEMANTIC\_EVENT (Alpha のみ)

次のパラメータは、STEP コマンドを実行したときにどの出力を表示するかを制御します。

[NO]SILENT  
[NO]SOURCE

次のパラメータは、ルーチン呼び出したときの動作を制御します。

INTO  
[NO]JSB (VAX のみ)  
OVER  
[NO]SHARE  
[NO]SYSTEM

STEP コマンドで省略時の設定以外の修飾子を指定すると、そのコマンドの実行中だけ、現在の STEP の省略時の修飾子を上書きできます。SHOW STEP コマンドを使用すると現在の STEP の省略時の修飾子を表示できます。

PF1-PF3 を押して画面モードにすると、SET STEP NOSOURCE コマンドと SET MODE SCREEN コマンドが指定されます。したがって、STEP コマンドの実行の結果またはブレークポイント、トレースポイント、ウォッチポイントが検出された結果作成された出力表示と DO 表示内ではソース・コードが表示されず、ソース表示による無駄を省くことができます。

OpenVMS VAX システムでは、STEP/OVER コマンドはステップ実行ではなく Fortran 実行時ライブラリ・ルーチンを実行することがあります。詳細は、『デバッグ説明書』を参照してください。

## 関連コマンド

SHOW STEP



## STEP

---

例

1. DBG> SET STEP INSTRUCTION,NOSOURCE

このコマンドを使用すると、デバッガはSTEP コマンドを入力したときに次の命令まで実行します。STEP コマンドごとにソース・コード行を表示することはありません。

2. DBG> SET STEP LINE,INTO,NOSYSTEM,NOSHARE

このコマンドを使用すると、デバッガはSTEP コマンドを入力したときにプログラムの次の行まで実行してから、ユーザ空間にある呼び出されたルーチン内の命令だけをステップ実行します。システム空間内のルーチンと共用可能イメージ内のルーチンは1 ステップとして実行されます。

---

## SET TASK|THREAD

タスキング・プログラム (マルチスレッド・プログラムとも呼ばれます) の 1 つまたは複数のタスクの属性を変更します。

---

### フォーマット

```
SET TASK  [task-spec], ... ]]
```

---

### パラメータ

*task-spec*

タスク値を指定します。次のいずれかの形式を指定します。

- イベント機能が THREADS の場合
  - プログラムで宣言されているタスク (スレッド) ID 番号, または結果がタスク ID 番号となる言語式。
  - 2 などのタスク ID 番号 (SHOW TASK コマンドで表示される)
- イベント機能が ADA の場合
  - プログラムで宣言されているタスク (スレッド) 名, または結果がタスク値となる言語式。パス名も使用できます。
  - %TASK 2 などのタスク ID (SHOW TASK コマンドで表示される)
- 次のタスク組み込みシンボルのいずれか

%ACTIVE_TASK	GO, STEP, CALL, または EXIT コマンドの実行時に実行されるタスク。
%CALLER_TASK	Ada プログラムにのみ適用される。accept 文の実行時に, その accept 文に対応するエントリを呼び出したタスク。
%NEXT_TASK	デバッガのタスク・リスト中で可視タスクの次のタスク。タスクの順番は自由に決められますが, その順番は 1 回のプログラム実行の中で不変です。
%PREVIOUS_TASK	デバッガのタスク・リスト中で可視タスクの前のタスク。
%VISIBLE_TASK	シンボル, レジスタ値, ルーチン呼び出し, ブレークポイントなどの検索時に現在のコンテキストになっている呼び出しスタックとレジスタ・セットを持つタスク。

ワイルドカード文字のアスタリスク(\*)は使用できません。代わりに/ALL 修飾子を使用してください。/ALL または/TIME\_SLICE を指定する場合, タスクは指定できません。タスクを指定しないか, /ABORT, /[NO]HOLD, /PRIORITY, または/RESTORE とともに/ALL を指定しないと, 可視タスクが選択されます。

---

## 修飾子

### /ABORT

指定されたタスクに終了のマークをつけます。指定されたタスクが実行を再開したあとに次に認められる時点で終了します。

Compaq Ada タスクの場合、指定されたタスクに対して Ada 強制終了文を実行することと同じであり、これらのタスクは異常として示されます。依存タスクにも終了のマークがつけられます。

POSIX Threads スレッドの場合、次のコマンドを使用します。

```
PTHREAD tset -c thread-number
```

### /ACTIVE

指定されたタスクをアクティブ・タスクにします。これは、コマンド STEP, GO, CALL または EXIT コマンドを実行したときに実行されるタスクです。この修飾子を使用すると、タスクは新しいアクティブ・タスクに切り替わり、そのタスクが可視タスクになります。指定されたタスクは、RUNNING 状態か READY 状態になければなりません。/ACTIVE を使用する場合、タスクを 1 つ指定しなければなりません。

SET TASK/ACTIVE コマンドは、VAX 上の Compaq Ada でのみサポートされています。POSIX Threads プログラムまたは Alpha プログラム上の Compaq Ada では、次のコマンドのいずれかを使用します。

- クエリー型のアクションには、SET TASK/VISIBLE コマンドを使用
- 実行の制御を得るには、ブレイクポイントを効果的な位置に配置する
- PTHREAD tset -a thread-number コマンドを使用する

### /ALL

SET TASK コマンドをすべてのタスクに適用します。

### /HOLD

#### /NOHOLD (省略時の設定)

イベント機能が THREADS の場合は、PTHREAD tset -h thread-number コマンドまたは PTHREAD tset -n thread-number コマンドを使用します。

指定されたタスクを保留するかどうかを制御します。/HOLD 修飾子は指定されたタスクを保留します。

タスクを保留すると、そのタスクを RUNNING 状態にできなくなります。保留状態にあるタスクは、他の状態に移行できます。たとえば、SUSPENDED 状態から READY 状態に変わることができます。

すでに RUNNING 状態にあるタスク (アクティブ・タスク) は、RUNNING 状態にあるかぎり実行を続けることができます。実行は保留状態にあっても続けられます。何らかの理由 (タイム・スライスが使用可能になっている場合、タイム・スライス時

間の終了など)でRUNNING状態でなくなると、保留条件が解消されないかぎり、RUNNING状態に戻ることはできません。

SET TASK/ACTIVE コマンドを使用すれば、タスクが保留されていても、保留条件を無効にし、タスクをRUNNING状態にできます。

/NOHOLD 修飾子は、指定されたタスクの保留を解除します。

/PRIORITY=*n*

イベント機能がTHREADSの場合は、PTHREAD tset -s thread-numberコマンドを使用します。

指定されたタスクの優先順位を*n*に設定します。ここで*n*は0～15までの10進整数です。これにより、実行中(たとえば、Ada ランデブまたはPOSIX Threads同期化の実行中)にあとで優先順位を変更できなくなるわけではありません。この修飾子は、タスクのスケジューリング方法には影響を与えません。

/RESTORE

(Compaq Ada on VAXのみ) 指定されたタスクの優先順位を、作成されたときの優先順位に復元します。タスクのスケジューリング方法には影響を与えません。

/TIME\_SLICE=*t*

(Compaq Ada on VAXのみ) タイム・スライス期間を値*t*に設定します。ここで、*t*は秒数を表す10進整数または実数値です。ここで設定された値は、プログラム内で指定されるタイム・スライス値(指定されている場合)より優先されます。タイム・スライスを禁止するには、/TIME\_SLICE=0.0を使用します。/TIME\_SLICEはイベント機能がADAのときのみ有効になります。

/VISIBLE

指定されたタスクを可視タスクにします。可視タスクは、シンボル、レジスタ値、ルーチン呼び出し、ブレークポイントなどの検索時に、呼び出しスタックとレジスタ・セットが現在のコンテキストになるタスクです。EXAMINEなどのコマンドは、可視タスクに対して実行されます。/VISIBLE 修飾子はアクティブ・タスクには影響を与えません。/VISIBLEを使用する場合は、タスクを1つ指定しなければなりません。

---

## 説明

---

### 注意

SET TASKとSET THREADは同じ意味のコマンドです。これらのコマンドの動作は同じです。

---

SET TASK コマンドは、可視タスクとアクティブ・タスクの設定、これらのタスクの実行の制御、タスク状態の変更を直接または間接的に行えるようにします。

タスクの現在の状態を調べるには、SHOW TASK コマンドを使用します。タスクは、RUNNING、READY、SUSPENDED、TERMINATED のいずれかの状態にあります。

#### 関連コマンド

DEPOSIT/TASK  
 EXAMINE/TASK  
 SET BREAK/EVENT  
 SET TRACE/EVENT  
 (SET, SHOW) EVENT\_FACILITY  
 SHOW TASK | THREAD

---

#### 例

1. DBG> SET TASK/ACTIVE %TASK 3  
 (イベント機能= ADA) このコマンドは、タスク 3 (タスク ID = 3) をアクティブ・タスクにします。
2. DBG> PTHREAD tset -a 3  
 (イベント機能= THREADS) このコマンドは、タスク 3 (タスク ID = 3) をアクティブ・タスクにします。
3. DBG> SET TASK %NEXT\_TASK  
 このコマンドは、デバッガのタスク・リスト中で次のタスクを可視タスクにします (/VISIBLE 修飾子は、SET TASK コマンドの省略時の修飾子です)。
4. DBG> SET TASK/HOLD/ALL  
 DBG> SET TASK/ACTIVE %TASK 1  
 DBG> GO  
     ...  
 DBG> SET TASK/ACTIVE %TASK 3  
 DBG> STEP  
     ...

この例では、SET TASK/HOLD/ALL コマンドでアクティブ・タスク以外のすべてのタスクの状態を凍結します。その後、必要なタスクに対して SET TASK /ACTIVE を使用して (GO コマンドと STEP コマンドとともに)、1 つまたは複数の指定されたタスクの動作を切り離して観察できます。

---

## SET TERMINAL

デバッガが画面やその他の出力を編集するときに使用する端末画面の高さと幅を設定します。

---

### 注意

このコマンドは、デバッガへの HP DECwindows Motif for OpenVMS ユーザ・インタフェースでは使用できません。

---

---

## フォーマット

SET TERMINAL

---

## 修飾子

/PAGE:n

端末画面の高さを  $n$  行に設定することを指定します。18 から 100 まで指定できます。

/WIDTH:n

端末画面の幅を  $n$  欄に設定することを指定します。20 から 255 まで指定できます。

VT100, VT200 または VT300 のシリーズの端末では、 $n$  は通常の場合 80 または 132 です。

/WRAP

デバッガが定義済みのディスプレイ OUT に /WIDTH 修飾子によって指定されたカラムで出力テキストをラップするようにします。/WIDTH を現在のコマンドに指定しない場合は、/WRAP は %WIDTH 設定を省略時の設定にします。

---

## 説明

SET TERMINAL コマンドを使用すると、デバッガが画面出力の編集に使用できる画面部分を定義できます。

このコマンドは、VT100, VT200 または VT300 のシリーズの端末を使用する場合に便利です。この場合、たいいていは画面幅を 80 欄または 132 欄に設定できます。また、ワークステーションではデバッガが使用するターミナル・エミュレータ・ウィンドウのサイズを変更できるので、やはり便利です。

少なくとも 1 つの修飾子を指定しなければなりません。全部の修飾子を指定することも可能です。/PAGE 修飾子と /WIDTH 修飾子には、それぞれ値が必要です。

SET TERMINAL コマンドを入力すると、すべての画面ウィンドウ定義が、新しい画面サイズに合うように自動的に調整されます。たとえば、RH1 は、画面の右上半分を表示するようにサイズを比例して変更します。

同様に、すべての"動的"表示ウィンドウは、相対的な比率を維持するように自動的に調整されます。すべての標準ウィンドウは動的ですが、DISPLAY/NODYNAMIC コマンドで参照した場合は例外です。その場合には、あとで SET TERMINAL コマンドを実行した場合も、表示ウィンドウは現在のサイズのまま変化しません。しかし、DISPLAY コマンドを使用すると、表示ウィンドウを再構成できます (また、BLUE-MINUS などのキーパッド・キーの組み合わせを使用して、定義済みの DISPLAY コマンドを入力することも可能です)。

#### 関連コマンド

DISPLAY/[NO]DYNAMIC  
EXPAND  
(SET,SHOW,CANCEL) WINDOW  
SHOW TERMINAL

---

#### 例

```
DBG> SET TERMINAL/WIDTH:132
```

このコマンドは、端末画面幅を 132 欄に設定することを指定します。

---

## SET TRACE

特定のクラスの命令または指定されたイベントの発生時に、アドレス式で示された記憶位置にトレースポイントを設定します。

---

### フォーマット

```
SET TRACE  [address-expression [, . . . ]]  
           [WHEN(conditional-expression)]  
           [DO(command [, . . . ])]
```

---

### パラメータ

*address-expression*

トレースポイントを設定するアドレス式 (プログラム記憶位置) を指定します。高級言語の場合、これはたいていは行番号、ルーチン名、またはラベルです。値を一意に指定するパス名をいれることもできます。また、アドレス式はメモリ・アドレスまたはレジスタの場合もあります。数字 (オフセット) とシンボルで構成したり、1 つまたは複数の演算子、オペランド、または区切り文字で構成したりできます。アドレス式で使用する演算子についての詳しい説明は、ヘルプ・トピック `Address_Expressions` を参照してください。

ワイルドカード文字のアスタリスク (\*) は使用できません。また次の修飾子を指定する場合は、アドレス式は指定できません。

```
/ACTIVATING  
/BRANCH  
/CALL  
/EXCEPTION  
/INSTRUCTION  
/INSTRUCTION=(opcode [, . . . ]) (VAX のみ)  
/INTO  
/[NO]JSB (VAX のみ)  
/LINE  
/OVER  
/[NO]SHARE  
/[NO]SYSTEM  
/TERMINATING
```

/MODIFY 修飾子と/RETURN 修飾子は、特定の種類のアドレス式に指定できます。



メモリ・アドレス、または値がシンボリック記憶位置でないアドレス式を指定する場合は、示しているメモリ・バイトで命令が実際に始まっているかを (EXAMINE コマンドを使用して) チェックしてください。命令がこのバイトで始まっていないと、そのバイトを持つ命令を実行したときに実行時エラーが発生します。値がシンボリック記憶位置でないアドレス式を指定してトレースポイントを設定しても、デバグは指定された記憶位置が命令の開始位置を示しているかどうかをチェックしません。

VAX プロセッサの場合、CALLS ルーチンと CALLG ルーチンは、エントリ・マスクで始まります。

#### conditional-expression

現在設定されている言語で条件式を指定します。この式は実行がブレークポイントに達したときに評価されます。ブレークポイントが設定されたときではなく、実行がブレークポイントに達すると、デバグは WHEN 句にある式の構文をチェックします。式が真であれば、デバグは、ブレークポイントが発生したことを報告します。ブレーク・アクション (DO 句) がブレークポイントと関連付けられている場合は、同時に発生します。式が偽の場合は報告されません。また、DO 句によって指定されたコマンドは実行されず、プログラムの実行は続行されます。

#### command

ブレーク・アクションが実行されたときに、DO 句の一部として実行されるデバグ・コマンドを指定します。ブレークポイントが設定されたときではなく、DO 句が実行されると、デバグは DO 句にある式の構文をチェックします。

---

## 修飾子

### /ACTIVATING

新しいプロセスがデバグの制御下に置かれると、デバグはトレースします。  
/TERMINATING 修飾子も参照してください。

### /AFTER:n

指定されたトレースポイントが  $n$  回検出されるまで、トレース・アクションを実行しないことを指定します ( $n$  は 10 進整数です)。それ以降は、WHEN 句の条件 (指定された場合) が真、検出されるたびにトレースポイントが発生します。SET TRACE/AFTER:1 コマンドは、SET TRACE と同じです。

### /BRANCH

デバグは、プログラムの実行中に検出されるすべての分岐命令をトレースします。  
/INTO 修飾子と/OVER 修飾子も参照してください。

### /CALL

デバグは、プログラムの実行中に呼び出し命令 (復帰命令を含む) を検出するたびにトレースします。  
/INTO 修飾子と/OVER 修飾子も参照してください。

/EVENT=event-name

指定されたイベントが発生する (現在のイベント機能によってイベントが定義され、そのイベントが検出される) と、デバッガがトレースします。アドレス式に/EVENTを指定すると、そのアドレス式に対して指定されたイベントが発生するたびにデバッガがトレースします。アドレス式に特定のイベント名を指定することはできません。

イベント機能は、Ada ルーチンまたは SCAN ルーチンを呼び出すプログラムまたは POSIX Threads サービスを使用するプログラムで使用できます。現在のイベント機能および関連したイベント名を表示するには、SHOW EVENT\_FACILITY コマンドを使用します。

/EXCEPTION

デバッガはシグナル通知されるすべての例外をトレースします。トレース・アクションは、アプリケーションで宣言した例外ハンドラが起動される前に実行されます。

プログラムが例外を発生すると、SET TRACE/EXCEPTION コマンドの結果として、デバッガは例外を報告しその例外を再度シグナル通知します。その結果、アプリケーションで宣言した例外ハンドラを実行できます。

/INSTRUCTION

/INSTRUCTION[=(opcode[, ... ])] (VAX のみ)

命令コードを指定しないと、デバッガはプログラム実行中に検出されたすべての命令をトレースします。

(VAX のみ) 命令コードを 1 つまたは複数個指定すると、デバッガは、リスト内で指定したすべての命令コードをトレースします。

/INTO 修飾子と/OVER 修飾子も参照してください。

/INTO

(省略時の設定。) 次の修飾子で設定されているトレースポイント (すなわち、アドレス式が明示的に指定されていない場合) にだけ指定できます。

/BRANCH

/CALL

/INSTRUCTION

/INSTRUCTION=(opcode[, ... ]) (VAX のみ)

/LINE

/INTO をこれらの修飾子といっしょに使用すると、デバッガは、呼び出されたルーチン内 (実行が現在中断されているルーチン内だけでなく) の指定された箇所をトレースします。/INTO 修飾子は省略時の設定であり、/OVER の反対です。

/INTO を使用する場合、/[NO]JSB、/[NO]SHARE、/[NO]SYSTEM の修飾子でトレース・アクションをさらに修飾できます。

```
/JSB
/NOJSB
```

(VAX のみ) の場合、/INTO を修飾します。/INTO と次のいずれかの修飾子といっしょに使用します。

```
/BRANCH
/CALL
/INSTRUCTION
/INSTRUCTION=(opcode[, ... ])
/LINE
```

/JSB 修飾子は、DIBOL 以外のすべての言語における省略時の値です。これを指定すると、デバッガは、JSB 命令または CALL 命令によって呼び出されるルーチン内にトレースポイントを設定します。/NOJSB 修飾子 (DIBOL の省略時の設定) は、JSB 命令によって呼び出されるルーチン内でトレースポイントを設定しないことを指定します。DIBOL の場合、アプリケーションで宣言したルーチンは CALL 命令によって呼び出され、DIBOL 実行時ライブラリ・ルーチンが JSB 命令によって呼び出されます。

```
/LINE
```

デバッガは、プログラム実行中にソース行が検出されるたびにその行の先頭をトレースします。/INTO 修飾子と/OVER 修飾子も参照してください。

```
/MODIFY
```

指定されたアドレス式が示す記憶位置に命令が値を書き込んで変更したとき、デバッガはトレースを行います。通常の場合、アドレス式は変数名です。

SET TRACE/MODIFY X コマンドは SET WATCH X DO(GO) と同じです。SET TRACE/MODIFY コマンドは、SET WATCH コマンドと全く同じ制限事項の下で動作します。

アドレス式に絶対アドレスを指定すると、デバッガがアドレスを特定のデータ・オブジェクトに関連づけることができない場合があります。この場合、デバッガは省略時の長さとして 4 バイトを使用します。ただし、この長さは、入力を WORD (SET TYPE WORD, 省略時の長さを 2 バイトに変更します) か BYTE (SET TYPE BYTE, 省略時の長さを 1 バイトに変更します) に設定すれば変更できます。SET TYPE LONGWORD コマンドを指定すると、省略時の長さは 4 バイトに戻ります。

```
/OVER
```

次のいずれかの修飾子で設定されているトレースポイント (すなわち、アドレス式が明示的に指定されていない場合) だけに指定できます。

```
/BRANCH
/CALL
/INSTRUCTION
/INSTRUCTION=(opcode[, ... ]) (VAX のみ)
/LINE
```

/OVER をこれらの修飾子といっしょに使用すると、(呼び出されたルーチンではなく) 現在実行を中断しているルーチン内だけの指定された箇所をデバッガはトレースします。/OVER 修飾子は、/INTO (省略時の設定) の反対です。

/RETURN

指定されたアドレス式 (ルーチン名、行番号など) に関連しているルーチンの復帰命令でデバッガがブレークします。復帰命令でブレークすると、ルーチンがアクティブである間ローカル環境を調べる (たとえば、ローカル変数の値を得るなど) ができます。ローカル環境のビューはアーキテクチャにより異なるので注意してください。

VAX プロセッサの場合、この修飾子は、CALLS 命令または CALLG 命令で呼び出したルーチンにだけ指定できます。JSB ルーチンでは使用できません。Alpha プロセッサの場合、この修飾子はどのルーチンにも指定できます。

address-expression パラメータは、ルーチン内の命令アドレスです。単なるルーチン名の場合もあります。この場合は、ルーチンの開始アドレスを指定します。ただし、ルーチン内の別の場所を指定することもできます。こうすると、特定のコード・パスを実行したあとに行われる復帰だけを調べることができます。

SET TRACE/RETURN コマンドで SET TRACE と同じアドレス式を指定すると、SET TRACE は取り消されます。

/SHARE (省略時の設定)

/NOSHARE

/INTO を修飾します。/INTO と次のいずれかの修飾子といっしょに使用します。

/BRANCH

/CALL

/INSTRUCTION

/INSTRUCTION=(opcode[, ... ]) (VAX のみ)

/LINE

/SHARE 修飾子を使用すると、デバッガは共用可能イメージ・ルーチン内でもその他のルーチンの場合と同様にトレースポイントを設定できます。/NOSHARE 修飾子を指定すると、共用可能イメージ内にはトレースポイントは設定されません。

/SILENT

/NOSILENT (省略時の設定)

"trace ..." メッセージと、現在の記憶位置のソース行をトレースポイントで表示するかどうかを制御します。/NOSILENT 修飾子を指定すると、メッセージが表示されます。/SILENT 修飾子を指定すると、メッセージとソース行は表示されません。/SILENT 修飾子を指定すると、/SOURCE は上書きされます。

/SOURCE

/NOSOURCE (省略時の設定)

現在の記憶位置のソース行をトレースポイントで表示するかどうかを制御します。/SOURCE 修飾子を指定すると、ソース行が表示されます。/NOSOURCE 修飾子を指

定すると、ソース行は表示されません。/SILENT 修飾子を指定すると、/SOURCE は上書きされます。SET STEP [NO]SOURCE コマンドも参照してください。

/SYSTEM (省略時の設定)

/NOSYSTEM

/INTO を修飾します。/INTO と次のいずれかの修飾子といっしょに使用します。

/BRANCH

/CALL

/INSTRUCTION

/INSTRUCTION=(*opcode* [, . . . ]) (VAX のみ)

/LINE

/SYSTEM 修飾子を指定すると、他のルーチンだけでなくシステム・ルーチン (P1 空間) 内でもデバッガはトレースポイントを設定できます。/NOSYSTEM 修飾子を指定すると、システム・ルーチン内ではトレースポイントは設定されません。

/TEMPORARY

トレースポイントを検出したあとでそのトレースポイントを消去します (トレースポイントを一時的に設定するとき 사용합니다)。

/TERMINATING

省略時の設定。プロセスがイメージを終了したときにデバッガがトレースします。単一プロセス・プログラムまたはマルチプロセス・プログラムの最後のイメージが終了すると、デバッガに制御が戻り、そのプロンプトを表示します。/ACTIVATING 修飾子も参照してください。

---

説明

トレースポイントが検出されると、デバッガは次の動作を行います。

1. トレースポイント記憶位置でプログラムの実行を中断する。
2. トレースポイントの設定時に/AFTER を指定した場合、AFTER 回数をチェックする。指定された回数に達していないと実行が再開され、デバッガは残りのステップを実行しない。
3. トレースポイントの設定時に WHEN 句を指定した場合、WHEN 句の式を評価する。式の値が偽であれば実行が再開され、デバッガは残りのステップを実行しない。
4. /SILENT が指定されていない場合、"trace . . ."メッセージを発行して、実行がトレースポイント記憶位置に達したことを報告する。
5. トレースポイントの設定時に/NOSOURCE も/SILENT も指定しないか、または SET STEP NOSOURCE コマンドを入力していない場合、トレースポイントに対応したソース・コード行を表示する。
6. トレースポイントの設定時に DO 句を指定していれば、その DO 句内のコマンドを実行する。
7. 実行を再開する。

プログラムの特定の記憶位置にトレースポイントを設定するには、SET TRACE コマンドでアドレス式を指定します。連続したソース行、命令クラス、またはイベントにトレースポイントを設定するには、SET TRACE コマンドとともに修飾子を指定します。通常はアドレス式か修飾子のどちらかを指定するだけでよく、両方を指定する必要はありません。ただし、/EVENT と/RETURN の場合は両方指定しなければなりません。

/LINE 修飾子を指定すると、各ソース・コード行ごとにトレースポイントが設定されます。

次の修飾子は命令クラスにトレースポイントを設定します。これらの修飾子と/LINE 句をいっしょに使用すると、デバッガはプログラムの実行中に各命令をトレースするので、実行速度が著しく遅くなります。

```
/BRANCH  
/CALL  
/INSTRUCTION  
/INSTRUCTION=(opcode[, . . . ]) (VAX のみ)  
/RETURN  
/SYSEMULATE (Alpha のみ)
```

次の修飾子は、イベント・クラスにトレースポイントを設定します。

```
/ACTIVATING
```

```

/EVENT=event-name
/EXCEPTION
/TERMINATING

```

次の修飾子は、ルーチンを呼び出したときに何が起こるかを決定します。

```

/INTO
/[NO]JSB (VAX のみ)
/OVER
/[NO]SHARE
/[NO]SYSTEM

```

次の修飾子は、トレースポイントに達したときにどんな出力を表示するかを決定します。

```

/[NO]SILENT
/[NO]SOURCE

```

次の修飾子は、トレースポイントのタイミングと期間を決定します。

```

/AFTER:n
/TEMPORARY

```

プログラム記憶位置の内容の変更 (通常は変数の値の変更) をモニタするには、  
/MODIFY 修飾子を使用します。

現在ブレークポイントとして使用されている記憶位置をトレースポイントとして設定すると、ブレークポイントは取り消されます。また、逆も同様です。

トレースポイントには、ユーザが定義するものと定義済みのものがあります。ユーザ定義のトレースポイントとは、ユーザが SET TRACE コマンドで明示的に設定したトレースポイントです。定義済みのトレースポイントは、デバッグするプログラムの種類 (Ada あるいはマルチプロセスなど) によって異なりますが、デバッガの起動時に自動的に設定されます。現在設定されているすべてのトレースポイントを表示するには、SHOW TRACE コマンドを使用します。定義済みのトレースポイントは定義済みのものとして表示されます。

ユーザ定義トレースポイントと定義済みトレースポイントは、それぞれ別々に設定したり取り消したりします。たとえば、1 つの記憶位置またはイベントに、ユーザ定義トレースポイントと定義済みトレースポイントの両方を設定することができます。ユーザ定義トレースポイントを取り消しても、定義済みトレースポイントは影響を受けません。逆も同様です。

関連コマンド

```

(ACTIVATE,DEACTIVATE,SHOW,CANCEL) TRACE
CANCEL ALL
GO

```

SET BREAK  
(SET,SHOW) EVENT\_FACILITY  
SET STEP [NO]SOURCE  
SET WATCH

---

## 例

1. DBG> SET TRACE SUB3  
このコマンドの場合、SUB3 ルーチンが実行されるとルーチンの先頭をトレースします。
2. DBG> SET TRACE/BRANCH/CALL  
このコマンドの場合、プログラム実行中に検出されたすべての BRANCH 命令と CALL 命令をトレースします。
3. DBG> SET TRACE/LINE/INTO/NOSHARE/NOSYSTEM  
このコマンドの場合、各ソース行の先頭をトレースします。この行には、呼び出されたルーチン (/INTO) 内の行は含まれますが、共用可能イメージ・ルーチン (/NOSHARE) またはシステム・ルーチン (/NOSYSTEM) 内の行は含まれません。
4. DBG> SET TRACE/NOSOURCE TEST5\%LINE 14 WHEN (X .NE. 2) DO (EXAMINE Y)  
このコマンドの場合、X が 2 ではないとき TEST5 モジュールの行 14 をトレースします。トレースポイントでは EXAMINE Y コマンドが実行されます。/NOSOURCE 修飾子が指定されているので、トレースポイントではソース・コードは表示されません。WHEN 句内の条件式の構文は言語固有です。
5. DBG> SET TRACE/INSTRUCTION WHEN (X .NE. 0)  
このコマンドの場合、X がゼロでないときにトレースします。実行中に検出されたすべての命令で条件が調べられます。WHEN 句内の条件式の構文は言語固有です。
6. DBG> SET TRACE/SILENT SUB2 DO (SET WATCH K)  
このコマンドの場合、実行中に SUB2 ルーチンの先頭をトレースします。トレースポイントでは、DO 句によって変数 K にウォッチポイントが設定されます。/SILENT 修飾子が指定されているので、トレースポイントを検出したとき "trace . . ." メッセージとソース・コードは表示されません。これは、非静的変数 (スタックまたはレジスタ) にウォッチポイントを設定する便利な方法です。非静的変数は、その定義ルーチン (この例では SUB2) がアクティブのとき (呼び出しスタック上に存在するとき) だけ定義されます。



7. `DBG> SET TRACE/RETURN ROUT4 DO (EXAMINE X)`

このコマンドの場合、デバッガはROUT4ルーチンの復帰命令を(すなわち、呼び出し元のルーチンに実行が戻る直前に)トレースします。トレースポイントでは、DO句がEXAMINE Xコマンドを実行します。これは、非静的変数の定義ルーチンの実行が終了する直前にその変数の値を得るのに便利です。

8. `DBG> SET TRACE/EVENT=TERMINATED`

このコマンドの場合、いずれかのタスクがTERMINATED状態に移行するとその時点をトレースします。

---

## SET TYPE

シンボリック名がない(および、そのために関連したコンパイラ生成型を持たない) プログラム記憶位置に関連した省略時の型を設定します。/OVERRIDE とともに使用すると、すべての記憶位置に関連した省略時の型を設定し、コンパイラ生成型を上書きします。

---

### フォーマット

SET TYPE *type-keyword*

---

### パラメータ

*type-keyword*

設定する省略時の型を指定します。次のいずれかのキーワードを指定できます。

ASCIC	1 バイトのカウント・フィールドに続く、このカウント・フィールドにより長さを指定された ASCII 文字列を省略時の型と設定します。AC と入力することもできます。
ASCID	省略時の型を ASCII 文字列ディスクリプタに設定します。ディスクリプタの CLASS フィールドと DTYPE フィールドはチェックされません。LENGTH フィールドと POINTER フィールドは ASCII 文字の文字長さとアドレスを示します。次に文字列が表示されます。AD と入力することもできます。
ASCII:n	省略時の型を ASCII 文字列 (長さ <i>n</i> バイト) に設定します。この長さは、調べるメモリのバイト数と、表示する ASCII 文字数の両方を示します。 <i>n</i> の値を指定しないと、デバッガは省略時の値 4 バイトを使用します。 <i>n</i> の値は 10 進形式の基数で解釈されます。
ASCIW	2 バイトのカウント・フィールドに続く、このカウント・フィールドにより長さを指定された ASCII 文字列を省略時の型と設定します。このデータ型は PASCAL および PL/I の場合に指定できます。AW と入力することもできます。
ASCIZ	省略時の型を 0 で終了する ASCII 文字列に設定します。最後の 0 のバイトは文字列の終わりを示します。AZ と入力することもできます。
BYTE	省略時の型をバイト整数 (1 バイト長) に設定します。
D_FLOAT	省略時の型を D 浮動小数点数 (8 バイト長) に設定します。
DATE_TIME	省略時の型を日時に設定します。これはクォードワード整数 (8 バイト長) であり、日時の内部表現を含んでいます。値は、 <i>dd-mmm-yyyy hh:mm:ss.cc</i> の形式で表示されます。絶対日時は、次のように指定します。
	[dd-mmm-yyyy[:]] [hh:mm:ss.cc]
EXTENDED_FLOAT	(Alpha および I64 のみ) 省略時の型を IFEE の X 浮動小数点 (16 バイト長) に設定します。

FLOAT	VAX プロセッサの場合、省略時の型を IFEE の F 浮動小数点数 (4 バイト長) に設定します。 Alpha プロセッサの場合は、省略時の型を IFEE の T 浮動小数点数 (8 バイト長) に設定します。
G_FLOAT	省略時の型を G 浮動小数点数 (8 バイト長) に設定します。
H_FLOAT	(VAX のみ) 省略時の型を H 浮動小数点数 (16 バイト長) に設定します。
INSTRUCTION	省略時の型を命令 (可変長、使用する命令オペランドの数とアドレッシング・モードの種類によって異なります) に設定します。
LONG_FLOAT	(Alpha および I64 のみ) 省略時の型を IEEE S 浮動小数点数 (単精度、4 バイト長) に設定します。
LONG_LONG_FLOAT	(Alpha および I64 のみ) 省略時の型を IEEE T 浮動小数点数 (倍精度、8 バイト長) に設定します。
LONGWORD	省略時の型をロングワード整数 (4 バイト長) に設定します。これは、シンボリック名がない (コンパイラ生成型を持たない) プログラム記憶位置の省略時の型です。
OCTAWORD	省略時の型をオクタワード整数 (16 バイト長) に設定します。
PACKED:n	省略時の型をパック 10 進数に設定します。 <i>n</i> の値は 10 進数字です。各桁とも 1 ニブル (4 ビット) を占めます。
QUADWORD	省略時の型をクオードワード整数 (8 バイト長) に設定します。これは、64 ビットのアプリケーションのデバッグには使用しないでください
TYPE=( <i>expression</i> )	省略時の型を <i>expression</i> が示す型 (プログラム内で宣言された変数またはデータ型の名前) に設定します。これを指定すると、アプリケーションで宣言した型を指定できます。
S_FLOAT	(Alpha および I64 のみ) LONG_FLOAT と同じです。
T_FLOAT	(Alpha および I64 のみ) LONG_LONG_FLOAT と同じです。
WORD	省略時の型をワード整数 (2 バイト長) に設定します。
X_FLOAT	(Alpha および I64 のみ) EXTENDED_FLOAT と同じです。

---

## 修飾子

### /OVERRIDE

シンボリック名の有無 (関連したコンパイラ生成型の有無)にかかわらず、指定された型をすべてのプログラム記憶位置に関連づけます。

---

## 説明

コマンド EXAMINE、DEPOSIT、または EVALUATE を使用すると、アドレス式の省略時の型によって、デバッガがプログラムの値を解釈し表示する方法が異なります。

デバッガは、シンボリック・アドレス式 (プログラム内で宣言したシンボリック名) に対応したコンパイラ生成型を認識し、これらの記憶位置の内容を解釈して表示します。シンボリック名を持たない (したがって関連したコンパイラ生成型を持たない) プ

ログラム記憶位置の場合，省略時の型はどの言語でもロングワード整数で，32 ビトのアプリケーションのデバッグにも使用できます。

Alpha システムでは，64 ビット・アドレス空間を使用するアプリケーションをデバッグするには，SET TYPE QUADWORD コマンドを使用してください。

SET TYPE コマンドを使用すると，シンボリック名を持たない記憶位置の省略時の型を変更できます。また，SET TYPE/OVERRIDE コマンドを使用すると，シンボリック名の有無にかかわらずすべてのプログラム記憶位置の省略時の型を設定できます。

EXAMINE コマンドと DEPOSIT コマンドには，任意のプログラム記憶位置の型を 1 つのコマンドの実行中に上書きできる型修飾子 (/ASCII , /BYTE , /G\_FLOAT など) があります。

#### 関連コマンド

CANCEL TYPE/OVERRIDE  
DEPOSIT  
EXAMINE  
(SET,SHOW,CANCEL) RADIX  
(SET,SHOW,CANCEL) MODE  
SHOW TYPE

---

#### 例

1. DBG> SET TYPE ASCII:8

このコマンドは，未定義のプログラム記憶位置の省略時の型として 8 バイトの ASCII 文字列を設定します。

2. DBG> SET TYPE/OVERRIDE LONGWORD

このコマンドは，未定義のプログラム記憶位置とコンパイラ生成型を持つプログラム記憶位置の両方の省略時の型としてロングワード整数を設定します。

3. DBG> SET TYPE D\_FLOAT

このコマンドは，未定義のプログラム記憶位置の省略時の型として D 浮動小数点数を指定します。

4. DBG> SET TYPE TYPE=(S\_ARRAY)

このコマンドは，未定義のプログラム記憶位置の省略時の型として S\_ARRAY 変数を設定します。

---

# SET WATCH

アドレス式で示された記憶位置にウォッチポイントを設定します。

---

## フォーマット

```
SET WATCH address-expression[, ... ]
           [WHEN(conditional-expression)]
           [DO(command [, ... ])]
```

---

## パラメータ

### *address-expression*

ウォッチポイントを設定するアドレス式 (プログラム記憶位置) を指定します。高級言語の場合、これはたいていはプログラム変数です。変数を一意に指定するパス名をいれることができます。また、アドレス式はメモリ・アドレスまたはレジスタの場合もあります。数字 (オフセット) とシンボルで構成したり、1 つまたは複数の演算子、オペランド、または区切り文字で構成したりできます。アドレス式で使用する演算子についての詳しい説明は、`HELP Address_Expressions`をタイプしてください。

ワイルドカード文字のアスタリスク (\*) は使用できません。

### *conditional-expression*

現在設定されている言語で条件式を指定します。この式は実行がウォッチポイントに達したときに評価されます。デバッガは、実行がウォッチポイントに達すると、`WHEN` 句にある式の構文をチェックします。式が真であれば、デバッガはウォッチポイントが発生したことを報告します。ブレーク・アクション (`DO` 句) がブレークポイントと関連付けられている場合は、同時に発生します。式が偽の場合は報告されません。また、`DO` 句 (指定されている場合) によって指定されたコマンドは実行されず、プログラム実行が続行されます。

### *command*

ウォッチ・アクションが実行されたときに、実行するデバッガ・コマンドを `DO` 句の一部として指定します。デバッガは、ウォッチポイントが設定されたときではなく、`DO` 句を実行したときに、`DO` 句にある式の構文をチェックします。

---

## 修飾子

### `/AFTER:n`

指定されたウォッチポイントが  $n$  回 ( $n$  は 10 進整数) 検出されるまで、ウォッチ・アクションを実行しないことを指定します。それ以降は、`WHEN` 句の条件 (指定

された場合) が真ならば、検出されるたびにウォッチポイントが発生します。SET WATCH/AFTER:1 コマンドは、SET WATCH と同じです。

#### /INTO

定義ルーチン内だけでなく定義ルーチンから呼び出されたルーチン内 (およびそのようにネストしたそれ以外の呼び出しから呼び出されたルーチン内) の命令もトレースすることにより、非静的変数をデバッガがモニタするよう指定します。SET WATCH/INTO コマンドを使用すると、呼び出されたルーチン内の非静的変数を SET WATCH/OVER を使用したときよりも正確にモニタできます。ただし、呼び出されたルーチンにおける実行速度は SET WATCH/OVER の方が速くなります。

#### /OVER

定義ルーチンが呼び出すルーチン内でなく定義ルーチン内だけで命令のトレースを行うことによって、非静的変数のモニタをデバッガが行うよう指定します。その結果、デバッガは呼び出されたルーチンを通常でモニタし、定義ルーチンに実行が戻ったときだけ命令のトレースを再開します。SET WATCH/OVER コマンドは SET WATCH/INTO より実行速度が速くなります。しかし、呼び出されたルーチンがウォッチされる変数を変更すると定義ルーチンに戻ったときだけ実行が割り込まれます。非静的変数にウォッチポイントを設定する場合、SET WATCH/OVER が省略時の設定です。

#### /SILENT

##### /NOSILENT (省略時の設定)

"watch . . ." メッセージと、現在の記憶位置のソース行をウォッチポイントで表示するかどうかを制御します。/NOSILENT 修飾子を指定すると、メッセージが表示されます。/SILENT 修飾子を指定すると、メッセージとソース行は表示されません。/SILENT 修飾子を指定すると、/SOURCE は上書きされます。

#### /SOURCE (省略時の設定)

##### /NOSOURCE

現在の記憶位置のソース行をウォッチポイントで表示するかどうかを制御します。/SOURCE 修飾子を指定すると、ソース行が表示されます。/NOSOURCE 修飾子を指定すると、ソース行は表示されません。/SILENT 修飾子を指定すると、/SOURCE は上書きされます。SET STEP [NO]SOURCE コマンドも参照してください。

#### /STATIC

##### /NOSTATIC

指定された変数 (ウォッチポイント記憶位置) が静的か非静的かについてのデバッガの省略時の判定を上書きすることができます。/STATIC 修飾子を指定すると、デバッガは変数が P1 空間に割り当てられていても変数を静的変数として扱います。この結果、デバッガは各命令をトレースする代わりに、高速の書き込み保護モードを使用して記憶位置をモニタできます。/NOSTATIC 修飾子を指定すると、デバッガは変数が P0 空間に割り当てられていても変数を非静的変数として扱わなければなりません。このため、デバッガは各命令をトレースして記憶位置をモニタします。したがって、これらの修飾子を使用する場合には、注意が必要です。

**/TEMPORARY**

ウォッチポイントを検出したあとでそのウォッチポイントを消去します (ウォッチポイントを一時的に設定するとき 사용합니다)。

---

**説明**

命令でウォッチポイント記憶位置を変更すると、デバッガは次の処理を実行します。

1. その命令が実行を完了したあと、プログラム実行を中断する。
2. ウォッチポイントの設定時に/AFTER を指定した場合、AFTER 回数をチェックする。指定された回数に達していないと実行が再開され、デバッガは残りのステップを実行しない。
3. ウォッチポイントの設定時に WHEN 句を指定した場合、WHEN 句の式を評価する。式の値が偽であれば実行が継続され、デバッガは残りのステップを実行しない。
4. /SILENT を指定していなければ、実行がウォッチポイント記憶位置に達したことを報告する ("watchof ... ")。
5. ウォッチポイント記憶位置での古い (変更前の) 値を報告する。
6. ウォッチポイント記憶位置での新しい (変更後の) 値を報告する。
7. ウォッチポイント設定時に/NOSOURCE も/SILENT も指定しないか、または SET STEP NOSOURCE コマンドを入力していない場合、実行を中断したソース・コード行を表示する。
8. ウォッチポイント設定時に DO 句を指定していれば、その DO 句内のコマンドを実行する。DO 句に GO コマンドが含まれていれば実行を継続し、デバッガは次のステップを実行しない。
9. プロンプトを表示する。

高級言語の場合、SET WATCH コマンドで指定するアドレス式はたいてい変数です。コンパイラ生成型に対応した絶対メモリ・アドレスを指定すると、デバッガはそのアドレスをシンボル化し、その型に対応したバイト長を使用して、ウォッチポイント記憶位置のバイト長を決定します。デバッガがコンパイラ生成型と関連づけることができない絶対メモリ・アドレスが指定されると、デバッガはアドレス式が示すバイトで始まる 4 バイト (省略時の設定) のメモリをモニタします。ただし、この長さは、入力を WORD (SET TYPE WORD, 省略時の長さを 2 バイトに変更します) か BYTE (SET TYPE BYTE, 省略時の長さを 1 バイトに変更します) に設定すれば変更できます。SET TYPE LONGWORD を指定すると、省略時の長さは 4 バイトに戻ります。

たとえば、次のような範囲にウォッチポイントを設定できます。

```
SET WATCH 30000:300018
```

デバッガは、この範囲をカバーするロングワード・ウォッチ群を設定します。

集合体 (すなわち、配列全体またはレコード全体) にウォッチポイントを設定できます。この場合、配列またはレコードのいずれかの要素が変化すると、配列またはレコードに設定したウォッチポイントが検出されます。このため、個々の配列要素やレコードの構成要素にウォッチポイントを設定する必要はありません。ただし、可変レコードには集合体のウォッチポイントを設定できないことに注意してください。

レコードの構成要素、個々の配列要素、配列断面 (ある範囲の配列要素) にもウォッチポイントを設定できます。この場合、配列断面内のいずれかの要素が変化すればウォッチポイントが検出されます。ウォッチポイントは、現在の言語の構文に従って設定します。

次の修飾子は、ウォッチポイントに達したときにどんな出力を表示するのかを決定します。

```
/[NO]SILENT
/[NO]SOURCE
```

次の修飾子は、ウォッチポイントのタイミングと期間を決定します。

```
/AFTER:n
/TEMPORARY
```

次の修飾子は非静的変数にだけ指定します。

```
/INTO
/OVER
```

次の修飾子は、デバッガによる変数が静的か非静的かの判定を上書きするのに使用します。

```
/[NO]STATIC
```

---

#### 注意

---

VAX システムでは、アドレスがグローバル・セクションにある変数に設定したウォッチポイントは動作しません。グローバル・セクション内の記憶位置にウォッチポイントを設定しようとすると、%DEBUG-E-BADWATCH メッセージが表示されます。

---

#### 静的ウォッチポイントと非静的ウォッチポイント

ウォッチポイントの設定方法は変数が静的か非静的かで異なります。

静的変数はプログラムの実行中ずっと同じメモリ・アドレスに関連づけられています。したがって、実行中は静的変数にウォッチポイントをずっと設定しておくことができます。



非静的変数は呼び出しスタックかレジスタに割り当てられ、その定義ルーチンがアクティブである場合だけ値を持ちます (呼び出しスタック上に存在します)。このため、定義ルーチン (定義ルーチンが呼び出した任意のルーチンを含む) の有効範囲内で実行が現在中断しているときだけ、非静的変数にウォッチポイントを設定できます。定義ルーチンから制御が戻ると、ウォッチポイントは取り消されます。非静的変数を使用した場合、デバuggは、すべての命令をトレースしてウォッチされた変数または位置に変更があるかどうかを検出します。

静的ウォッチポイントと非静的ウォッチポイントは実行速度も異なります。静的変数をウォッチする場合、デバuggはその変数を含むページを書き込み保護にします。プログラムがそのページに書き込もうとすると、アクセス違反が発生しデバuggが例外を処理し、ウォッチしている変数の内容が変更されたかどうかを調べます。そのページへの書き込みが発生しないかぎりプログラムは通常の実行速度で実行します。

非静的変数をウォッチするには、変数の定義ルーチン内の各命令をトレースし、各命令実行後に変数の値を調べます。このため、実行が極めて遅くなるので、非静的ウォッチポイントを設定すると、メッセージが発行されます。

次に説明するように、`/[NO]STATIC`、`/INTO`、`/OVER` を使用すれば、変数をウォッチするときに実行速度やその他の要因をある程度制御できます。

デバuggは、変数の割り当てを調べて、静的か非静的かを判断します。通常、静的変数は P0 空間 (0 から 3FFFFFFF, 16 進数) に、非静的変数は P1 空間 (40000000 から 7FFFFFFF) またはレジスタに割り当てられます。定義ルーチンの有効範囲内で現在実行が中断していないときに、P1 空間またはレジスタに割り当てられた変数にウォッチポイントを設定しようとする、デバuggは警告を出力します。

`/[NO]STATIC` 修飾子を指定すると、この省略時の動作を上書きできます。たとえば、P1 空間に非スタック記憶を割り当てた場合、その記憶領域に割り当てられた変数にウォッチポイントを設定するときには `/STATIC` を使用します。こうすれば、デバuggは各命令をトレースする代わりにもっと高速の書き込み保護による記憶位置のウォッチ方法を使用できます。たとえば、逆に P0 空間にユーザの呼び出しスタックを割り当てた場合、呼び出しスタックに割り当てられた変数にウォッチポイントを設定するときには、`/NOSTATIC` を使用します。こうすれば、デバuggはウォッチポイントを非静的ウォッチポイントとして扱うことができます。

`/INTO` および `/OVER` を使用すれば、呼び出されたルーチン内の非静的ウォッチポイントの実行速度も制御できます。

Alpha および VAX プロセッサの両方で、静的ウォッチポイントと非静的ウォッチポイントの両方が使用できます。静的ウォッチポイントを使用した場合、デバuggは、ウォッチする変数が保存されているメモリのページを書き込み保護します。このため、デバuggのシステム・サービス割り込み (SSI) のためでなければ、静的ウォッチポイントは、システム・サービス自体とインタフェースをとります。

静的ウォッチポイントが有効な場合、システム・サービス割り込みを通して、システム・サービス呼び出しの前にデバグは、静的ウォッチポイント、非同期トラップ (AST)、およびスレッド・スイッチを非アクティブにします。デバグは、システム・サービス呼び出しが完了するとすぐに、ウォッチポイント、使用可能な AST、スレッド・スイッチを元の状態に戻し、最後にウォッチポイントのヒットをチェックします。この動作は、ウォッチポイントが非アクティブにされても、システム・サービスが通常どおりに動作し (書き込み保護のページなし)、AST コードまたは異なるスレッドが、ウォッチポイントする位置に変更の可能性がないように設計されています。お使いのアプリケーションで AST が使用可能かどうかをテストする場合などは、この動作に気をつけてください。

システム・サービスが system service interception (SSI) 機能 (OpenVMS VAX システムでは DBGSSISHR、OpenVMS Alpha システムでは SYS\$SSISHR) でサポートされない場合、アクティブな静的ウォッチポイントが検出されると、システム・サービスは異常終了し、ほとんどの場合、ACCVIO 状態になります。SYS\$PUBLIC\_VECTORS に存在しないシステム・サービスは SSI でサポートされません。これにはユーザ作成システム・サービス (UWSS) や、\$MOUNT などのロード可能なシステム・サービスも含まれます。

静的ウォッチポイントがアクティブな場合には、デバグはウォッチされる変数を含むページを書き込み禁止にします。SSI でサポートされないシステム・サービス呼び出しは、ユーザ・メモリのそのページに書き込もうとしたときに、異常終了します。

この問題を回避するには、次のいずれかの処理を実行してください。

- サービス呼び出しの前に静的ウォッチポイントを無効にします。呼び出しが完了したら、ウォッチポイントを手動で確認し、再度有効にします。
- 非静的ウォッチポイントを使用します。ただし、非静的ウォッチポイントを使用すると、実行速度が低下する可能性があります。

システム・サービス・ルーチンの途中でウォッチしている記憶位置が変化した場合には、通常と同様に、ウォッチポイントが発生したことが通知されます。非常にまれですが、スタックでユーザ・プログラムのフレームの上に 1 つ以上のデバグ・フレームが表示されることがあります。この問題を回避するには、1 つ以上の STEP/RETURN コマンドを入力して、プログラムに戻ります。

省略時の設定では、system service interception (SSI) はオンですが、Alpha プロセッサの場合のみ、次のコマンドを使用して、デバグ・セッションの前にインターセプションを無効に設定できます。

```
$ DEFINE SSI$AUTO_ACTIVATE OFF
```

system service interception (SSI) を再度有効にするには、次のいずれかのコマンドを使用します。

```
$ DEFINE SSI$AUTO_ACTIVATE ON
$ DEASSIGN SSI$AUTO_ACTIVATE
```

グローバル・セクション・ウォッチポイント (Alpha および I64 のみ)

Alpha プロセッサでは、グローバル・セクションの変数または任意のプログラム記憶位置にウォッチポイントを設定できます。グローバル・セクションとは、マルチプロセス・プログラムのすべてのプロセスで共用されるメモリ領域です。グローバル・セクション内の記憶位置に設定されたウォッチポイント (グローバル・セクション・ウォッチポイント) は、プロセスがその記憶位置の内容を変更したときに起動されます。

グローバル・セクション・ウォッチポイントは、静的変数にウォッチポイントを設定するときと同じ方法で設定します。しかし、デバッガがグローバル・セクション・ウォッチポイントをモニタする方法により、次の点に注意する必要があります。ウォッチポイントを配列またはレコードに設定する場合には、SET WATCH コマンドを使用して構造体全体を指定するより、個々の要素を指定する方が性能を向上できます。

グローバル・セクションにまだマッピングされていない記憶位置にウォッチポイントを設定すると、そのウォッチポイントは通常の静的ウォッチポイントとして取り扱われます。そのあとで記憶位置がグローバル・セクションにマッピングされると、ウォッチポイントは自動的にグローバル・セクション・ウォッチポイントとして取り扱われるようになり、情報メッセージが出力されます。その後、ウォッチポイントはマルチプロセス・プログラムの各プロセスから確認できます。

関連コマンド

```
(ACTIVATE,DEACTIVATE,SHOW,CANCEL) WATCH
MONITOR
SET BREAK
SET STEP [NO]SOURCE
SET TRACE
```

---

## 例

1. DBG> SET WATCH MAXCOUNT

このコマンドは変数 MAXCOUNT にウォッチポイントを設定します。

```

2.  DBG> SET WATCH ARR
    DBG> GO
    . . .
    watch of SUBR\ARR at SUBR\%LINE 12+8
      old value:
        (1):      7
        (2):     12
        (3):      3
      new value:
        (1):      7
        (2):     12
        (3):     28

    break at SUBR\%LINE 14
    DBG>

```

この例では、SET WATCH コマンドが3つの要素を持つ整数の配列 ARR にウォッチポイントを設定します。GO コマンドで実行を再開します。いずれかの配列要素の内容が変化するとウォッチポイントが検出されます。この例では、3番目の要素が変化しています。

```

3.  DBG> SET WATCH ARR(3)

```

このコマンドは ARR 配列 (Fortran 配列構文) の要素 3 にウォッチポイントを設定します。要素 3 が変化するとウォッチポイントが検出されます。

```

4.  DBG> SET WATCH P_ARR[3:5]

```

このコマンドは、P\_ARR 配列 (Pascal 配列構文) の要素 3 ~ 5 で構成される配列断面にウォッチポイントを設定します。これらの要素のどれかが変化するとウォッチポイントが検出されます。

```

5.  DBG> SET WATCH P_ARR[3]:P_ARR[5]

```

このコマンドは、P\_ARR 配列の要素 3 ~ 5 のそれぞれに、個別にウォッチポイントを設定します (Pascal 配列構文)。ターゲット要素が変化すると、各ウォッチポイントが起動されます。

```

6.  DBG> SET TRACE/SILENT SUB2 DO (SET WATCH K)

```

この例では、変数 K は非静的変数であり、その定義ルーチン SUB2 (スタック上に存在します) がアクティブのときだけ定義されます。SET TRACE は SUB2 にトレースポイントを設定します。実行中にトレースポイントが検出されると、DO 句は K にウォッチポイントを設定します。SUB2 ルーチンから制御が戻るとウォッチポイントは取り消されます。/SILENT 修飾子が指定されているので、トレースポイントを検出したときの "trace . . . " メッセージとソース・コードは表示されません。

---

# SET WINDOW

画面ウィンドウ定義を作成します。

---

## 注意

このコマンドは、デバッガへの HP DECwindows Motif for OpenVMS ユーザ・インタフェースでは使用できません。

---

---

## フォーマット

```
SET WINDOW window-name  
          AT (start-line,line-count  
            [,start-column,column-count])
```

---

## パラメータ

*window-name*

定義するウィンドウの名前を指定します。その名前を持つウィンドウ定義がすでに存在するときには、古い定義は取り消され新しい定義が有効になります。

*start-line*

ウィンドウの開始行番号を指定します。この行はウィンドウの表題またはヘッダ行を表示します。画面の最上行は行 1 です。

*line-count*

ウィンドウのテキスト行の数を指定します。ヘッダ行は計算にいません。この値の最小値は 1 です。start-line と line-count の合計は現在の画面の高さを超えてはなりません。

*start-column*

ウィンドウの開始欄番号を指定します。これは、ウィンドウの最初の文字が表示される欄のことです。画面の最も左の欄は欄 1 です。

*column-count*

ウィンドウの行当たりの文字数を指定します。最小値は 1 です。start-column と column-count の合計が現在の画面幅を超えてはなりません。

---

説明

画面ウィンドウは端末画面上の長方形の領域で、これを通して表示を見ることができます。SET WINDOW コマンドはウィンドウ名と画面領域とを関連づけることによってウィンドウ定義を設定します。画面領域は、開始行と高さ (行数) で定義しますが、オプションで開始欄と幅 (欄数) を指定します。開始欄と欄数を指定しないと、省略時の設定により欄 1 と現在の画面幅に設定されます。

組み込みシンボル %PAGE と %WIDTH を使用した式でウィンドウ領域を指定することもできます。

SET WINDOW コマンドで定義したウィンドウの名前を DISPLAY コマンドで使って画面上の表示位置を指定することもできます。

ウィンドウ定義は動的に行われます。すなわち、SET TERMINAL コマンドで端末の幅と高さを変更すると、それに比例してウィンドウのサイズも拡大または縮小します。

## 関連コマンド

```
DISPLAY
(SHOW,CANCEL) DISPLAY
(SET,SHOW) TERMINAL
(SHOW,CANCEL) WINDOW
```

---

例

1. DBG> SET WINDOW ONELINE AT (1,1)

このコマンドは画面の最上部に ONELINE という名前のウィンドウを定義します。このウィンドウの高さは 1 行であり、幅は省略時の設定によって画面の幅に設定されます。

2. DBG> SET WINDOW MIDDLE AT (9,4,30,20)

このコマンドは画面の中央に MIDDLE という名前のウィンドウを定義します。このウィンドウは行 9 から開始して 4 行の高さであり、欄 30 から開始して 20 欄の幅に設定されます。

3. DBG> SET WINDOW FLEX AT (%PAGE/4,%PAGE/2,%WIDTH/4,%WIDTH/2)

このコマンドは画面の中央付近に FLEX という名前のウィンドウを定義します。このウィンドウは現在の画面の高さ (%PAGE) と幅 (%WIDTH) に定義されます。

---

## SHOW ABORT\_KEY

デバッガ・コマンドの実行を強制終了するか、またはプログラムの実行に割り込みをかける機能として現在定義されている Ctrl キー・シーケンスを示します。

---

### 注意

このコマンドは、デバッガへの HP DECwindows Motif for OpenVMS ユーザ・インタフェースでは使用できません。

---

---

## フォーマット

SHOW ABORT\_KEY

---

## 説明

省略時の設定では、デバッグ・セッション中に Ctrl/C を入力すると、デバッガ・コマンドの実行が強制終了され、プログラムの実行に割り込みがかかります。SET ABORT\_KEY コマンドを使用して、この強制終了機能を他の Ctrl キー・シーケンスに割り当てることができます。SHOW ABORT\_KEY コマンドは強制終了機能として現在有効になっている Ctrl キー・シーケンスを示します。

### 関連コマンド

Ctrl/C  
SET ABORT\_KEY

---

## 例

```
DBG> SHOW ABORT_KEY
Abort Command Key is CTRL_C
DBG> SET ABORT_KEY = CTRL_P
DBG> SHOW ABORT_KEY
Abort Command Key is CTRL_P
DBG>
```

この例では、最初の SHOW ABORT\_KEY コマンドは省略時の強制終了コマンドのキー・シーケンスである Ctrl/C を示します。SET ABORT\_KEY = CTRL\_P コマンドは強制終了コマンド機能を Ctrl/P に割り当て、2 番目の SHOW ABORT\_KEY コマンドがそれを確認します。

---

## SHOW AST

AST (asynchronous system traps) の実行要求が許可されているか、禁止されているかを示します。

---

### フォーマット

SHOW AST

---

### 説明

SHOW AST コマンドは AST の実行要求が許可されているか、禁止されているかを示します。このコマンドは実行要求が保留中になっている AST は示しません。AST の実行要求は省略時の設定は許可です。または、ENABLE AST コマンドを使用して許可します。AST の実行要求を禁止するには DISABLE AST コマンドを使用します。

関連コマンド

(ENABLE,DISABLE) AST

---

### 例

```
DBG> SHOW AST
ASTs are enabled
DBG> DISABLE AST
DBG> SHOW AST
ASTs are disabled
DBG>
```

SHOW AST コマンドは AST の実行要求が許可されているかどうかを示します。



---

# SHOW ATSIGN

SET ATSIGN コマンドで設定した省略時のファイル指定を示します。デバッガは@ (実行プロシージャ) コマンドの処理中にこのファイル指定を使用します。

---

## フォーマット

SHOW ATSIGN

---

## 説明

### 関連コマンド

@ (実行プロシージャ)  
SET ATSIGN

---

## 例

1. DBG> SHOW ATSIGN  
No indirect command file default in effect, using DEBUG.COM  
DBG>

この例は、SET ATSIGN コマンドを使用していなかった場合、デバッガがコマンド・プロシージャの省略時のファイル指定として SYS\$DISK:[ ]DEBUG.COM を使用することを示します。

2. DBG> SET ATSIGN USER:[JONES.DEBUG].DBG  
DBG> SHOW ATSIGN  
Indirect command file default is USER:[JONES.DEBUG].DBG  
DBG>

この例では、SHOW ATSIGN コマンドはコマンド・プロシージャの省略時のファイル指定を示しています。それは以前に SET ATSIGN コマンドで設定されたものです。

---

## SHOW BREAK

ブレークポイントに関する情報を表示します。

---

### フォーマット

SHOW BREAK

---

### 修飾子

/PREDEFINED

定義済みブレークポイントに関する情報を表示します。

/USER

ユーザ定義ブレークポイントに関する情報を表示します。

---

### 説明

SHOW BREAK コマンドは WHEN 句や DO 句、/AFTER の数などのオプションも含めて、現在設定されているブレークポイントに関する情報と、ブレークポイントが無効になっているかどうかを表示します。

省略時の設定では、SHOW BREAK はユーザ定義ブレークポイントと定義済みブレークポイントの両方 (ただし、存在する場合) に関する情報を表示します。これは SHOW BREAK/USER/PREDEFINED コマンドを入力した場合と同じです。ユーザ定義ブレークポイントは SET BREAK コマンドで設定されます。定義済みブレークポイントは、デバグを起動すると自動的に設定され、デバグの対象になっているプログラムの種類によって異なります。

SET BREAK/AFTER:*n* を使用してブレークポイントを設定した場合、SHOW BREAK コマンドは 10 進整数 *n* の現在の値、つまり最初に指定された整数値から、ブレークポイント記憶位置に到達するたびに 1 を引いた値を表示します (デバグは *n* の値が 0 になるまでブレークポイント記憶位置に到達するたびに *n* を減少させていきます。0 になると、デバグはブレーク動作を取ります)。

Alpha システムの場合、ブレークが命令の特定のクラスにあっても、SHOW BREAK コマンドは、個々の命令 (SET BREAK/CALL または SET BREAK /RETURN と同様) は表示しません。

## 関連コマンド

(ACTIVATE,CANCEL,DEACTIVATE,SET) BREAK

## 例

```

1.  DBG> SHOW BREAK
    breakpoint at SUB1\LOOP
    breakpoint at MAIN\MAIN+1F
      do (EX SUB1\D ; EX/SYMBOLIC PSL; GO)
    breakpoint at routine SUB2\SUB2
      /after: 2
    DBG>

```

SHOW BREAK コマンドは現在設定されているすべてのブレークポイントを示します。この例は、実行がそれぞれ SUB1\LOOP、MAIN\MAIN、SUB2\SUB2 に到達すると必ず検出されるユーザ定義ブレークポイントを示します。

```

2.  DBG> SHOW BREAK/PREDEFINED
    predefined breakpoint on Ada event "DEPENDENTS_EXCEPTION"
      for any value
    predefined breakpoint on Ada event "EXCEPTION_TERMINATED"
      for any value
    DBG>

```

このコマンドは現在設定されている定義済みブレークポイントを示します。この例では、2つの定義済みブレークポイントが表示されていて、その2つはAdaタスキング例外イベントに対応しています。これらのブレークポイントはすべてのAdaプログラムに対してとAdaモジュールにリンクされている混合言語プログラムに対して自動的にデバッガによって設定されます。

---

## SHOW CALLS

現在アクティブなルーチン呼び出しを示します。

---

### フォーマット

SHOW CALLS *[integer]*

---

### パラメータ

*integer*

表示するルーチンの数を指定する 10 進整数。このパラメータを省略すると、デバッガはデバッガ内に関係する情報があるすべてのルーチン呼び出しを示します。

---

### 修飾子

/IMAGE

呼び出しスタック上の個々のアクティブな呼び出しのイメージ名を表示します。

---

### 説明

SHOW CALLS コマンドは、実行が中断されたルーチンまでの、アクティブなルーチン呼び出しのシーケンスをリストするトレースバックを表示します。再帰的ルーチン呼び出しはすべて表示されるため、SHOW CALLS コマンドを使用して再帰呼び出しのチェーンをチェックできます。

SHOW CALLS は最新の呼び出しを先頭として、呼び出しスタック上の呼び出しフレーム 1 つにつき 1 行の情報を表示します。最上行は現在実行中のルーチンを示し、2 行目はその呼び出し元、3 行目は呼び出し元の呼び出し元を示すというようになります。

プログラムが最初に起動されるときにそのプログラム用のスタック・フレームが少なくとも 1 個作成されるため、プログラムでルーチン呼び出しを行っていない場合でも、SHOW CALLS コマンドはアクティブな呼び出しを表示します。

Alpha プロセッサと I64 プロセッサでは、通常、システム・フレームも表示され、場合によっては DCL ベース・フレームも表示されます。SHOW CALLS がアクティブな呼び出しを表示しない場合、その原因はプログラムが終了しているか、呼び出しスタックが破損しているかのいずれかです。プログラムの実行時にルーチンへの呼び出

しを行うと、スタックまたはレジスタ・セット上に新しい呼び出しフレームが作成されます。各呼び出しフレームには、呼び出し元のルーチンや現在のルーチンについての情報が格納されます。たとえば、フレームの PC 値によって、SHOW CALLS コマンドはモジュールとルーチンの情報をシンボル化することができます。

VAX プロセッサでは、ルーチン呼び出しのシーケンスは、メモリ・スタック上の呼び出しフレームのシーケンスに対応します。Alpha プロセッサでは、ルーチン呼び出しは、スタック・フレーム・プロシージャ (メモリ・スタック上に呼び出しフレームが作成される)、レジスタ・フレーム・プロシージャ (呼び出しフレームがレジスタ・セットに格納される)、空フレーム・プロシージャ (呼び出しフレームがない) のいずれかになります。

I64 プロセッサでは、ルーチン呼び出しは、メモリ・スタック・フレームまたはレジスタ・スタック・フレームとなります。つまり、I64 には、レジスタとメモリの 2 つのスタックがあります。I64 ルーチンを呼び出すと、これらのスタックのどちらか、またはその両方に呼び出しフレームが作成されます。また、I64 リーフ・ルーチンの呼び出し (それ自身は呼び出しを行わない) は、どちらのスタックにも呼び出しフレームが作成されない、空フレーム・プロシージャとなります。SHOW CALLS は、どのスタック・フレーム (メモリまたはレジスタ) であっても、情報を 1 行出力します。 (下記の例を参照してください。)

SHOW CALLS が示す行ごとに次の情報が提供されます。

- 呼び出しを行っているモジュールの名前。モジュール名の左側にアスタリスク(\*)が付いている場合、そのモジュールが設定されていることを表す。
- モジュールが設定されている場合、呼び出し元ルーチンの名前 (1 行目は現在実行中のルーチンを示す)。
- モジュールが設定されている場合、そのルーチンで呼び出しが行われた行番号 (1 行目は実行が中断される行番号を示す)。
- 制御が呼び出し先ルーチンに渡されたときの呼び出し元ルーチンでの PC の値。

VAX プロセッサでは、PC 値は直前のシンボル値 (たとえばルーチン) を基準としたメモリ・アドレスとして示され、さらに絶対アドレスとしても示されます。Alpha プロセッサと I64 プロセッサでは、PC はモジュール内の先頭コード・アドレスを基準としたメモリ・アドレスとして示され、さらに絶対アドレスとしても示されます。

/IMAGE 修飾子を指定すると、デバグはまずデバグ情報を持っている各イメージ (/DEBUG または /TRACEBACK 修飾子を使ってリンクされたもの) に対して SET IMAGE コマンドを実行します。その後、デバグは呼び出しスタック上の個々のアクティブな呼び出しについて、イメージ名を表示します。出力ディスプレイが拡大され、イメージ名が最初の欄に表示されるようになっています。

share\$image\_name モジュール名は /IMAGE 修飾子によって提供されるので、デバグはこの情報は表示しません。

SET IMAGE コマンドは SHOW CALLS/IMAGE コマンドが有効な間のみ作用します。デバッガは SHOW CALLS/IMAGE コマンドが完了したときに、セット・イメージ状態を復元します。

Alpha プロセッサと I64 プロセッサでは、SHOW CALLS コマンドの出力には、プログラムに関連付けられたユーザ呼び出しフレームの他に、システム呼び出しフレームも含まれることがあります。システム呼び出しフレームが含まれるのは、次の場合です。

- 例外のディスパッチの場合
- 非同期システム・トラップのディスパッチの場合
- システム・サービスのディスパッチの場合
- システム空間でウォッチポイントがトリガされた場合
- システム (インストールされている常駐 RTL を含む) 空間をステップ実行した場合
- 呼び出しスタックがベース・フレームの場合

システム呼び出しフレームが表示されても、問題を示すわけではありません。

関連コマンド

SHOW SCOPE  
SHOW STACK

---

## 例

```
1. DBG> SHOW CALLS
module name  routine name  line    rel PC    abs PC
      SUB2      SUB2              00000002  0000085A
*SUB1      SUB1              5      00000014  00000854
*MAIN      MAIN             10      0000002C  0000082C
DBG>
```

このコマンドは、VAX システム上で現在アクティブなプロシージャ呼び出しのシーケンスに関する情報を表示しています。

```
2. DBG> SHOW CALLS
module name  routine name  line    rel PC    abs PC
*MAIN      FFFF              31      0000000000002B8  00000000000203C4
-the above appears to be a null frame in the same scope as the frame below
*MAIN      MAIN             13      0000000000000A8  00000000000200A8
                                000000000000000  FFFFFFFF8255A1F8
```

これは、Alpha システムでの例です。ルーチンのプロローグおよびエピローグが空フレームとしてデバッガに表示されることに注意してください。フレーム・ポインタ (FP) が変更される前のプロローグの部分およびフレーム・ポインタ (FP) が回復された後のエピローグの部分は、それぞれ空フレームのように見えるため、空フレームと報告されます。

3. DBG> SHOW CALLS

module name	routine name	line	rel PC	abs PC
*MAIN	FFFF	18	00000000000000190	00000000000010190
*MAIN	MAIN	14	00000000000000180	00000000000010180
			FFFFFFFFF80C2A200	FFFFFFFFF80C2A200

これは、I64 システムでの例です。I64 プロローグは、デバッガからは空フレームとしては見えません。

---

## SHOW DEFINE

DEFINE コマンドに対して現在有効になっている省略時の値 (/ADDRESS , /COMMAND , /PROCESS\_GROUP , または/VALUE) を指定します。

---

### フォーマット

SHOW DEFINE

---

### 説明

DEFINE コマンドの省略時の修飾子は前回 SET DEFINE コマンドで設定された修飾子です。SET DEFINE コマンドを入力していなかった場合には、省略時の修飾子として/ADDRESS が使用されます。

DEFINE コマンドで定義されたシンボルを示すには、SHOW SYMBOL/DEFINED コマンドを使用します。

#### 関連コマンド

DEFINE  
DEFINE/PROCESS\_SET  
DELETE  
SET DEFINE  
SHOW SYMBOL/DEFINED

---

### 例

```
DBG> SHOW DEFINE  
Current setting is: DEFINE/ADDRESS  
DBG>
```

このコマンドは DEFINE コマンドがアドレスによって定義されるように設定されていることを通知します。



---

# SHOW DISPLAY

1 つまたは複数の既存の画面ディスプレイを示します。

---

## 注意

このコマンドは、デバッガへの HP DECwindows Motif for OpenVMS ユーザ・インタフェースでは使用できません。

---

---

## フォーマット

SHOW DISPLAY [*display-name*], ... ]]

---

## パラメータ

*display-name*

ディスプレイの名前を指定します。名前を指定しない場合またはワイルドカード文字のアスタリスク(\*)をそれだけで指定する場合、すべてのディスプレイ定義がリストされます。ディスプレイ名の中ではワイルドカードを使用できます。/ALL 修飾子を指定する場合、ディスプレイ名は指定できません。

---

## 修飾子

/ALL

すべてのディスプレイ定義をリストします。

---

## 説明

SHOW DISPLAY コマンドはディスプレイ・リストでの順番に従ってすべてのディスプレイをリストします。ディスプレイ・ペーストボードの一番下になっているディスプレイが最初にリストされ、ディスプレイ・ペーストボードの一番上にあるディスプレイが最後にリストされます。

1 つのディスプレイにつき、SHOW DISPLAY コマンドはその名前、最大サイズ、画面ウィンドウ、表示対象(デバッグ・コマンド・リストも含めて)をリストします。また、ディスプレイをペーストボードから削除するか、動的なものにするか(画面サイズが SET TERMINAL コマンドで変更された場合に、動的ディスプレイは自動的にそのウィンドウ寸法を調整する)も示します。

## 関連コマンド

DISPLAY  
EXTRACT/SCREEN\_LAYOUT  
(CANCEL) DISPLAY  
(SET,CANCEL,SHOW) WINDOW  
SHOW SELECT

---

例

```
DBG> SHOW DISPLAY
display SRC at H1, size = 64, dynamic
    kind = SOURCE (EXAMINE/SOURCE .%SOURCE_SCOPE\%PC)
display INST at H1, size = 64, removed, dynamic
    kind = INSTRUCTION (EXAMINE/INSTRUCTION .0\%PC)
display REG at RH1, size = 64, removed, dynamic, kind = REGISTER
display OUT at S45, size = 100, dynamic, kind = OUTPUT
display EXSUM at Q3, size = 64, dynamic, kind = DO (EXAMINE SUM)
display PROMPT at S6, size = 64, dynamic, kind = PROGRAM
DBG>
```

SHOW DISPLAY コマンドは現在定義されているディスプレイをすべてリストします。この例では、ディスプレイの中には、定義済みディスプレイが5つ (SRC , INST , REG , OUT , PROMPT) とユーザ定義 DO ディスプレイの EXSUM が含まれています。ディスプレイ INST と REG はディスプレイ・ペーストボードから削除されます。それらを画面上に表示するには DISPLAY コマンドを使用しなければなりません。

---

# SHOW EDITOR

SET EDITOR コマンドで設定された , EDIT コマンドが取る処置を示します。

---

## フォーマット

SHOW EDITOR

---

## 説明

関連コマンド

EDIT

SET EDITOR

---

## 例

1. 

```
DBG> SHOW EDITOR
The editor is SPANed, with command line
"EDT/START_POSITION=(n,1)"
DBG>
```

この例では , EDIT コマンドはサブプロセスで EDT エディタを作成します。コマンド行に付けられた /START\_POSITION 修飾子は , 初期状態では編集カーソルをデバッガの現在のソース表示の中央の行の先頭に置くことを表します。

2. 

```
DBG> SET EDITOR/CALLABLE_TPU
DBG> SHOW EDITOR
The editor is CALLABLE_TPU, with command line "TPU"
DBG>
```

この例では , SHOW EDITOR コマンドは EDIT コマンドが DEC Text Processing ユーティリティ (DECTPU) の呼び出し可能バージョンを起動することを表します。編集カーソルは初期状態ではソース行 1 の先頭に置かれます。

---

## SHOW EVENT\_FACILITY

現在のイベント機能と対応するイベント名を示します。

イベント機能は Ada ルーチンを呼び出すプログラムか、または POSIX Threads サービスを使用するプログラムの場合に使用できます。VAX プロセッサでは、イベント機能は SCAN ルーチンを呼び出すプログラムの場合にも使用できます

---

### フォーマット

SHOW EVENT\_FACILITY

---

### 説明

現在のイベント機能 (ADA , THREADS または SCAN) は SET BREAK/EVENT コマンドと SET TRACE/EVENT コマンドで設定できるイベントポイントを定義します。

SHOW EVENT\_FACILITY コマンドは現在のイベント機能に対応するイベント名を示します。それらのイベント名は (SET,CANCEL) BREAK/EVENT コマンドと (SET,CANCEL) TRACE/EVENT コマンドで指定できるキーワードです。

#### 関連コマンド

(SET,CANCEL) BREAK/EVENT  
SET EVENT\_FACILITY  
(SET,CANCEL) TRACE/EVENT  
SHOW BREAK  
SHOW TASK  
SHOW TRACE

---

### 例

```
DBG> SHOW EVENT_FACILITY
event facility is THREADS
...
```

このコマンドは現在のイベント機能が THREADS (POSIX Threads) であることを示し、それに対応する、SET BREAK/EVENT コマンドまたは TRACE/EVENT コマンドで使用するイベント名をリストします。

---

# SHOW EXIT\_HANDLERS

プログラムで宣言されている終了ハンドラを示します。

---

## フォーマット

SHOW EXIT\_HANDLERS

---

## 説明

終了ハンドラ・ルーチンが呼び出された順番 (つまり、後入れ先出し法) で表示されます。ルーチン名は可能な場合にはシンボルで表示されます。そうでない場合には、そのアドレスが表示されます。デバッガの終了ハンドラは表示されません。

---

## 例

```
DBG> SHOW EXIT_HANDLERS
exit handler at STACKS\CLEANUP
DBG>
```

このコマンドは終了ハンドラ・ルーチンの CLEANUP を示します。CLEANUP はモジュール STACKS で宣言されています。

---

## SHOW IMAGE

実行中のプログラムの一部になっている 1 つまたは複数のイメージに関する情報を表示します。

---

### フォーマット

SHOW IMAGE *[image-name]*

---

### パラメータ

image-name

表示にイれるイメージの名前を指定します。名前を指定しない場合またはワイルドカード文字のアスタリスク(\*)をそれだけで指定する場合、すべてのイメージがリストされます。イメージ名の中に\*を使用できます。

---

### 説明

SHOW IMAGE コマンドは次の情報を表示します。

- イメージの名前
- イメージの開始アドレスと終了アドレス
- イメージが SET IMAGE コマンドで設定されているか (実行時シンボル・テーブル RST にロードされているか)
- デバッグ・コンテキスト (アスタリスク(\*)でマークされる) になっている現在のイメージ
- 表示で選択されるイメージの総数
- RST やその他の内部構造に対して割り当てられているバイト数

SHOW IMAGE は /RESIDENT 修飾子を使用してインストールされているイメージのメモリ範囲すべてを表示するわけではありません。代わりに、このコマンドはプロセス・データ・リージョンだけを表示します。

関連コマンド

(SET,CANCEL) IMAGE  
(SET,SHOW) MODULE

---

例

```
DBG> SHOW IMAGE SHARE*
image name      set    base address  end address
*SHARE          yes    00000200    00000FFF
SHARE1          no     00001000    000017FF
SHARE2          yes    00018C00    000191FF
SHARE3          no     00019200    000195FF
SHARE4          no     00019600    0001B7FF

total images: 5    bytes allocated: 33032
DBG>
```

この SHOW IMAGE コマンドは名前が SHARE で始まり、プログラムに対応づけられているイメージすべてを示します。SHARE と SHARE2 のイメージは設定されています。アスタリスク(\*)は SHARE を現在のイメージとして示します。

---

## SHOW KEY

デバッグの定義済みキー定義と DEFINE/KEY コマンドで作成されたキー定義を表示します。

---

### 注意

このコマンドは、デバッグへの HP DECwindows Motif for OpenVMS ユーザ・インタフェースでは使用できません。

---



---

## フォーマット

SHOW KEY *[key-name]*

---

## パラメータ

key-name

その定義を表示するファンクション・キーを指定します。ワイルドカード文字のアスタリスク(\*)は使用できません。代わりに/ALL 修飾子を使用してください。/ALL または/DIRECTORY を指定する場合は、キー名は指定できません。有効なキー名は次のとおりです。

キー名	LK201 キーボード	VT100 型	VT52 型
PF1	PF1	PF1	Blue
PF2	PF2	PF2	Red
PF3	PF3	PF3	Black
PF4	PF4	PF4	
KP0-KP9	キーパッド 0 ~ 9	キーパッド 0 ~ 9	キーパッド 0 ~ 9
PERIOD	キーパッドのピリオド(.)	キーパッドのピリオド(.)	
COMMA	キーパッドのコンマ(,)	キーパッドのコンマ(,)	
MINUS	キーパッドのマイナス(-)	キーパッドのマイナス(-)	
ENTER	Enter	ENTER	ENTER
E1	Find		
E2	Insert Here		
E3	Remove		
E4	Select		



キー名	LK201 キーボード	VT100 型	VT52 型
E5	Prev Screen		
E6	Next Screen		
HELP	Help		
DO	Do		
F6-F20	F6-F20		

## 修飾子

/ALL

省略時の設定では現在の状態に対するすべてのキー定義を表示し、そうでない場合には/STATE で指定された状態に対するすべてのキー定義を表示します。

/BRIEF

キー定義だけを表示します (省略時の設定では、指定された状態も含めて、キー定義に対応するすべての修飾子も表示されます)。

/DIRECTORY

対応するキーが定義されているすべての状態の名前を表示します。この修飾子を指定する場合、他の修飾子は指定できません。

/STATE=(state-name [, . . . ])

/NOSTATE (省略時の設定)

対応するキー定義が表示される 1 つまたは複数の状態を選択します。/STATE 修飾子を指定すると、指定された状態に対するキー定義を表示します。DEFAULT や GOLD などの定義済みキー状態、またはユーザ定義状態を指定できます。状態名には任意の適切な英数字文字列を指定できます。/NOSTATE 修飾子を指定すると、現在の状態に対するキー定義だけが表示されます。

## 説明

このコマンドを使用するには、その前にキーパッド・モードが有効 (SET MODE KEYPAD) に設定されていなければなりません。省略時の設定では、キーパッド・モードは有効になっています。

省略時の設定では、現在のキー状態は DEFAULT 状態です。SET KEY/STATE コマンドを使用するか、または状態を変更するキー (つまり、DEFINE/KEY/LOCK\_STATE または/SET\_STATE で定義されたキー) を押すことにより現在の状態を変更できます。

関連コマンド

DEFINE/KEY

**DELETE/KEY**  
**SET KEY**

---

**例**

1. DBG> SHOW KEY/ALL

このコマンドは現在の状態に対するすべてのキー定義を表示します。

2. DBG> SHOW KEY/STATE=BLUE KP8  
GOLD keypad definitions:  
KP8 = "Scroll/Top" (noecho,terminate,nolock)  
DBG>

このコマンドはBLUE状態のキーパッド・キー8の定義を表示します。

3. DBG> SHOW KEY/BRIEF KP8  
DEFAULT keypad definitions:  
KP8 = "Scroll/Up"  
DBG>

このコマンドは現在の状態のキーパッド・キー8の定義を表示します。

4. DBG> SHOW KEY/DIRECTORY  
MOVE\_GOLD  
MOVE\_BLUE  
MOVE\_  
GOLD  
EXPAND\_GOLD  
EXPAND\_BLUE  
EXPAND\_  
DEFAULT  
CONTRACT\_GOLD  
CONTRACT\_BLUE  
CONTRACT\_  
BLUE  
DBG>

このコマンドは対応するキーが定義されている状態の名前を表示します。

---

# SHOW LANGUAGE

現在の言語を示します。

---

## フォーマット

SHOW LANGUAGE

---

## 説明

現在の言語は前回 SET LANGUAGE コマンドで設定された言語です。SET LANGUAGE コマンドを入力しなかった場合、省略時の設定では現在の言語はメイン・プログラムを含んでいるモジュールの言語です。

関連コマンド

SET LANGUAGE

---

## 例

```
DBG> SHOW LANGUAGE
language: BASIC
DBG>
```

このコマンドは現在の言語の名前を BASIC と表示します。

---

## SHOW LOG

デバッガがログ・ファイルに書き込みを行っているのかどうかを示し、現在のログ・ファイルを示します。

---

### フォーマット

SHOW LOG

---

### 説明

現在のログ・ファイルは前回 SET LOG コマンドで設定されたログ・ファイルです。省略時の設定では、SET LOG コマンドを入力していなかった場合、現在のログ・ファイルは SYSSDISK:[ ]DEBUG.LOG ファイルになります。

#### 関連コマンド

SET LOG  
SET OUTPUT [NO]LOG  
SET OUTPUT [NO]SCREEN\_LOG

---

### 例

1. DBG> SHOW LOG  
not logging to DEBUG.LOG  
DBG>

このコマンドは現在のログ・ファイルの名前を DEBUG.LOG (省略時のログ・ファイル) と表示し、デバッガがそのファイルに書き込みを行っていないことを通知します。

2. DBG> SET LOG PROG4  
DBG> SET OUTPUT LOG  
DBG> SHOW LOG  
logging to USER\$:[JONES.WORK]PROG4.LOG  
DBG>

この例では、SET LOG コマンドは現在のログ・ファイルが (現在の省略時のディレクトリにある) PROG4.LOG であることを設定します。SET OUTPUT LOG コマンドを実行すると、デバッガはそのファイルにデバッガの入出力を記録します。SHOW LOG コマンドはデバッガが現在の省略時のディレクトリにあるログ・ファイル PROG4.COM に書き込み中であることを確認します。

---

# SHOW MARGINS

ソース・コードを表示するための現在のソース行のマージン設定を示します。

---

## 注意

このコマンドは、デバッガへの HP DECwindows Motif for OpenVMS ユーザ・インタフェースでは使用できません。

---

---

## フォーマット

SHOW MARGINS

---

## 説明

現在のマージン設定は前回 SET MARGINS コマンドで設定されたマージン設定です。省略時の設定では、SET MARGINS コマンドを入力しなかった場合、左マージンは 1 に、右マージンは 255 に設定されます。

関連コマンド

SET MARGINS

---

## 例

1. DBG> SHOW MARGINS  
left margin: 1 , right margin: 255  
DBG>

このコマンドは省略時のマージン設定として 1 と 255 を表示します。

2. DBG> SET MARGINS 50  
DBG> SHOW MARGINS  
left margin: 1 , right margin: 50  
DBG>

このコマンドは省略時の左マージン設定として 1 を、変更された右マージン設定として 50 を表示します。

## SHOW MARGINS

```
3. DBG> SET MARGINS 10:60
DBG> SHOW MARGINS
left margin: 10 , right margin: 60
DBG>
```

このコマンドはそれぞれ 10 と 60 に変更された左右のマージン設定を表示します。

---

## SHOW MODE

現在のデバッガ・モード (画面ありまたは画面なし, キーパッドありまたはキーパッドなしなど) と現在の基数を示します。

---

### フォーマット

SHOW MODE

---

### 説明

現在のデバッガ・モードは前回 SET MODE コマンドで設定されたモードです。省略時の設定では, SET MODE コマンドを入力していなかった場合, 現在のモードは次のようになります。

DYNAMIC  
NOG\_FLOAT (D 浮動の小数点数)  
INTERRUPT  
KEYPAD  
LINE  
NOSCREEN  
SCROLL  
NOSEPARATE  
SYMBOLIC

#### 関連コマンド

(SET,CANCEL) MODE  
(SET,SHOW,CANCEL) RADIX

---

### 例

```
DBG> SHOW MODE
modes: symbolic, line, d_float, screen, scroll, keypad,
      dynamic, interrupt, no separate window
input radix :decimal
output radix:decimal
DBG>
```

SHOW MODE コマンドは現在のモードと現在の入出力基数を表示します。

---

# SHOW MODULE

現在のイメージでのモジュールに関する情報を表示します。

---

## フォーマット

SHOW MODULE *[module-name]*

---

## パラメータ

module-name

表示にいれるモジュールの名前を指定します。名前を指定しない場合またはワイルドカード文字のアスタリスク(\*)をそれだけで指定する場合、すべてのモジュールがリストされます。モジュール名の中にはワイルドカードを使用できます。/SHARE を指定する場合だけ、共用可能イメージ・モジュールが選択されます。

---

## 修飾子

/RELATED

/NORELATED (省略時の設定)

(Ada プログラムに適用される。) 指定されたモジュールに WITH 句またはサブユニット関係によって関連づけられたモジュールがあればそれをデバッガが SHOW MODULE 表示の中にいれるかどうかを制御します。

SHOW MODULE/RELATED コマンドは指定されたモジュールだけでなく、関連モジュールも表示します。表示は正確な関係を示します。省略時の設定 (/NORELATED) では、関連モジュールは表示の対象として選択されません (指定されたモジュールだけが選択されます)。

/SHARE

/NOSHARE (省略時の設定)

プログラムにリンクされている共用可能イメージをデバッガが SHOW MODULE 表示にいれるかどうかを制御します。省略時の設定 (/NOSHARE) では、共用可能イメージ・モジュールは表示の対象として選択されません。

デバッガはプログラム内の共用可能イメージごとにダミー・モジュールを作成します。これらの共用可能"イメージ・モジュール"の名前には接頭辞として"SHARE\$"が付きます。SHOW MODULE/SHARE コマンドは現在のイメージ内のモジュールだけでなく、これらの共用可能イメージ・モジュールも示します。



共用可能イメージ・モジュールを設定すると、そのイメージに対するユニバーサル・シンボルが実行時シンボル・テーブルにロードされ、現在のイメージからそれらのシンボルを参照できるようになります。ただし、現在のイメージからそのイメージ内の他の(ローカルまたはグローバルな)シンボルを参照することはできません。この機能は新しいSET IMAGE コマンドやSHOW IMAGE コマンドの働きと重なります。

---

## 説明

SHOW MODULE コマンドは表示の対象として選択された 1 つまたは複数のモジュールに関して次の情報を表示します。

- モジュールの名前
- モジュールがコード化されたプログラミング言語、ただしすべてのモジュールが同一の言語でコード化されている場合を除く
- モジュールがSET MODULE コマンドで設定されているかどうか。つまり、モジュールのシンボル・レコードがデバッガの実行時シンボル・テーブル(RST)にロードされているかどうか
- モジュール内のシンボル・レコードに対して RST で必要となるスペース(バイト単位)
- 表示で選択されたモジュールの総数
- RST やその他の内部構造用に割り当てられたバイト数(メイン・デバッガのプロセスで使用するヒープ・スペースの量)

---

### 注意

---

現在のイメージは(省略時の設定では)メイン・イメージか、前回のSET IMAGE コマンドで現在のイメージとして設定されたイメージです。

---

Ada プログラムに特有の情報については、ヘルプ・トピックLanguage\_Support Adaを参照してください。

### 関連コマンド

```
(SET,SHOW,CANCEL) IMAGE
SET MODE [NO]DYNAMIC
(SET) MODULE
(SET,SHOW,CANCEL) SCOPE
SHOW SYMBOL
```

## 例

1. DBG> SHOW MODULE

module name	symbols	size
TEST	yes	432
SCREEN_IO	no	280

total PASCAL modules: 2.    bytes allocated: 2740.  
DBG>

この例では、SHOW MODULE コマンドにはパラメータが指定されておらず、現在のイメージ内のモジュールすべてに関する情報を表示します。現在のイメージは省略時の設定ではメイン・イメージです。この例は、すべてのモジュールが同一のソース言語を持っているときの表示形式を示します。"symbols" 欄は TEST モジュールは設定されていても、SCREEN\_IO モジュールは設定されていないことを示します。

2. DBG> SHOW MODULE FOO,MAIN,SUB\*

module name	symbols	language	size
FOO	yes	MACRO	432
MAIN	no	FORTTRAN	280
SUB1	no	FORTTRAN	164
SUB2	no	FORTTRAN	204

total modules: 4.    bytes allocated: 60720.  
DBG>

この例では、SHOW MODULE コマンドは FOO モジュールと MAIN モジュールに関する情報と接頭辞として SUB を持つすべてのモジュールに関する情報を表示します。この例は、モジュールが同一のソース言語を持っていないときの表示形式を示します。

3. DBG> SHOW MODULE/SHARE

module name	symbols	language	size
FOO	yes	MACRO	432
MAIN	no	FORTTRAN	280
...			
SHARE\$DEBUG	no	Image	0
SHARE\$LIBRTL	no	Image	0
SHARE\$MTHRTL	no	Image	0
SHARE\$SHARE1	no	Image	0
SHARE\$SHARE2	no	Image	0

total modules: 17.    bytes allocated: 162280.  
DBG> SET MODULE SHARE\$SHARE2  
DBG> SHOW SYMBOL \* IN SHARE\$SHARE2

この例では、SHOW MODULE/SHARE コマンドは現在のイメージ内のモジュールすべてと共用可能イメージすべて (共用可能イメージの名前にはその前に SHARE\$が付きます) を示します。SET MODULE SHARE\$SHARE2 コマンドは共用可能イメージ・モジュールの SHARE\$SHARE2 を設定します。SHOW

SYMBOL コマンドは共用可能イメージ SHARE2 に定義されたユニバーサル・シンボルを示します。

---

## SHOW OUTPUT

現在の出力オプションを示します。

---

### フォーマット

SHOW OUTPUT

---

### 説明

現在の出力オプションは SET OUTPUT コマンドで前回設定したオプションです。省略時の設定では、SET OUTPUT コマンドを入力しなかった場合、出力オプションは NOLOG、NOSCREEN\_LOG、TERMINAL、NOVERIFY です。

#### 関連コマンド

SET LOG  
SET MODE SCREEN  
SET OUTPUT

---

### 例

```
DBG> SHOW OUTPUT
noverify, terminal, screen_log,
    logging to USER$:[JONES.WORK]DEBUG.LOG;9
DBG>
```

このコマンドは次の現在の出力オプションを示します。

- デバッガ・コマンド・プロシージャから読み込まれたデバッガ・コマンドは端末にエコーバックされない。
- デバッガ出力が端末に表示されている最中である。
- デバッグ・セッションはログ・ファイル USER\$:[JONES.WORK]DEBUG.LOG;9 に記録されている最中である。
- 画面の内容は画面モードで更新されると同時に記録される。

---

# SHOW PROCESS

現在デバッガの制御下にあるプロセスについての情報を表示します。

---

## フォーマット

SHOW PROCESS *[process-spec], ... ]]*

---

## パラメータ

*process-spec*

現在デバッガの制御下にあるプロセスを指定します。次のいずれかの形式で指定します。

*[%PROCESS\_NAME] process-name*

スペースや小文字を含まないプロセス名。プロセス名にはワイルドカード文字(\*)を含めることができる。

*[%PROCESS\_NAME] "process-name"*

スペースまたは小文字を含むプロセス名。二重引用符(")の代わりに、一重引用符(') 使用することもできる。

*%PROCESS\_PID process\_id*

プロセス識別子 (PID, 16 進数)。

*[%PROCESS\_NUMBER] process-number*  
(または *%PROC process-number*)

デバッガの制御下に入ったときにプロセスに割り当てられた番号。新しい番号は、1 から順番に各プロセスに割り当てられる。EXIT コマンドまたは QUIT コマンドによってプロセスが終了した場合、そのデバッグ・セッション中にその番号が再割り当てされることがある。プロセス番号は SHOW PROCESS コマンドの実行で表示される。プロセスは、組み込みシンボル *%PREVIOUS\_PROCESS* および *%NEXT\_PROCESS* によってインデックスづけできるように、循環リスト内に順序づけされる。

*process-set-name*

DEFINE/PROCESS\_SET コマンドで定義された、プロセスのグループを表すシンボル。

*%NEXT\_PROCESS*

デバッガの循環プロセス・リスト中で可視プロセスの次のプロセス。

*%PREVIOUS\_PROCESS*

デバッガの循環プロセス・リスト中で可視プロセスの前のプロセス。

*%VISIBLE\_PROCESS*

シンボル、レジスタ値、ルーチン呼び出し、ブレークポイントなどの検索時に現在のコンテキストになっているスタック、レジスタ・セット、およびイメージを持つプロセス。

すべてのプロセスを指定するためにワイルドカード文字のアスタリスク(\*)または/ALL 修飾子を使用することもできます。/ALL または/DYNAMIC を指定する場合、プロセスは指定できません。プロセスを指定しないか、/ALL を/BRIEF、/FULL または/[NO]HOLD とともに指定しないと、可視プロセスが選択されます。

---

### 修飾子

/ALL

表示の対象としてデバッガが知っているプロセスすべてを選択します。

/BRIEF

(省略時の設定。) 表示の対象として選択されたプロセス 1 つにつき 1 行だけの情報を表示します。

/DYNAMIC

動的プロセス設定が有効と無効のどちらになっているかを示します。動的プロセス設定は省略時の設定では有効になっていて、SET PROCESS/[NO]DYNAMIC コマンドで制御されます。

/FULL

表示の対象として選択されたプロセスごとに最大の情報を表示します。

/VISIBLE

(省略時の設定。) 表示の対象として可視プロセスを選択します。

---

### 説明

SHOW PROCESS コマンドは指定されたプロセスとそれらのプロセスで実行中のイメージに関する情報を表示します。

SHOW PROCESS/FULL コマンドはベクタ型プロセッサの可用性と使用法に関する情報也表示します。この情報は、ベクタ命令を使用するプログラムをデバッグする場合に役立ちます。

プロセスはデバッガの制御下に置かれるとすぐ、まず SHOW PROCESS 表示に表示できます。プロセスは EXIT コマンドまたは QUIT コマンドによって終了されると、SHOW PROCESS コマンドを実行しても表示できなくなります。

省略時の設定 (/BRIEF) では、次の情報も含め、1 つのプロセスにつき 1 行の情報が表示されます。

- デバッガにより割り当てられるプロセス番号。プロセス番号はデバッガの制御下に置かれる各プロセスにプロセス 1 から順番に割り当てられます。プロセスが EXIT コマンドまたは QUIT コマンドで終了される場合、そのプロセス番号はデ

バグ・セッション中には再使用されません。可視プロセスには左端の欄にアスタリスク(\*)がマークされます。

- プロセス名。
- 当該プロセスに対する現在のデバッグ状態 (表 2-2 を参照)。
- イメージが当該プロセスで中断される記憶位置 (可能な場合には、シンボル化されている)。

表 2-2 デバッグ状態

状態	説明
Activated	イメージとそのプロセスがデバッガの制御下に置かれている。
Break	ブレークポイントが検出された。
Break on branch	
Break on call	
Break on instruction	
Break on lines	
Break on modify of	
Break on return	
Exception break	
Exception break preceding	
Interrupted	実行が他のプロセスで中断されているためか、またはユーザが強制終了キー・シーケンス (省略時の設定では Ctrl/C) を使用してプログラムの実行に割り込みをかけたために実行がそのプロセスで割り込みをかけられた。
Step	コマンドが完了した。
Step on return	
Terminated	指示されたイメージは実行を終了したが、そのプロセスは依然としてデバッガの制御下にある。したがって、イメージとそのプロセスに関する情報を獲得できる。プロセスを終了するには、EXIT コマンドまたは QUIT コマンドを使用できる。
Trace	トレースポイントが検出された。
Trace on branch	
Trace on call	
Trace on instruction	
Trace on lines	
Trace on modify of	
Trace on return	
Exception trace	
Exception trace preceding	
Unhandled exception	未処理例外が検出された。
Watch of	ウォッチポイントが検出された。

SHOW PROCESS/FULL コマンドはプロセスに関する補足的な情報を表示します。(例を参照してください。)

#### 関連コマンド

CONNECT  
Ctrl/C  
DEFINE/PROCESS\_SET

EXIT  
QUIT  
SET PROCESS

---

例

1. all> SHOW PROCESS

```

Number Name           State      Current PC
*   2 _WTA3:          break      SCREEN\%LINE 47
all>
```

省略時の設定では、SHOW PROCESS コマンドは左端の欄に可視プロセス (アスタリスク(\*)で示される) に関する情報を 1 行表示します。可視プロセスはプロセス名として\_WTA3: を持っています。\_WTA3: はデバッガの制御下に置かれた 2 番目のプロセス (プロセス番号 2) です。このプロセスは凍結状態になっていて、イメージの実行は SCREEN モジュールの 47 行目のブレークポイントで中断されま

す。

2. all> SHOW PROCESS/FULL %PREVIOUS\_PROCESS

```

Process number: 1          Process name: JONES_1:
                          Visible process: NO

Current PC: TEST_VALVES\%LINE 153
State: interrupted
PID: 20400885              Owner PID: 00000000
Current/Base priority: 5/4  Terminal: VTA79:

Image name: USER$:[JONES.PROG1]TEST_VALVES.EXE;31

Elapsed CPU time: 0 00:03:17.17 CPU Limit:      Infinite
Buffered I/O Count:      14894 Remaining buffered I/O quota:      80
Direct I/O Count:        6956 Remaining direct I/O quota:      40
Open file count:          7   Remaining open file quota:      43
Enqueue count:            200 Remaining enqueue quota:      198
Vector capable:           Yes
Vector consumer:          Yes Vector CPU time:      00:00:00.00
Fast Vector context switches: 0 Slow Vector context switches: 0
Current working set size: 1102 Working set size quota:      1304
Current working set extent: 12288 Maximum working set extent: 12288
Peak working set size:    4955 Maximum authorized working set: 1304
Current virtual size:     255 Peak virtual size:      16182
Page faults:              41358

Active ASTs:               Remaining AST Quota:      27
Event flags: FF800000 60000003 Event flag wait mask: 7FFFFFFF
all>
```

SHOW PROCESS/FULL %PREVIOUS\_PROCESS コマンドはプロセスの循環リスト中の前のプロセス (この場合にはプロセス番号 1) に関する最大レベルの情報を表示します。



```
3. all> SHOW PROCESS TEST_3
   Number Name      State      Current PC
     7 TEST_3      watch of TEST_3\ROUT4\COUNT
                                TEST_3\%LINE 54

all>
```

この SHOW PROCESS コマンドは TEST\_3 プロセスに関する情報を 1 行表示します。イメージは変数 COUNT のウォッチポイントで中断されます。

```
4. all> SHOW PROCESS/DYNAMIC
   Dynamic process setting is enabled

all>
```

このコマンドは動的プロセス設定が有効になることを表します。

---

## SHOW RADIX

整数データの入力や表示のための現在の基数を示します。/OVERRIDE が指定された場合には、現在の上書き型の基数を示します。

---

### フォーマット

SHOW RADIX

---

### 修飾子

/OVERRIDE

現在の上書き型の基数を示します。

---

### 説明

デバッグは 2 進数、10 進数、16 進数、8 進数の 4 つの基数のどれか 1 つで整数データを解釈したり表示したりできます。整数データの入力や表示のための現在の基数は前回 SET RADIX コマンドで設定した基数です。

SET RADIX コマンドを入力していなかった場合、データの入力や表示のための省略時の基数はほとんどの言語の場合 10 進数です。例外は BLISS と MACRO です。これらの言語での省略時の基数は 16 進数です。

すべてのデータの表示用の現在の上書き型の基数は前回 SET RADIX/OVERRIDE コマンドで設定した上書き型の基数になります。SET RADIX/OVERRIDE コマンドを入力していなかった場合、変更型の基数は"ありません"。

#### 関連コマンド

DEPOSIT  
EVALUATE  
EXAMINE  
(SET,CANCEL) RADIX

---

## 例

1. 

```
DBG> SHOW RADIX
input radix: decimal
output radix: decimal
DBG>
```

このコマンドは入力基数と出力基数を 10 進数として表示します。

2. 

```
DBG> SET RADIX/OVERRIDE HEX
DBG> SHOW RADIX/OVERRIDE
output override radix: hexadecimal
DBG>
```

この例では、SET RADIX/OVERRIDE コマンドは上書き型の基数を 16 進数に設定し、SHOW RADIX/OVERRIDE コマンドは上書き型の基数を示します。これは、EXAMINE などのコマンドがすべてのデータを 16 進整数データとして表示することを意味します。

---

## SHOW SCOPE

シンボル検索のための現在の有効範囲検索リストを示します。

---

### フォーマット

SHOW SCOPE

---

### 説明

現在の有効範囲検索リストは、デバッガ・コマンドにパス名接頭辞を付けずに指定されるシンボルの解釈に使用する (パス名またはその他の特殊文字で指定される) 1 つまたは複数のプログラム記憶位置を指定します。

現在の有効範囲検索リストは前回 SET SCOPE コマンドで設定された有効範囲検索リストです。省略時の設定では、SET SCOPE コマンドが入力されていなかった場合、現在の有効範囲検索リストは 0,1,2, ...,  $n$  になります。

省略時の有効範囲検索リストは、パス名接頭辞を持たないシンボルに対して、EXAMINE X のようなシンボル検索の場合、最初に現在実行中のルーチン (有効範囲 0) で X を検索し、そこで X が可視になっていなければ、そのルーチンの呼び出し元 (有効範囲 1) を検索していくというように、呼び出しスタックを順々に検索していくことを指定します。有効範囲  $n$  にも X が見つけれなければ、デバッガは実行時シンボル・テーブル (RST) の残り、つまり、設定されているすべてのモジュールと必要であればグローバル・シンボル・テーブル (GST) を検索します。

呼び出しスタックのルーチンを表現するために SET SCOPE コマンドに 10 進整数を使用した場合には、SHOW SCOPE コマンドは可能であれば、整数で表現されるルーチン名を表示します。

### 関連コマンド

(SET,CANCEL) SCOPE

## 例

```

1.  DBG> CANCEL SCOPE
    DBG> SHOW SCOPE
    scope:
      * 0 [ = EIGHTQUEENS\TRYCOL\REMOVEQUEEN ],
        1 [ = EIGHTQUEENS\TRYCOL ],
        2 [ = EIGHTQUEENS\TRYCOL 1 ],
        3 [ = EIGHTQUEENS\TRYCOL 2 ],
        4 [ = EIGHTQUEENS\TRYCOL 3 ],
        5 [ = EIGHTQUEENS\TRYCOL 4 ],
        6 [ = EIGHTQUEENS ]
    DBG> SET SCOPE/CURRENT 2
    DBG> SHOW SCOPE
    scope:
      0 [ = EIGHTQUEENS\TRYCOL\REMOVEQUEEN ],
      1 [ = EIGHTQUEENS\TRYCOL ],
      * 2 [ = EIGHTQUEENS\TRYCOL 1 ],
        3 [ = EIGHTQUEENS\TRYCOL 2 ],
        4 [ = EIGHTQUEENS\TRYCOL 3 ],
        5 [ = EIGHTQUEENS\TRYCOL 4 ],
        6 [ = EIGHTQUEENS ]
    DBG>

```

CANCEL SCOPE コマンドは省略時の有効範囲検索リストを復元します。省略時の有効範囲検索リストは(最初の) SHOW SCOPE コマンドで表示されます。この例では、ルーチン TRYCOL への数回の再帰呼び出しのあと、実行は REMOVEQUEEN ルーチンで中断されます。アスタリスク(\*)は有効範囲検索リストが有効範囲 0、つまり実行が中断されるルーチンの有効範囲で始まることを示します。

例の SET SCOPE/CURRENT コマンドは有効範囲検索リストの始まりを有効範囲 2 に再設定します。有効範囲 2 は実行が中断されるルーチンの呼び出し元の有効範囲です。(2 番目の) SHOW SCOPE コマンドの出力に付いているアスタリスクは有効範囲検索リストが今度は有効範囲 2 で始まることを表します。

```

2.  DBG> SET SCOPE 0,STACKS\R2,SCREEN_IO,\
    DBG> SHOW SCOPE
    scope:
      0, [= TEST ],
      STACKS\R2,
      SCREEN_IO,
      \
    DBG>

```

この例では、SET SCOPE コマンドはデバッガに次の有効範囲検索リストに従ってパス名接頭辞を持たないシンボルを検索するように指示します。最初に、デバッガは PC 範囲 (0 で示され、モジュール TEST にある) を検索します。PC 範囲内に指定されたシンボルが見つからないと、次にモジュール STACKS のルーチン

## SHOW SCOPE

R2 を検索します。必要であれば、次にモジュール SCREEN\_IO を検索し、最後に (グローバル有効範囲(\))で示される) グローバル・シンボル・テーブルを検索します。SHOW SCOPE コマンドはシンボル検索のための現在の有効範囲検索リストを示します。省略時の有効範囲検索リストが有効になっているか、または SET SCOPE/CURRENT コマンドを入力した場合を除き、SHOW SCOPE 表示にアスタリスクは示されません。

---

## SHOW SEARCH

SEARCH コマンドに対して現在有効になっている省略時の修飾子 (/ALL か/NEXT , /IDENTIFIER か/STRING) を示します。

---

### フォーマット

SHOW SEARCH

---

### 説明

SEARCH コマンドの省略時の修飾子は前回 SET SEARCH コマンドで設定された省略時の修飾子です。SET SEARCH コマンドを入力していなかった場合には、省略時の修飾子として/NEXT と/STRING が使用されます。

#### 関連コマンド

SEARCH  
(SET,SHOW) LANGUAGE  
SET SEARCH

---

### 例

```
DBG> SHOW SEARCH
search settings: search for next occurrence, as a string
DBG> SET SEARCH IDENT
DBG> SHOW SEARCH
search settings: search for next occurrence, as an identifier
DBG> SET SEARCH ALL
DBG> SHOW SEARCH
search settings: search for all occurrences, as an identifier
DBG>
```

この例では、最初の SHOW SEARCH コマンドは SET SEARCH コマンドの省略時の設定を表示します。省略時の設定では、デバッガは指定文字列が次に現れる箇所を検索し、表示します。

2 番目の SHOW SEARCH コマンドは、デバッガが指定文字列の次に現れる箇所を検索するものの、その前後が現在の言語の識別子の一部になり得る文字に接していない場合だけその文字列を表示することを通知します。

## SHOW SEARCH

3 番目の SHOW SEARCH コマンドはデバッガが指定文字列の現れる箇所すべてを検索するものの、その前後が現在の言語の識別子の一部になり得る文字に接していない場合だけそれらの文字列を表示することを通知します。



---

## SHOW SELECT

ディスプレイ属性、つまりエラー、入力、命令、出力、プログラム、プロンプト、スクロール、ソースのそれぞれに対して現在選択されているディスプレイを示します。

---

### 注意

このコマンドは、デバッガへの HP DECwindows Motif for OpenVMS ユーザ・インタフェースでは使用できません。

---

---

## フォーマット

SHOW SELECT

---

## 説明

ディスプレイ属性には次の特性があります。

- エラー属性を持つディスプレイはデバッグ診断メッセージを表示する。
- 入力属性を持つディスプレイはデバッグ入力をエコーバックする。
- 命令属性を持つディスプレイはデバッグの対象になっているルーチンのデコードされた命令ストリームを表示する。EXAMINE/INSTRUCTION コマンドを入力すると、ディスプレイは更新される。
- 出力属性を持つディスプレイは他のディスプレイには出力されないデバッグ出力を表示する。
- プログラム属性を持つディスプレイはプログラムの入出力を表示する。現在、PROMPT ディスプレイだけがプログラム属性を持つことができる。
- プロンプト属性を持つディスプレイは、デバッガが入力を求める場所である。現在、PROMPT ディスプレイだけが PROMPT 属性を持つことができる。
- スクロール属性を持つディスプレイは、SCROLL、MOVE、EXPAND の各コマンドの省略時のディスプレイである。
- ソース属性を持つディスプレイはデバッグの対象になっているモジュールのソース・コードが表示できればそれを表示する。TYPE コマンドまたは EXAMINE /SOURCE コマンドを入力するとディスプレイは更新される。

### 関連コマンド

SELECT  
SHOW DISPLAY

---

### 例

```
DBG> SHOW SELECT
display selections:
  scroll = SRC
  input  = none
  output = OUT
  error  = PROMPT
  source = SRC
  instruction = none
  program = PROMPT
  prompt = PROMPT
DBG>
```

SHOW SELECT コマンドはディスプレイ属性のそれぞれに対して現在選択されているディスプレイを示します。選択されているディスプレイは言語の省略時の設定です。

---

## SHOW SOURCE

現在有効になっている，ソース・ディレクトリの検索リストと検索方法を示します。

---

### フォーマット

SHOW SOURCE

---

### 修飾子

/DISPLAY

デバッガがソース・コードを表示するときに使用する検索リストを指定します。

/EDIT

デバッガが EDIT コマンドの実行時に使用する検索リストを示します。

---

### 説明

SET SOURCE/MODULE=*module-name* コマンドは特定のモジュールに対するソース・ディレクトリ検索リストを設定します。SET SOURCE コマンドは SET SOURCE/MODULE=*module-name* コマンドで明示的に指定されていないすべてのモジュールに対してソース・ディレクトリ検索リストを設定します。これらのコマンドを使用していた場合，SHOW SOURCE は各検索カテゴリに対応するソース・ディレクトリ検索リストを示します。

ソース・ディレクトリ検索リストが SET SOURCE コマンドまたは SET SOURCE /MODULE=*module-name* コマンドによって設定されていない場合には，SHOW SOURCE コマンドはディレクトリ検索リストが現在有効になっていないことを通知します。この場合，各ソース・ファイルはそれがコンパイル時に存在していたディレクトリにあるものと判断されます (デバッガはさらにソース・ファイルのバージョン番号と作成日時がデバッガのシンボル・テーブルの情報に一致するかもチェックします)。

ソース・コードの表示用に使用するファイルが EDIT コマンドを使用して編集するファイルと異なっている場合には，/EDIT 修飾子を指定する必要があります。Ada プログラムの場合がそうです。Ada プログラムの場合，SHOW SOURCE コマンドはソース表示に使用するファイル (Ada プログラム・ライブラリの"コピーされたソース・ファイル"の検索リストを示します。SHOW SOURCE/EDIT コマンドは，EDIT コマンドを使用して編集されるソース・ファイルの検索リストを示します。

Ada プログラムに特有の情報については , Help Language\_Support Ada とタイプして参照してください。

関連コマンド

(SET,CANCEL) SOURCE

---

## 例

```
1. DBG> SHOW SOURCE
no directory search list in effect,
  match the latest source file version
DBG> SET SOURCE [PROJA],[PROJB],DISK:[PETER.PROJC]
DBG> SHOW SOURCE
source directory search list for all modules,
  match the latest source file version:
    [PROJA]
    [PROJB]
    DISK:[PETER.PROJC]
DBG>
```

この例では , SET SOURCE コマンドはデバッガに[PROJA] , [PROJB] , DISK:[PETER.PROJC]の各ディレクトリを検索するように指示します。省略時の設定では , デバッガはソース・ファイルの最新バージョンを検索します。

```
2. DBG> SET SOURCE/MODULE=CTEST/EXACT [], DISK$2:[PROJD]
DBG> SHOW SOURCE
source directory search list for CTEST,
  match the exact source file version:
    []
    DISK$2:[PROJD]
source directory search list for all other modules,
match the latest source file version:
    [PROJA]
    [PROJB]
    DISK:[PETER.PROJC]
DBG>
```

この例では , SET SOURCE コマンドはデバッガに現在の省略時のディレクトリ ( []) と DISK\$2:[PROJD]ディレクトリを検索して , CTEST モジュールで使用するソース・ファイルを検索するように指示します。/EXACT 修飾子は , デバッグ・シンボル・テーブルにあるバージョンと正確に一致するバージョンの CTEST ソース・ファイルを検索するように指定しています。

---

# SHOW STACK

現在アクティブなルーチン呼び出しに関する情報を表示します。

---

## フォーマット

SHOW STACK *[integer]*

---

## パラメータ

*integer*

表示するフレームの数を指定します。このパラメータを省略すると、デバッガはすべての呼び出しフレームに関する情報を表示します。

---

## 説明

各呼び出しフレームについて、SHOW STACK コマンドは、スタック・ポインタ、条件ハンドラ、保存されたレジスタ値 (Alpha および VAX)、ローカル・レジスタ割り当て (I64)、および存在する場合は引数リスト (VAX) などの情報を表示します。レジスタまたは引数リストで渡された引数には、実際の引数のアドレスが入っていることもあります。そのような場合、それらの引数の値を表示するには、EXAMINE *address-expression* コマンドを使用します。

VAX プロセッサでは、1 回のルーチン呼び出しでメモリ・スタック上に呼び出しフレームが 1 つ作成されます。

Alpha プロセッサと I64 プロセッサでは、ルーチン呼び出しは、次のいずれかとして扱われます。

- スタック・フレーム・プロシージャ。メモリ・スタック上に呼び出しフレームが作成されます。
- レジスタ・フレーム・プロシージャ。レジスタ・セット (Alpha) またはレジスタ・スタック (I64) に呼び出しフレームが作成されます。
- 空フレーム・プロシージャ。呼び出しフレームは作成されません。

SHOW STACK コマンドは、3 つのプロシージャ (スタック・フレーム、レジスタ・フレーム、空フレーム) すべての情報を出力します。(下記の例を参照してください。)

## 関連コマンド

## SHOW CALLS

---

例

VAX の例を示します。

```
DBG> SHOW STACK
stack frame 0 (2146814812)
    condition handler: 0
    SPA:              0
    S:                0
    mask:             ^M<r2>
    PSW:              0000 (hexadecimal)
    saved AP:         7
    saved FP:         2146814852
    saved PC:         EIGHTQUEENS\%LINE 69
    saved R2:         0
    argument list:(1) EIGHTQUEENS\%LINE 68+2

stack frame 1 (2146814852)
    condition handler: SHARE$PASRTL+888
    SPA:              0
    S:                0
    mask:             none saved
    PSW:              0000 (hexadecimal)
    saved AP:         2146814924
    saved FP:         2146814904
    saved PC:         SHARE$DEBUG+667
DBG>
```

上記の例では、SHOW STACK コマンドは、現在の PC および FP の位置にあるすべての呼び出しスタック・フレームに関する情報を表示します。

Alpha の例を示します。

```
DBG> SHOW STACK
invocation block 0
```

```

FP: 000000007F907AD0
Detected what appears to be a NULL frame
NULL frames operate in the same invocation context as their caller
NULL Procedure Descriptor (0000000000010050):
  Flags:                3089
  KIND:                 PDSC$K_KIND_FP_STACK (09)
  Signature Offset      0000
  Entry Address:        MAIN\FFFF
Procedure Descriptor (0000000000010000):
  Flags:                3089
  KIND:                 PDSC$K_KIND_FP_STACK (09)
  FP is Base Register
  Rsa Offset:           0008
  Signature Offset      0000
  Entry Address:        MAIN
  Ireg Mask:            20000004 <R2,FP>
    RA Saved @ 000000007F907AD8: FFFFFFFF8255A1F8
    R2 Saved @ 000000007F907AE0: 000000007FFBF880
    FP Saved @ 000000007F907AE8: 000000007F907B30
  Freg Mask:            00000000
  Size:                 00000020

```

invocation block 1

```

FP: 000000007F907B30
Procedure Descriptor (FFFFFFFF8255D910):
  Flags:                3099
  KIND:                 PDSC$K_KIND_FP_STACK (09)
  Handler Valid
  FP is Base Register
  Rsa Offset:           0048
  Signature Offset      0001
  Entry Address:        -2108317536
  Ireg Mask:            20002084 <R2,R7,R13,FP>
    RA Saved @ 000000007F907B78: 000000007FA28160
    R2 Saved @ 000000007F907B80: 0000000000000000
    R7 Saved @ 000000007F907B88: 000000007FF9C9E0
    R13 Saved @ 000000007F907B90: 000000007FA00900
    FP Saved @ 000000007F907B98: 000000007F907BB0
  Freg Mask:            00000000
  Size:                 00000070
  Condition Handler:    -2108303104

```

DBG>

上記の例では、ルーチンのプロローグおよびエピローグが空フレームとしてデバッガに表示されることに注意してください。フレーム・ポインタ (FP) が変更される前のプロローグの部分およびフレーム・ポインタ (FP) が回復された後のエピローグの部分は、それぞれ空フレームのように見えるため、空フレームと報告されます。

164 の例を示します。例の中では、次の略語が使用されています。

GP— グローバル・データ・セグメント・ポインタ (%R1)  
 PC— プログラム・カウンタ (命令ポインタ + 命令スロット番号)

## SHOW STACK

SP— スタック・ポインタ(メモリ・スタック)

BSP— バッキング・ストア・ポインタ(レジスタ・スタック)

CFM— 現在のフレーム・マーカ

```
DBG> SHOW STACK
Invocation block 0      Invocation handle 000007FDC0000270

GP:      0000000000240000
PC:      MAIN\FFFF
          In prologue region
RETURN PC: MAIN\%LINE 15
SP:      000000007AD13B40
Is memory stack frame:
          previous SP: 000000007AD13B40
BSP:      000007FDC0000270
Is register stack frame:
          previous BSP: 000007FDC0000248
CFM:      0000000000000005
          No locals      Outs R32 : R36

Invocation block 1      Invocation handle 000007FDC0000248

GP:      0000000000240000
PC:      MAIN\%LINE 15
RETURN PC: 0FFFFFFFF80C2A200
SP:      000000007AD13B40
Is memory stack frame:
          previous SP: 000000007AD13B70
BSP:      000007FDC0000248
Is register stack frame:
          previous BSP: 000007FDC0000180
CFM:      000000000000028A
          Ins/Locals R32 : R36      Outs R37 : R41

Invocation block 2      Invocation handle 000007FDC0000180

GP:      0FFFFFFFFF844DEC00
PC:      0FFFFFFFFF80C2A200
RETURN PC: SHARE$DCL_CODE0+5AB9F
SP:      000000007AD13B70
Is memory stack frame:
          previous SP: 000000007AD13BC0
BSP:      000007FDC0000180
Is register stack frame:
          previous BSP: 000007FDC00000B8
Has handler:
          function value: 0FFFFFFFFF842DFBD0
CFM:      0000000000000C20
          Ins/Locals R32 : R55      Outs R56 : R63

DBG>
```

詳細は、『OpenVMS Calling Standard』を参照してください。



---

## SHOW STEP

STEP コマンドに対して現在有効になっている省略時の修飾子 (/INTO , /INSTRUCTION , /NOSILENT など) を示します。

---

### フォーマット

SHOW STEP

---

### 説明

STEP コマンドの省略時の修飾子は前回 SET STEP コマンドで設定された省略時の修飾子です。SET STEP コマンドを入力していなかった場合、省略時の修飾子として /LINE , /OVER , /NOSILENT , /SOURCE が使用されます。

PF1-PF3 を押すことにより画面モードを有効にすると、(出力表示と DO 表示の冗長ソース表示を削除するために) SET MODE SCREEN コマンドだけでなく SET STEP NOSOURCE コマンドも入力されます。この場合、省略時の修飾子として /LINE , /OVER , /NOSILENT , /NOSOURCE が使用されます。

関連コマンド

STEP  
SET STEP

---

### 例

```
DBG> SET STEP INTO,NOSYSTEM,NOSHARE,INSTRUCTION,NOSOURCE
DBG> SHOW STEP
step type: nosystem, noshare, nosource, nosilent, into routine calls,
          by instruction
DBG>
```

この例では、SHOW STEP コマンドはデバッガが次の処置を取ることを示します。

- 呼び出し先ルーチン内の命令はステップ実行するが、システム空間内や共用可能イメージ内の命令はステップ実行しない。
- 命令単位でステップ実行する。
- ステップ実行中はソース・コードの行を表示しない。

---

## SHOW SYMBOL

現在のイメージに対してデバッガの実行時シンボル・テーブル (RST) に関する情報を表示します。

---

### 注意

現在のイメージはメイン・イメージ (省略時の設定) または前回 SET IMAGE コマンドで現在のイメージとして設定されたイメージのどちらかになります。

---

---

## フォーマット

```
SHOW SYMBOL symbol-name[, ... ] [/IN scope[, ... ]]
```

---

## パラメータ

*symbol-name*

表示するシンボルを指定します。有効なシンボル名は単一の識別子または %LABEL*n* という形式のラベル名です。ただし、*n* は整数です。RECORD.FIELD や ARRAY[1,2] などの複合名は有効ではありません。ワイルドカード文字のアスタリスク (\*) をそれだけで指定すると、すべてのシンボルがリストされます。シンボル名の中にはワイルドカードを使用できます。

*scope*

モジュール、ルーチンまたはレキシカル・ブロックの名前、あるいは数値有効範囲を指定します。このパラメータは SET SCOPE コマンドの有効範囲指定と同じ構文を持ち、パス名修飾子を含むことができます。指定する有効範囲はすべて現在のイメージ内の設定されたモジュールになければなりません。

SHOW SYMBOL コマンドは指定された名前に一致し、しかも *scope* パラメータで指定されたレキシカル要素の中で宣言されている現在のイメージの RST にあるシンボルだけを表示します。このパラメータを省略すると、*symbol-name* パラメータで指定された名前に一致するシンボルを検索するために、現在のイメージに対して設定されているすべてのモジュールとグローバル・シンボル・テーブル (GST) が検索されます。

---

修飾子**/ADDRESS**

選択されたそれぞれのシンボルに対するアドレス指定を表示します。アドレス指定はシンボルのアドレスを求める方法です。アドレス指定は単にシンボルのメモリ・アドレスにすることもできますが、間接参照やレジスタ値からのオフセットをいれることもできます。シンボルの中には、複雑すぎて、わかりやすく表示できないアドレス指定を持つものもあります。このようなアドレス指定には"complex address specifications"というラベルが付きます。

Alpha プロセッサでは、SHOW SYMBOL/ADDRESS procedure-name コマンドは指定されたルーチン、エントリ・ポイントまたは Ada パッケージのコード・アドレスとプロシージャ・ディスクリプタ・アドレスの両方を表示します。

**/DEFINED**

DEFINE コマンドで定義したシンボル (DEFINE シンボル・テーブルにあるシンボル定義) を表示します。

**/DIRECT**

*scope* パラメータで直接宣言されるシンボルだけを表示します。 *scope* パラメータで指定された有効範囲内でネストされたレキシカル要素で宣言されたシンボルは示されません。

**/FULL**

/ADDRESS, /TYPE, /USE\_CLAUSE 修飾子に関連するすべての情報を表示します。

C++ モジュールの場合、symbol-name がクラスの場合は、SHOW SYMBOL/FULL はそのクラスに関する情報も表示します。

**/LOCAL**

*scope* パラメータで直接宣言されるシンボルだけを表示します。 *scope* パラメータで指定された有効範囲内でネストされたレキシカル要素で宣言されたシンボルは示されません。

**/TYPE**

選択された各シンボルについてデータ型情報を表示します。

**/USE\_CLAUSE**

(Ada プログラムに適用される。) 指定のブロック、サブプログラムまたはパッケージが USE 句で名前を付ける Ada パッケージを示します。指定されたシンボルがパッケージである場合には、指定したシンボルに USE 句で名前を付けるブロック、サブプログラム、パッケージなども示します。

## 説明

SHOW SYMBOL コマンドはデバッガが現在のイメージ内の指定のシンボルに関して持っている情報を表示します。この情報はコンパイラが持っていた情報や、ソース・コードで確認できる情報と同じことがあります。それでも、このコマンドは、シンボルを処理するときにデバッガが取る動作の理由を理解するために役立ちます。

修飾子を指定しないと、SHOW SYMBOL コマンドは現在のイメージに対する RST に存在する (つまり、現在のイメージに対して設定されているすべてのモジュールと GST にある)、指定されたシンボルの宣言または定義のすべてをリストします。シンボルはそのパス名といっしょに表示されます。パス名はデバッガがシンボルの特定の宣言に達するためにたどらなければならない検索有効範囲 (モジュール、ネストされたルーチン、ブロックなど) を識別します。デバッガ・コマンドでシンボリック・アドレス式を指定するときは、シンボルが 2 回以上定義されていて、デバッガがあいまいさを解消できない場合にだけパス名を使用します。

/DEFINED 修飾子と/LOCAL 修飾子は (プログラムから派生するシンボルではなく) DEFINE コマンドで定義されたシンボルに関する情報を表示します。他の修飾子はプログラム内で宣言されたシンボルに関する情報を表示します。

Ada プログラムに特有の情報については、「Ada」ヘルプ・トピックを参照してください。

### 関連コマンド

```
DEFINE
DELETE
SET MODE [NO]LINE
SET MODE [NO]SYMBOLIC
SHOW DEFINE
SYMBOLIZE
```

---

## 例

1. DBG> SHOW SYMBOL I  
data FORARRAY\I  
DBG>

このコマンドはシンボル I が FORARRAY モジュールに定義されていて、ルーチンではなく変数 (データ) であることを示します。

```
2. DBG> SHOW SYMBOL/ADDRESS INTARRAY1
data FORARRAY\INTARRAY1
    descriptor address: 0009DE8B
DBG>
```

このコマンドはシンボル INTARRAY1 が FORARRAY モジュールに定義されていて、メモリ・アドレスが 0009DE8Bであることを示します。

```
3. DBG> SHOW SYMBOL *PL*
```

このコマンドは名前に"PL"という文字列が含まれているシンボルをすべてリストします。

```
4. DBG> SHOW SYMBOL/TYPE COLOR
data SCALARS\MAIN\COLOR
    enumeration type (primary, 3 elements), size: 4 bytes
```

このコマンドは変数 COLOR が列挙型であることを示します。

```
5. DBG> SHOW SYMBOL/TYPE/ADDRESS *
```

このコマンドはすべてのシンボルに関するすべての情報を表示します。

```
6. DBG> SHOW SYMBOL * IN MOD3\COUNTER
routine MOD3\COUNTER
data MOD3\COUNTER\X
data MOD3\COUNTER\Y
DBG>
```

このコマンドはパス名 MOD3\COUNTER で示される有効範囲で定義されているすべてのシンボルをリストします。

```
7. DBG> DEFINE/COMMAND SB=SET BREAK
DBG> SHOW SYMBOL/DEFINED SB
defined SB
    bound to: SET BREAK
    was defined /command
DBG>
```

この例では、DEFINE/COMMAND コマンドは SET BREAK コマンドのシンボルとして SB を定義します。SHOW SYMBOL/DEFINED コマンドはその定義を表示します。

---

## SHOW TASK|THREAD

マルチスレッド・プログラム (タスキング・プログラムとも呼ばれる) のタスクに関する情報を表示します。

---

### フォーマット

```
SHOW THREAD [task-spec[, ... ]]
```

---

### パラメータ

task-spec

タスク値を指定します。次のいずれかの形式で指定します。

- イベント機能が THREADS の場合
  - プログラムで宣言されているタスク (スレッド) 名, または結果がタスク ID 番号となる言語式。
  - 2 などのタスク ID 番号 (SHOW THREAD コマンドで表示される)
- イベント機能が ADA の場合
  - プログラムで宣言されているタスク (スレッド) 名, または結果がタスク値となる言語式。パス名も使用できます。
  - 2 などのタスク ID (SHOW THREAD コマンドで表示される)
- 次のタスク組み込みシンボルのいずれか

%ACTIVE_TASK	GO, STEP, CALL または EXIT コマンドの実行時に実行されるタスク。
%CALLER_TASK	Ada プログラムにのみ適用される。accept 文の実行時に, その accept 文に対応するエントリを呼び出したタスク。
%NEXT_TASK	デバッガのタスク・リスト中の可視タスクの次のタスク。タスクの順番は自由に決められますが, その順番は 1 回のプログラム実行の中で不変です。
%PREVIOUS_TASK	デバッガのタスク・リスト中の可視タスクの前のタスク。
%VISIBLE_TASK	シンボル, レジスタ値, ルーチン呼び出し, ブレークポイントなどの検索時に現在のコンテキストになっている呼び出しスタックとレジスタ・セットを持つタスク。

ワイルドカード文字のアスタリスク(\*)は使用できません。代わりに/ALL 修飾子を使用してください。/ALL, /STATISTICS または/TIME\_SLICE を指定する場合, タスクは指定できません。

---

## 修飾子

/ALL

既存のすべてのタスク，つまり，すでに作成されているタスクで，(Ada タスクの場合には) マスタがまだ終了していないタスクを表示の対象として選択します。

/CALLS[=*n*]

表示の対象として選択されたタスクごとに SHOW CALLS コマンドを実行します。これはタスクの現在アクティブなルーチン呼び出し (呼び出しスタック) を示します。

/FULL

イベント機能が THREADS の場合は，コマンドを使用します。

表示の対象として選択された各タスクについて補足的な情報を表示します。他の修飾子を指定しないで/FULL を指定した場合，あるいは/CALLS または/STATISTICS を指定した場合には，補足的な情報が表示されます。

/HOLD

/NOHOLD (省略時の設定)

イベント機能が THREADS の場合は，PTHREAD tset -n thread-number コマンドを使用します。

表示の対象として凍結状態になっているタスクが凍結状態になっていないタスクのいずれかを選択します。

タスクを指定しない場合，/HOLD は凍結状態のすべてのタスクを選択します。タスク・リストを指定すると，/HOLD はタスク・リストに載っている凍結状態のタスクを選択します。

タスクを指定しない場合，/NOHOLD は凍結状態になっていないすべてのタスクを選択します。タスク・リストを指定すると，/NOHOLD はタスク・リストに載っている凍結状態になっていないタスクを選択します。

/IMAGE

呼び出しスタック上の個々のアクティブな呼び出しのイメージ名を表示します。

/CALLS 修飾子と組み合わせたときにのみ有効です。

/PRIORITY=(*n*[, . . . ])

イベント機能が THREADS の場合は，PTHREAD tset -n thread-number コマンドを使用します。

タスクを指定しない場合には，指定した優先順位 *n* のどれかを持つすべてのタスクを選択します。ただし，*n* は 0 ~ 15 の 10 進整数です。タスク・リストを指定すると，指定された優先順位のどれかを持つ，タスク・リストに載っているタスクを選択します。

/STATE=(state[, . . . ])

タスクを指定しない場合、指定された状態、つまり、RUNNING、READY、SUSPENDED または TERMINATED のいずれかになっているタスクすべてを選択します。タスク・リストを指定すると、指定された状態のいずれかになっている、タスク・リスト内のタスクを選択します。

/STATISTICS

(Compaq Ada on VAX のみ) タスキング・システム全体のタスク統計情報を表示します。この情報を使用すると、タスキング・プログラムの性能を測定できます。全体のスケジューリング (コンテキスト・スイッチとも言う) の数が多ければ多いほど、タスキングのオーバーヘッドが増えます。

/TIME\_SLICE

(VAX のみ) 前回の SET TASK/TIME\_SLICE コマンドによる指定に従って、現在のタイム・スライス値を秒単位で表示します。SET TASK/TIME\_SLICE コマンドがそれまでに入力されていなかった場合には、プログラムで指定されているタイム・スライス値があればそれを表示します。それまでにタイム・スライス値が設定されていない場合には、値は 0.0、つまりタイム・スライスは無効ということになります。

/TIME\_SLICE はイベント機能が ADA のときのみ有効になります。

---

## 説明

タスクは作成されるとすぐに、まず SHOW THREAD コマンドで表示することができます。タスクは終了するか、または (Ada タスキング・プログラムの場合) そのマスタが終了すると、SHOW THREAD コマンドを実行しても表示されなくなります。省略時の設定では、SHOW THREAD コマンドは選択されたタスク 1 つにつき 1 行の情報を表示します。

/IMAGE 修飾子を指定すると、デバッガはまずデバッグ情報を持っている各イメージ (/DEBUG または /TRACEBACK 修飾子を使ってリンクされたもの) に対して SET IMAGE コマンドを実行します。その後、デバッガは呼び出しスタック上の個々のアクティブな呼び出しについて、イメージ名を表示します。出力ディスプレイが拡大され、イメージ名が最初の欄に表示されるようになっています。

share\$image\_name モジュール名は/IMAGE 修飾子によって提供されるので、デバッガはこの情報は表示しません。

SET IMAGE コマンドの実行結果は、SHOW THREAD/CALLS/IMAGE コマンドを実行している間のみ有効です。デバッガは SHOW THREAD/CALLS/IMAGE コマンドが完了したときに、セット・イメージ状態を復元します。



## 関連コマンド

DEPOSIT/TASK  
 EXAMINE/TASK  
 (SET, SHOW) EVENT\_FACILITY  
 SET TASK | THREAD

## 例

1. DBG> SHOW EVENT\_FACILITY  
 event facility is ADA

...

DBG> SHOW TASK/ALL

task id	pri	hold	state	substate	task object
* %TASK 1	7		RUN		122624
%TASK 2	7	HOLD	SUSP	Accept	H4.MONITOR
%TASK 3	6		READY	Entry call	H4.CHECK_IN

DBG>

この例では、SHOW EVENT\_FACILITY コマンドは現在のイベント機能として ADA を表示しています。SHOW TASK/ALL コマンドは、Ada サービスを使用して作成され、現在も存在しているすべてのタスクに関する基本的な情報を表示します。タスクごとに 1 行が使用されます。アクティブなタスクにはアスタリスク(\*)が付けられます。この例では、%TASK 1 がアクティブなタスク (RUN 状態にあるタスク) です。

2. DBG> SHOW TASK %ACTIVE\_TASK,3,MONITOR

このコマンドはアクティブなタスク、%TASK 3 タスク、および MONITOR というタスクを表示の対象として選択します。

3. DBG> SHOW TASK/PRIORITY=6

このコマンドは優先順位が 6 になっているすべてのタスクを表示の対象として選択します。

4. DBG> SHOW TASK/STATE=(RUN,SUSP)

このコマンドは実行中であるか、中断されているすべてのタスクを表示の対象として選択します。

5. DBG> SHOW TASK/STATE=SUSP/NOHOLD

このコマンドは、中断されていて、しかも凍結状態でないすべてのタスクを表示の対象として選択します。

6. DBG> SHOW TASK/STATE=(RUN,SUSP)/PRIO=7 %VISIBLE\_TASK, 3

このコマンドは、可視タスクと%TASK 3 タスクの中から、RUNNING または SUSPENDED 状態になっていて、優先順位が 7 になっているタスクを表示の対象として選択します。

---

## SHOW TERMINAL

出力を編集するために使用される現在の端末画面の高さ (ページ) と幅を示します。

---

### 注意

---

このコマンドは、デバッガへの HP DECwindows Motif for OpenVMS ユーザ・インタフェースでは使用できません。

---

---

## フォーマット

SHOW TERMINAL

---

## 説明

現在の端末画面の高さと幅は前回 SET TERMINAL コマンドで設定された高さ と 幅です。省略時の設定では、SET TERMINAL コマンドを入力していなかった場合、現在の高さと幅はターミナル・ドライバに既知の高さと幅 (通常、VT シリーズ端末では 24 行、80 列) であり、DCL コマンドの SHOW TERMINAL が表示するのはこの高さと幅です。

### 関連コマンド

SET TERMINAL  
SHOW DISPLAY  
SHOW WINDOW

---

## 例

```
DBG> SHOW TERMINAL
terminal width: 80
           page: 24
           wrap: 80
DBG>
```

このコマンドは現在の端末画面の幅と高さ (ページ) を 80 列、24 行と表示し、メッセージは 80 列でラップするよう設定されています。

---

# SHOW TRACE

トレースポイントに関する情報を表示します。

---

## フォーマット

SHOW TRACE

---

## 修飾子

/PREDEFINED

定義済みトレースポイントに関する情報を表示します。

/USER

ユーザ定義トレースポイントに関する情報を表示します。

---

## 説明

SHOW TRACE コマンドは、WHEN 句または DO 句、/AFTER の数などのオプションも含めて、現在設定されているトレースポイントについての情報と、そのトレースポイントが無効になっているかどうかを表示します。

省略時の設定では、SHOW TRACE はユーザ定義トレースポイントと定義済みトレースポイントの両方(ただし、ある場合)に関する情報を表示します。これは SHOW TRACE/USER/PREDEFINED コマンドを入力する場合と同じです。ユーザ定義トレースポイントは SET TRACE コマンドで設定されます。定義済みトレースポイントは、デバグを起動すると自動的に設定され、デバグの対象になっているプログラムの種類によって異なります。

SET TRACE/AFTER:*n*を使用してトレースポイントを設定した場合、SHOW TRACE コマンドは 10 進整数*n*の現在の値、つまり最初に指定された整数値からトレースポイント記憶位置に到達するたびに 1 を引いた値を表示します。デバグは*n*の値が 0 になるまでトレースポイント記憶位置に到達するたびに*n*を減少させていきます。0 になると、デバグはトレース動作を取ります。

Alpha システムでは、トレースが特定のクラスの命令にある場合は、(SET TRACE /CALL または SET TRACE/RETURN と同様に) SHOW TRACE コマンドは個々の命令を表示しません。

## 関連コマンド

(ACTIVATE, DEACTIVATE, SET, CANCEL) TRACE

## 例

1. 

```
DBG> SHOW TRACE
    tracepoint at routine CALC\MULT
    tracepoint on calls:
           RET      RSB      BSBB      JSB      BSBW      CALLG      CALLS
DBG>
```

この VAX 例では、SHOW TRACE コマンドは現在設定されているすべてのトレースポイントを示します。この例は、実行が CALC モジュール内の MULT ルーチンか、または RET、RSB、BSBB、JSB、BSBW、CALLG または CALLS の命令のどれか 1 つに到達すると必ず検出されるユーザ定義トレースポイントを示します。

2. 

```
all> SHOW TRACE/PREDEFINED
    predefined tracepoint on program activation
    DO (SET DISP/DYN/REM/SIZE:64/PROC SRC_ AT H1 SOURCE
        (EXAM/SOURCE .%SOURCE SCOPE\%PC));
        SET DISP/DYN/REM/SIZE:64/PROC INST_ AT H1 INST
        (EXAM/INSTRUCTION .0\%PC))
    predefined tracepoint on program termination
all>
```

このコマンドは現在設定されている定義済みトレースポイントを示します。この例はマルチプロセス・プログラムの場合、デバッガによって自動的に設定される定義済みトレースポイントを示します。新しいプロセスがデバッガの制御下に入ると、プログラム起動のトレースポイントが起動されます。DO 句はプロセス起動トレースポイントが検出されると、SRC\_*n* という名前のプロセス固有のソース表示と INST\_*n* という名前のプロセス固有の機械語命令ディスプレイを作成します。プロセスがイメージ終了を行うと、プログラム終了のトレースポイントが検出されます。

---

## SHOW TYPE

型がコンパイラ生成型ではないプログラム記憶位置の現在の型を示します。  
/OVERRIDE が指定された場合には、現在の上書き型を示します。

---

### フォーマット

SHOW TYPE

---

### 修飾子

/OVERRIDE  
現在の上書き型を示します。

---

### 説明

型がコンパイラ生成型ではないプログラム記憶位置の現在の型は前回 SET TYPE コマンドで設定された型です。SET TYPE コマンドを入力していなかった場合、そのような記憶位置の型はロングワード整数になります。

すべてのプログラム記憶位置に対する現在の上書き型は前回 SET TYPE/OVERRIDE コマンドで設定された上書き型です。SET TYPE/OVERRIDE コマンドを入力していなかった場合、上書き型はありません。

#### 関連コマンド

CANCEL TYPE/OVERRIDE  
DEPOSIT  
EXAMINE  
(SET,SHOW,CANCEL) MODE  
(SET,SHOW,CANCEL) RADIX  
SET TYPE

---

例

1. 

```
DBG> SET TYPE QUADWORD
DBG> SHOW TYPE
type: quadword integer
DBG>
```

この例では、型がコンパイラ生成型ではない記憶位置に対して型をクォドワードに設定します。SHOW TYPE コマンドはそれらの記憶位置の現在の省略時の型をクォドワード整数として表示します。これは、特に他に指定がないかぎり (たとえば EXAMINE コマンドで type 修飾子を使用するなどによる)、デバッガがそれらの記憶位置にある要素をクォドワード整数として解釈し、表示することを意味します。

2. 

```
DBG> SHOW TYPE/OVERRIDE
type/override: none
DBG>
```

このコマンドは上書き型が定義されていないことを表します。

---

# SHOW WATCH

ウォッチポイントに関する情報を表示します

---

## フォーマット

SHOW WATCH

---

## 説明

SHOW WATCH コマンドは、WHEN 句または DO 句、/AFTER の数などのオプションも含めて現在設定されているウォッチポイントに関する情報と、そのウォッチポイントが無効になっているかどうかを表示します。

SET WATCH/AFTER:*n*を使用してウォッチポイントを設定した場合、SHOW WATCH コマンドは 10 進整数 *n* の現在の値、つまり最初に指定された整数値からウォッチポイント記憶位置に到達するたびに 1 を引いた値を表示します。デバッガは *n* の値が 0 になるまでウォッチポイント記憶位置に到達するたびに *n* を減少させていきます。0 になると、デバッガはウォッチ動作を取ります。

関連コマンド

(ACTIVATE,CANCEL,DEACTIVATE,SET) WATCH

---

## 例

```
DBG> SHOW WATCH
watchpoint of MAIN\X
watchpoint of SUB2\TABLE+20
DBG>
```

このコマンドは 2 つのウォッチポイントを表示します。1 つは (MAIN モジュールに定義されている) 変数 X にあるもので、もう 1 つは SUB2\TABLE+20 の記憶位置 (アドレス式 TABLE によって示されるアドレスに 20 バイト加えた位置) にあるものです。

---

# SHOW WINDOW

定義済みとユーザ定義の画面モード・ウィンドウの名前と画面位置を示します。

---

## 注意

このコマンドは、デバッガへの HP DECwindows Motif for OpenVMS ユーザ・インタフェースでは使用できません。

---

---

## フォーマット

SHOW WINDOW *[window-name[, . . . ]]*

---

## パラメータ

windowname

画面ウィンドウ定義の名前を指定します。名前を指定しない場合またはワイルドカード文字のアスタリスク(\*)を単独で指定する場合、すべてのウィンドウ定義がリストされます。ウィンドウ名の中ではワイルドカードを使用できます。/ALL 修飾子を指定する場合、ウィンドウ定義名は指定できません。

---

## 修飾子

/ALL

すべてのウィンドウ定義をリストします。

---

## 説明

関連コマンド

(SHOW,CANCEL) DISPLAY  
(SET,SHOW) TERMINAL  
(SET,CANCEL) WINDOW  
SHOW SELECT



---

**例**

```
DBG> SHOW WINDOW LH*,RH*  
window LH1 at (1,11,1,40)  
window LH12 at (1,23,1,40)  
window LH2 at (13,11,1,40)  
window RH1 at (1,11,42,39)  
window RH12 at (1,23,42,39)  
window RH2 at (13,11,42,39)  
DBG>
```

このコマンドは LH または RH で始まる名前を持つすべての画面ウィンドウ定義の名前と画面部を表示します。

---

## SPAWN

サブプロセスを作成し、ユーザがデバッグ・セッションを終了したりデバッグ・コンテキストを失ったりせずに、DCL コマンドを実行できるようにします。

---

### 注意

このコマンドは、デバッガへの HP DECwindows Motif for OpenVMS ユーザ・インタフェースでは使用できません。

---

---

## フォーマット

SPAWN *[DCL-command]*

---

## パラメータ

### DCL-command

サブプロセスで実行する DCL コマンドを指定します。DCL コマンドが終了すると、制御はデバッグ・セッションに戻ります。

DCL コマンドを指定しないと、サブプロセスが作成され、DCL コマンドを入力できるようになります。作成されたプロセスからログ・アウトするか、(DCL コマンドの ATTACH を使用して) 親プロセスに接続すると、デバッグ・セッションを継続できるようになります。

DCL コマンドにセミコロンが含まれている場合には、そのコマンドを二重引用符(")で囲まなければなりません。セミコロンがない場合には、セミコロンはデバッガ・コマンドの区切り文字と解釈されます。文字列に二重引用符をいれるには、二重引用符を 2 つ続けて入力します ("")。

---

## 修飾子

### /INPUT=file-spec

作成されたサブプロセスで実行される 1 つまたは複数の DCL コマンドを含む入力 DCL コマンド・プロシージャを指定します。省略時のファイル型は.COM です。SPAWN コマンドで DCL コマンド文字列を指定し、/INPUT で入力ファイルを指定すると、入力ファイルの前にコマンド文字列が処理されます。入力ファイルの処理が完了したあと、サブプロセスが終了します。ファイル指定にはワイルドカード文字のアスタリスク(\*)は使用できません。

/OUTPUT=file-spec

指定されたファイルに SPAWN 操作からの出力を書き込みます。省略時のファイル型は LOG です。ファイル指定にはワイルドカード文字のアスタリスク(\*)は使用できません。

/WAIT (省略時の設定)

/NOWAIT

サブプロセスの実行中にデバッグ・セッション(親プロセス)を中断するかどうかを制御します。/WAIT 修飾子(省略時の設定)を指定すると、サブプロセスが終了するまでデバッグ・セッションは中断されます。制御が親プロセスに戻るまでデバッガ・コマンドは入力できません。

/NOWAIT 修飾子を指定すると、サブプロセスがデバッグ・セッションと並行して実行されます。サブプロセスの実行中でもデバッガ・コマンドは入力できます。/NOWAIT を使用する場合には、SPAWN コマンドといっしょに DCL コマンドも指定してください。これで、DCL コマンドがサブプロセスで実行されるようになります。作成されたサブプロセスの完了を通知するメッセージが表示されます。

保持デバッガ(つまり、DEBUG/KEEP コマンドで起動されたデバッガ)は、SPAWN/NOWAIT コマンドによって実行される場合に、親プロセスと I/O チャンネルを共用します。このため、HP DECwindows Motif for OpenVMS ユーザ・インタフェースでは、DECterm 上でリターン・キーを 2 回押さなければなりません。デバッガは、デバッガ・バージョン番号がコマンド・ビューに表示された後に、DECterm から実行されます。

オプションでは、次の方法で保持デバッガを実行できます。

```
$ DEFINE DBG$INPUT NL:
$ SPAWN/NOWAIT RUN DEBUG/KEEP
```

---

## 説明

SPAWN コマンドは DCL コマンドの SPAWN と全く同じように働きます。デバッグ・セッションを終了したり現在のデバッグ・コンテキストを失ったりせずにファイルの編集、プログラムのコンパイル、メールの読み込みなどを行えます。

さらに、DCL コマンドの SPAWN を実行することもできます。DCL は 2 番目の SPAWN コマンドを、そのコマンドで指定された修飾子も含めて処理します。

## 関連コマンド

ATTACH

---

例

1. DBG> SPAWN  
\$

この例は、パラメータを指定されていない SPAWN コマンドが DCL レベルでサブプロセスを作成することを示します。これで DCL コマンドを入力することができます。デバッガのプロンプトに戻るにはログ・アウトします。

2. DBG> SPAWN/NOWAIT/INPUT=READ\_NOTES/OUTPUT=0428NOTES

このコマンドはデバッグ・セッションと並行して実行されるサブプロセスを作成します。このサブプロセスは DCL コマンド・プロシージャの READ\_NOTES.COM を実行します。作成された操作からの出力は 0428NOTES.LOG ファイルに書き込まれます。

3. DBG> SPAWN/NOWAIT SPAWN/OUT=MYCOM.LOG @MYCOM

このコマンドはデバッグ・セッションと並行して実行されるサブプロセスを作成します。このサブプロセスは DCL コマンド・プロシージャ MYCOML.COM を実行するためにもう 1 つ別のサブプロセスを作成します。その操作からの出力は MYCOM.LOG ファイルに書き込まれます。

---

## START HEAP\_ANALYZER (I64 のみ)

ヒープ・アナライザを起動して、ヒープ・メモリの問題を診断します。

---

### 注意

ヒープ・アナライザは、DECwindows ディスプレイを必要とします。

---

---

## フォーマット

START HEAP\_ANALYZER *[integer]*

---

## 説明

デバッグ対象イメージのメモリ使用状況をグラフィカルに表示する、ヒープ・アナライザを起動します。ヒープ・アナライザのメイン・ウィンドウが表示されると、現在ロードされているイメージのデータがヒープ・アナライザに表示されます。デバッガのコマンド・プロンプト (DBG>) に戻るには、ヒープ・アナライザの「Start」ボタンを押します。

---

### 注意

START HEAP\_ANALYZER コマンドを入力する前に発生したヒープ・メモリ操作は、ヒープ・アナライザには記録されません。すべてのヒープ・メモリ操作を記録するには、監視対象のイメージが動作し始めてすぐにヒープ・アナライザを起動してください。

---

ヒープ・アナライザの使用についての詳細は、『デバッガ説明書』を参照してください。

### 関連コマンド

RUN  
RERUN

---

## 例

1. DBG> START HEAP\_ANALYZER

ヒープ・アナライザを起動してデバッグ対象プログラムのメモリ使用状況をグラフィカルに表示します。

---

## STEP

次の行または命令，あるいは他に指定された記憶位置に達するまでプログラムを実行します。

---

### フォーマット

STEP *[integer]*

---

### パラメータ

*integer*

実行するステップ単位 (行，命令など) の数を指定する 10 進整数。このパラメータを省略すると，デバッガは 1 つのステップ単位を実行します。

---

### 修飾子

/BRANCH

次の分岐命令までプログラムを実行します。STEP/BRANCH は SET BREAK/TEMPORARY/BRANCH;GO と同じ働きをします。

/CALL

次の呼び出しまたは復帰命令までプログラムを実行します。STEP/CALL は SET BREAK/TEMPORARY/CALL;GO と同じ働きです。

/EXCEPTION

もしあれば，次の例外までプログラムを実行します。STEP/EXCEPTION は SET BREAK/TEMPORARY/EXCEPTION;GO と同じ働きです。例外が発生しないと，STEP/EXCEPTION は GO と同じ働きになります。

/INSTRUCTION

/INSTRUCTION=(opcode[, . . . ])

命令コードが指定されないと，次の命令までプログラムを実行します。STEP/INSTRUCTION は SET BREAK/TEMPORARY/INSTRUCTION;GO と同じ働きです。

VAX プロセッサでは，1 つ以上の opcode を指定できます。デバッガは，opcode がリストに指定されている次の命令まで，プログラムを実行します。次のコマンドの結果は同じになります。

```
DBG> STEP/INSTRUCTION=(opcode[, . . . ])
```

```
DBG> SET BREAK/TEMPORARY/INSTRUCTION=(opcode[, . . . ]);GO
```

## /INTO

ルーチン呼び出しで実行が現在中断されている場合、STEP/INTO はそのルーチンの先頭までプログラムを実行します (そのルーチン内の命令をステップ実行する)。そうでない場合には、STEP/INTO は修飾子が指定されていないSTEP と同じ働きます。/INTO 修飾子は/OVER (省略時の動作) とは逆になります。

---

### 注意

---

Alpha プロセッサで、例外ブレークで実行が停止された場合には、STEP /INTO は制御をユーザ例外ハンドラに渡しません。ハンドラにブレークポイントを設定することにより、ハンドラ内で実行を停止してください。

---

STEP/INTO の動作は、/[NO]JSB、/[NO]SHARE、/[NO]SYSTEM 修飾子を使用して変更することができます。

## /JSB /NOJSB

(VAX のみ) 前回の SET STEP INTO コマンドまたは現在の STEP /INTO コマンドを修飾します。

ルーチン呼び出しで実行が現在中断されていて、そのルーチンが JSB 命令で呼び出されている場合、STEP/INTO/NOJSB は STEP/OVER と同じ働きます。そうでない場合、STEP/INTO/NOJSB は STEP/INTO と同じ働きになります。

前回の SET STEP NOJSB コマンドを上書きするには STEP/INTO/JSB コマンドを使用します。STEP/INTO/JSB を指定すると、STEP/INTO は CALL 命令で呼び出されたルーチン内の命令だけでなく、JSB 命令で呼び出されたルーチン内の命令もステップ実行できるようになります。

/JSB 修飾子は DIBOL 以外のすべての言語の省略時の設定です。DIBOL の場合、/NOJSB 修飾子が省略時の設定です。DIBOL では、アプリケーション宣言ルーチンは CALL 命令で呼び出され、DIBOL 実行時ライブラリ・ルーチンは JSB 命令で呼び出されます。

## /LINE

ソース・コードの次の行までプログラムを実行します。ただし、デバッガはコンパイルの結果、実行可能なコードにならないソース行 (たとえば、コメント行) はスキップします。STEP/LINE は SET BREAK/TEMPORARY/LINE;GO と同じ働きます。これがすべての言語の省略時の設定です。

## /OVER

ルーチン呼び出しで実行が現在中断されている場合、STEP/OVER はそのルーチンの復帰命令まで、その命令も含めて実行します (そのルーチンを 1 ステップとして実行します)。/OVER 修飾子は省略時の動作であり、/INTO とは逆になります。

---

 注意
 

---

Alpha プロセッサでは、ルーチン呼び出しによってループを含むソース行で実行が中断されると、STEP/OVER は呼び出されたルーチンにステップします。次のプログラム文にステップするには、その文に一時的なブレイクポイントを設定し、GO と入力します。

---

## /RETURN

実行が現在中断されているルーチンをその復帰命令まで (つまり、制御を呼び出し元ルーチンに戻す直前のポイントに至るまで) 実行します。この結果、復帰命令がルーチンの呼び出しフレームを呼び出しスタックから削除する前にローカル環境をチェックできます (たとえば、ローカル変数の値を獲得できます)。STEP/RETURN は SET BREAK/TEMPORARY/RETURN;GO と同じ働きです。

STEP/RETURN *n* は呼び出しスタックの *n* レベルまでプログラムを実行します。

## /SEMANTIC\_EVENT

(Alpha のみ) プログラムを次のセマンティック・イベントに実行します。

STEP/SEMANTIC\_EVENT は、デバッグの最適化されたコードを簡素化します (説明の項を参照)。

## /SHARE (省略時の設定)

## /NOSHARE

前回の SET STEP INTO コマンドまたは現在の STEP/INTO コマンドを修飾します。

共用可能イメージ・ルーチンへの呼び出しで実行が現在中断されている場合、STEP/INTO/NOSHARE は STEP/OVER と同じ働きです。そうでない場合、STEP/INTO/NOSHARE は STEP/INTO と同じ働きになります。

前回の SET STEP NOSHARE コマンドを上書きするには、STEP/INTO/SHARE を使用します。STEP/INTO/SHARE を指定すると、STEP/INTO は他の種類のルーチン内だけでなく、共用可能イメージ・ルーチン内の命令もステップ実行できるようになります。

## /SILENT

## /NOSILENT (省略時の設定)

STEP の完了後、"stepped to ..." メッセージと現在の記憶位置のソース行を表示するかどうかを制御します。/NOSILENT 修飾子はメッセージを表示することを指定します。/SILENT 修飾子はメッセージとソース行を表示しないことを指定します。/SILENT 修飾子を指定すると、/SOURCE は上書きされます。

## /SOURCE (省略時の設定)

## /NOSOURCE

STEP の完了後、現在の記憶位置のソース行を表示するかどうかを制御します。

/SOURCE 修飾子はソース行を表示することを指定します。/NOSOURCE 修飾子はソ



ース行を表示しないことを指定します。/SILENT 修飾子を指定すると、/SOURCE は上書きされます。SET STEP [NO]SOURCE コマンドも参照してください。

/SYSTEM (省略時の設定)

/NOSYSTEM

前回の SET STEP INTO コマンドまたは現在の STEP/INTO コマンドを修飾します。

(P1 空間内の) システム・ルーチンへの呼び出しで実行が現在中断されている場合、STEP/INTO/NOSYSTEM は STEP/OVER と同じ働きです。そうでない場合、STEP/INTO/NOSYSTEM は STEP/INTO と同じ働きです。

前回の SET STEP NOSYSTEM コマンドを上書きするには、STEP/INTO/SYSTEM を使用します。STEP/INTO/SYSTEM を指定すると、STEP/INTO は他の種類のルーチン内だけでなく、システム・ルーチン内の命令もステップ実行できるようになります。

---

## 説明

STEP コマンドはプログラムを実行するために使用できる 4 つのデバッガ・コマンドの 1 つです (他の 3 つは CALL, EXIT, GO です)。

STEP コマンドの動作は次の要因に依存します。

- 前回の SET STEP コマンドで設定された省略時の STEP モード (ただし、設定されている場合)。
- STEP コマンドで指定された修飾子 (ただし、指定されている場合)。
- STEP コマンドのパラメータとして指定されたステップ単位の数 (ただし、指定されている場合)。

それまでに SET STEP コマンドを入力していない場合、修飾子もパラメータも指定せずに STEP コマンドを入力すると、デバッガは次の省略時の動作を行います。

1. ソース・コードを 1 行実行する (省略時の設定は STEP/LINE)。
2. "stepped to ..." メッセージを発行することにより実行が完了したことを通知する (省略時の設定は STEP/NOSILENT)。
3. 実行が中断されているソース・コードの行を表示する (省略時の設定は STEP/SOURCE)。
4. プロンプトを表示する。

次の修飾子は命令をステップ実行する記憶位置に影響します。

/BRANCH

/CALL

```

/EXCEPTION
/INSTRUCTION
/INSTRUCTION=(opcode[, ... ]) (VAX のみ)
/LINE
/RETURN
/SEMANTIC_EVENT (Alpha のみ)

```

次の修飾子は 1 つのステップの完了時にどのような出力になるかに影響します。

```

/[NO]SILENT
/[NO]SOURCE

```

次の修飾子はルーチン呼び出し時に何が起きるかに影響します。

```

/INTO
/[NO]JSB (VAX のみ)
/OVER
/[NO]SHARE
/[NO]SYSTEM

```

同じ修飾子を指定して STEP コマンドを複数入力する予定ならば、最初に SET STEP コマンドを使用して新しい省略時の修飾子を設定することができます (たとえば、SET STEP INTO, NOSYSTEM と指定すれば、STEP コマンドは STEP/INTO /NOSYSTEM と同じ働きをするようになります)。この結果、STEP コマンドを指定するときにこれらの修飾子を使用する必要がなくなります。他の修飾子を指定することにより、1 つの STEP コマンドが継続している間、現在の省略時の修飾子を上書きすることができます。現在の STEP コマンドの省略時の設定を示すには、SHOW STEP コマンドを使用します。

SET BREAK/EXCEPTION コマンドまたは STEP/EXCEPTION コマンドを実行した結果、例外ブレークポイントが検出されると、アプリケーション宣言条件ハンドラが起動される前に実行は中断されます。そのあと、STEP コマンドで実行を再開すると、デバグは例外を再びシグナル通知し、条件ハンドラがあればその先頭までプログラムは実行されます (条件ハンドラ内の命令がステップ実行されます)。

Alpha システムでは、プログラムが/OPTIMIZE を使用してコンパイルされた場合は、STEP/SEMANTIC\_EVENT および SET STEP SEMANTIC\_EVENT コマンドを使用してセマンティック・ステップ・モードを使用することができます。最適化されたコードをデバッグしている場合は、ソース・プログラムの明らかな記憶位置は、同じ行が繰り返し現れる状態で前後にバウンドする傾向があります。セマンティック・ステップ・モードでは、プログラムは、プログラム内の次のポイントまで実行します。次のポイントでは、顕著な効果 (セマンティック・イベント) が現れます。

セマンティック・イベントは次のうちの 1 つです。

- データ・イベント — ユーザへの割り当てが可能

- 制御イベント — 呼び出し以外の条件付きまたは条件なしの転送の制御で、制御フローの決定
- 呼び出しイベント — 呼び出し (ステップされないルーチン) または呼び出しからの戻り

すべての割り当て、転送制御、または呼び出しがセマンティック・イベントというわけではありません。主な例外は次のようになります。

- 2つの命令が、複雑な値またはX浮動の値に割り当てる必要がある場合は、最初の命令だけがセマンティック・イベントとして扱われます。
- あるケースまたは選択構造を実行する判断ツリーの分岐など、単一の上位レベル構造の一部である分岐が複数ある場合は、最初のイベントだけがセマンティック・イベントとして扱われます。
- 一種の文字列または記憶域のコピー操作を処理する OTSS\$MOVE への呼び出しなどの、コンパイラ指定のヘルプ・ルーチンであるルーチンに呼び出しが作成された場合は、呼び出しはセマンティック・イベントとはみなされません。呼び出しの際に、制御は終了しません。

ルーチンにステップするには、次のいずれかを行わなければなりません。

- ルーチン・エントリ・ポイントにブレークポイントを設定するか、または
- 一連の STEP/INSTRUCTION コマンドを使用して関連する呼び出しに到達した後に、STEP/INSTRUCTION/INTO を使用して呼び出されたルーチンに入ります。
- 同じ行番号で、1行につき1つ以上の潜在的なセマンティック・イベントがある場合は、最初のイベントだけがセマンティック・イベントとして扱われます。

STEP/SEMANTIC\_EVENT コマンドは、ブレークポイントが次のセマンティック・イベントに設定されるようにします。実行は、次のイベントに進みます。異なる行および文のすべての番号部分は、進行を妨げられずに、方法に従って実行される可能性があります。セマンティック・イベントが到達した場合 (つまり、イベントに対応する命令が到達しているが、実行されていない場合) 実行は中断されます (STEP/LINE を使用する場合は、次行への到達と似ています)。

デバッガの最適化されたプログラムについての詳細は、『OpenVMS デバッガ説明書』を参照してください。

マルチプロセス・プログラムをデバッグしているときには、STEP コマンドは現在のプロセス・セットのコンテキストで実行されます。さらに、マルチプロセス・プログラムをデバッグしているときのプロセス内での実行のされ方は、入力したコマンドが SET MODE [NO]INTERRUPT コマンドか、それとも SET MODE [NO]WAIT コマンドかによって異なります。省略時の設定 (SET MODE NOINTERRUPT) では、1つのプロセスが停止しても、デバッガは他のプロセスに関しては何のアクションも行いません。また、やはり省略時の設定 (SET MODE WAIT) では、デバッガは現在のプロセス・セットの中のすべてのプロセスが停止するのを待ってから、新しいコマン

## STEP

ドの入力を求めるプロンプトを表示します。詳細は、『OpenVMS デバッガ説明書』を参照してください。

VAX システムでは、STEP/OVER コマンドは Fortran Run-Time Library ルーチンへと、飛び越されずに、ステップすることになります。詳細は、『OpenVMS デバッガ説明書』を参照してください。

### 関連コマンド

- CALL
- EXIT
- GO
- SET BREAK/EXCEPTION
- SET MODE [NO]INTERRUPT
- SET PROCESS
- (SET,SHOW) STEP

## 例

```

1.  DBG> SHOW STEP
    step type: source, nosilent, by line,
              over routine calls
    DBG> STEP
    stepped to SQUARES$MAIN\%LINE 4
        4:      OPEN(UNIT=8, FILE='DATAFILE.DAT', STATUS='OLD')
    DBG>

```

この例では、SHOW STEP コマンドはSTEP コマンドに対して現在有効になっている省略時の修飾子を示します。この場合、STEP コマンドにはパラメータも修飾子も指定されていないため、ソース・コードの次の行を実行します。STEP コマンドの完了後、実行は4行目の先頭で中断されます。

```

2.  DBG> STEP 5
    stepped to MAIN\%LINE 47
        47:      SWAP(X,Y);
    DBG>

```

このコマンドはソース・コードの次の5行を実行します。STEP コマンドの完了後、実行は47行目の先頭で中断されます。

```

3.  DBG> STEP/INTO
    stepped to routine SWAP
        23: procedure SWAP (A,B: in out integer) is
    DBG> STEP
    stepped to MAIN\SWAP\%LINE 24
        24:      TEMP: integer := 0;
    DBG> STEP/RETURN
    stepped on return from MAIN\SWAP\%LINE 24 to MAIN\SWAP\%LINE 29
        29: end SWAP;
    DBG>

```

この例では、実行はSWAP ルーチンへの呼び出しで一時停止され、STEP/INTO コマンドは呼び出し先ルーチンの先頭までプログラムを実行します。STEP /RETURN コマンドはSWAP ルーチンの残りの部分をそのRET 命令まで(つまり、制御を呼び出し元ルーチンに戻す直前のポイントに至るまで)実行します。

```

4.  DBG> SET STEP INSTRUCTION
    DBG> SHOW STEP
    step type: source, nosilent, by instruction,
              over routine calls
    DBG> STEP
    stepped to SUB1\%LINE 26: MOVL      S^#4,B^-20(FP)
        26:      Z:integer:=4;
    DBG>

```

この例では、SET STEP INSTRUCTION コマンドは省略時のSTEP コマンド修飾子として/INSTRUCTIONを設定します。これはSHOW STEP コマンドでチェックされます。STEP コマンドは次の命令を実行します。STEP コマンドの完了後、実行はSUB1 モジュールの26行目の最初の命令(MOVL)で中断されます。

---

# STOP

指定されたプロセスのうち、実行中のプロセスすべてに割り込みます。

---

## フォーマット

STOP *[process-spec[, ... ]]*

---

## パラメータ

*process-spec*

このパラメータは、停止するプロセス・セットを指定します。省略時の設定は現在のプロセス・セットです。以下のいずれかの形式を使用します。

*[%PROCESS\_NAME] process-name*

スペースや小文字を含まないプロセス名。プロセス名にはワイルドカード文字(\*)を含めることができる。

*[%PROCESS\_NAME] "process-name"*

スペースまたは小文字を含むプロセス名。二重引用符(")の代わりに、一重引用符(') 使用することもできる。

*%PROCESS\_PID process\_id*

プロセス識別子 (PID, 16 進数)。

*[%PROCESS\_NUMBER] process-number*  
(または *%PROC process-number*)

デバッガの制御下に入ったときにプロセスに割り当てられた番号。新しい番号は、1 から順番に各プロセスに割り当てられる。EXIT コマンドまたは QUIT コマンドによってプロセスが終了した場合、そのデバッグ・セッション中にその番号が再割り当てされることがある。プロセス番号は SHOW PROCESS コマンドの実行で表示される。プロセスは、組み込みシンボル %PREVIOUS\_PROCESS および %NEXT\_PROCESS によってインデックスづけできるように、循環リスト内に順序づけされる。

*process-set-name*

DEFINE/PROCESS\_SET コマンドで定義された、プロセスのグループを表すシンボル。

*%NEXT\_PROCESS*

デバッガの循環プロセス・リスト中で可視プロセスの次のプロセス。

*%PREVIOUS\_PROCESS*

デバッガの循環プロセス・リスト中で可視プロセスの前のプロセス。

*%VISIBLE\_PROCESS*

シンボル、レジスタ値、ルーチン呼び出し、ブレークポイントなどの検索時に現在のコンテキストになっているスタック、レジスタ・セット、およびイメージを持つプロセス。

また、アスタリスク(\*)のワイルドカード文字を使用して、すべてのプロセスを指定することができます。

---

## 説明

STOP コマンドは指定されたプロセスに割り込みます。STOP コマンドを NOWAIT モードで使用して、実行中のプロセスを停止することができます。

---

## 例

1. all> SHOW PROCESS

	Number	Name	State	Current PC
	1	DBGK\$\$2727282C	break	SERVER\main\%LINE 18834
	2	USER1_2	running	not available
*	3	USER1_3	running	not available

all> CLIENTS> STOP

all> show process

	Number	Name	State	Current PC
	1	DBGK\$\$2727282C	break	SERVER\main\%LINE 18834
	2	USER1_2	interrupted	0FFFFFFFF800F7A20
*	3	USER1_3	interrupted	0FFFFFFFF800F7A20

all>

このコマンド・シーケンスは、まずすべてのプロセスを表示した後に、プロセス・セット・クライアントの中のプロセスを停止します。最後の SHOW PROCESS コマンドは新しいプロセス状態を示しています。

---

## SYMBOLIZE

可能な場合、メモリ・アドレスをシンボリック表現に変換します。

---

### フォーマット

SYMBOLIZE *address-expression*[, ... ]

---

### パラメータ

*address-expression*

シンボル化の対象となるアドレス式を指定します。ワイルドカード文字のアスタリスク(\*)は指定できません。

---

### 説明

アドレスが静的アドレスの場合、それは直前のシンボル名にオフセットを加えたものとしてシンボル化されます。アドレスがコード・アドレスでもあり、そのアドレスに該当する行番号を見つけられる場合、その行番号はシンボル化の対象となります。

アドレスがレジスタ・アドレスの場合、デバッガはそのレジスタに関する設定されているすべてのモジュールの中のすべてのシンボルを表示します。そのようなシンボルそれぞれの完全パス名が表示されます。レジスタ名そのもの(たとえば, "%R5") も表示されます。

アドレスが、設定されているモジュール内のルーチンの呼び出しフレームにある呼び出しスタック記憶位置である場合、デバッガはそのルーチン内のシンボルで、フレーム・ポインタ (FP) またはスタック・ポインタ (SP) に相対するアドレスを持つシンボルをすべて検索します。直前のシンボル名にオフセットを加えたものがそのアドレスのシンボル化として表示されます。アドレス指定が複雑すぎるシンボルは無視されます。

Alpha プロセッサでは、SYMBOLIZE *procedure-code-address* コマンドとSYMBOLIZE *procedure-descriptor-address* コマンドは両方ともこれらのアドレスで指定されるルーチン、エントリ・ポイントまたは Ada パッケージのパス名を表示します。

デバッガがアドレスのシンボル化を行えないと、メッセージが表示されます。



## 関連コマンド

EVALUATE/ADDRESS  
SET MODE [NO]LINE  
SET MODE [NO]SYMBOLIC  
(SET,SHOW) MODULE  
SHOW SYMBOL

---

例

1. DBG> SYMBOLIZE %R5  
address PROG\%R5:  
PROG\X  
DBG>

この例は、PROG ルーチンのローカル変数 X が R5 レジスタにあることを示します。

2. DBG> SYMBOLIZE %HEX 27C9E3  
address 0027C9E3:  
MOD5\X  
DBG>

このコマンドは、整数リテラル 27C9E3 を 16 進数値として扱い、可能であればそのアドレスをシンボリック表現に変換するようにデバッガに指示します。アドレスは MOD5 モジュールのシンボル X に変換されます。

---

## TYPE

ソース・コードの行を表示します。

---

### フォーマット

```
TYPE  [[module-name\]line-number[:line-number]  
      [, [module-name\]line-number[:line-number][, . . . ]]]
```

---

### パラメータ

**module-name**

表示の対象となるソース行を含むモジュールを指定します。行番号といっしょにモジュール名を指定する場合には、標準のパス名表記を使用します。つまり、モジュール名と行番号の間に円記号(\)を挿入してください。

モジュール名を指定しないと、デバッガは表示の対象となるソース行を見つけるために現在の有効範囲を使用します (現在の有効範囲は前回の SET SCOPE コマンドで設定されます。SET SCOPE コマンドを入力していなかった場合には、PC 範囲が使用されます)。SET SCOPE コマンドで有効範囲検索リストを指定すると、デバッガはソース行を検索するために最初に指定された有効範囲に対応するモジュールだけを検索します。

**line-number**

コンパイラ生成行番号 (1 つまたは複数のソース言語文にラベルを付けるために使用される番号) を指定します。

行番号を 1 つだけ指定すると、デバッガはその行番号に対応するソース・コードを表示します。

1 つ 1 つをコンマで区切ることにより行番号のリストを指定すると、デバッガは指定された行番号のそれぞれに対応するソース・コードを表示します。

コロン(:)で範囲内の開始行番号と終了行番号を区切り、行番号の範囲を指定すると、デバッガは指定範囲の行番号に対応するソース・コードを表示します。

1 で始まり、モジュールの最大行番号以上の数字で終わる行番号の範囲を指定することにより、モジュールのソース行全部を表示できます。

ソース行を 1 行表示したあとは、行番号を指定しないで TYPE コマンドを入力する (つまり、TYPE を入力したあと Return キーを押す) ことにより、同じモジュール内の次の行を表示できます。そのあとは、この手順を繰り返すこと、つまり、ソース・

プログラムを一度に 1 行ずつ読むことにより、次の行とそれに続く行を表示できます。

---

## 説明

TYPE コマンドは指定された行番号に対応するソース・コードの行を表示します。ソース・コードの行を示すためにデバッガが使用する行番号はコンパイラによって作成されます。行番号はコンパイラ生成リストと画面モードのソース表示に示されます。

TYPE コマンドでモジュール名を指定する場合、モジュールを設定しなければなりません。特定のモジュールが設定されているかを判断するには、SHOW MODULE コマンドを使用します。次に必要であれば、SET MODULE コマンドを使用します。

画面モードでは、TYPE コマンドの出力は出力表示でも DO 表示でもなく、現在のソース表示に出力されます。ソース表示は指定された行とその前後の行で表示ウィンドウに収まるだけの行を示します。

### 関連コマンド

```
EXAMINE/SOURCE
SET (BREAK,TRACE,WATCH)/[NO]SOURCE
SET MODE [NO]SCREEN
(SET,SHOW,CANCEL) SCOPE
SET STEP [NO]SOURCE
STEP/[NO]SOURCE
```

---

## 例

```
1. DBG> TYPE 160
   module COBOLTEST
     160: START-IT-PARA.
DBG> TYPE
   module COBOLTEST
     161:      MOVE SC1 TO ES0.
DBG>
```

この例では、最初の TYPE コマンドは 160 行目を表示します。このとき、その行番号を含むモジュールを検索するために現在の有効範囲を使用します。2 番目の TYPE コマンドは行番号を指定せずに入力されており、同じモジュール内の次の行を表示します。

## TYPE

```
2.  DBG> TYPE 160:163
      module COBOLTEST
      160: START-IT-PARA.
      161:      MOVE SC1 TO ES0.
      162:      DISPLAY ES0.
      163:      MOVE SC1 TO ES1.
DBG>
```

このコマンドはモジュールを検索するために現在の有効範囲を使用し、160 行目から 163 行目までを表示します。

```
3.  DBG> TYPE SCREEN_IO\7,22:24
```

このコマンドは SCREEN\_IO モジュール内にある 7 行目と 22 行目から 24 行目までを表示します。

---

## WAIT

デバッガは、ターゲット・プロセスが停止するのを待ってから、次のコマンドの入力を求めるプロンプトを表示ようになります。

---

### フォーマット

WAIT

---

### 説明

マルチプロセス・プログラムをデバッグしているときに WAIT コマンドを入力すると、デバッガは前のコマンドで指定されたすべてのプロセスが実行を完了するのを待ってから、新たなコマンドを受け付けて実行するためのプロンプトを表示ようになります。

#### 関連コマンド

STOP  
SET MODE [NO]INTERRUPT  
SET MODE [NO]WAIT

---

### 例

```
all> 2,3> GO;WAIT

processes 2,3
  break at CLIENT\main\%LINE 18814
    18814:      status = sys$qio(ENF$_ENF, mbxchan,
                      IO$_READVBLKIO$_WRITERCHECK, myiosb)
process 1
  break at SERVER\main\%LINE 18834
    18834:      if ((myiosb.iosb$w_status ==
                      SS$_NOREADER) && (pos_status != -1))
all>
```

このコマンド・シーケンスは、ターゲット・プロセス (この例では 2 と 3) を実行します。その後デバッガは、両方のプロセスがブレークポイントに達するのを待ってから、次のコマンドの入力を求めるプロンプトを表示します。

---

## WHILE

指定した言語式 (ブール式) が真と評価されているときに、一連のコマンドを実行します。

---

### フォーマット

WHILE *Boolean-expression* DO (*command*[: ... ])

---

### パラメータ

*Boolean-expression*

現在設定されている言語でブール値 (真または偽) と評価される言語式を指定します。

*command*

デバッガ・コマンドを指定します。複数のコマンドを指定する場合には、それぞれをセミコロン(;)で区切ってください。デバッガは、実行ごとにコマンド内のすべての式の構文をチェックしてから、それら进行评估します。

---

### 説明

WHILE コマンドは現在の言語でブール式を評価します。値が真の場合には、DO 句のコマンド・リストが実行されます。すると、WHILE コマンドは手順を繰り返し、ブール式が偽と評価されるまでその式を再評価し、コマンド・リストを実行します。

ブール式が偽の場合、WHILE コマンドは終了します。

関連コマンド

EXITLOOP  
FOR  
REPEAT

---

### 例

DBG> WHILE (X .EQ. 0) DO (STEP/SILENT)

このコマンドは、X が 0 でなくなるまでプログラム内の命令をステップ実行していくようデバッガに指示します (Fortran の例)。

## A

/ABORT 修飾子 . . . . . 2-251  
 ACTIVATE BREAK コマンド . . . . . 2-4  
 ACTIVATE TRACE コマンド . . . . . 2-7  
 ACTIVATE WATCH コマンド . . . . . 2-10, 2-65  
 /ACTIVATING 修飾子 . . . . . 2-4, 2-7, 2-26, 2-41,  
 2-59, 2-62, 2-185, 2-257  
 /ACTIVE 修飾子 . . . . . 2-251  
 /ADDRESS 修飾子 . . . . . 2-70, 2-112, 2-331  
 %ADDR 組み込みシンボル . . . . . 2-18  
 /AFTER 修飾子 . . . . . 2-185, 2-257, 2-269  
 /ALL 修飾子  
 ACTIVATE BREAK コマンド . . . . . 2-4  
 ACTIVATE TRACE コマンド . . . . . 2-7, 2-62  
 ACTIVATE WATCH コマンド . . . . . 2-10, 2-65  
 CANCEL BREAK コマンド . . . . . 2-26  
 CANCEL DISPLAY コマンド . . . . . 2-30  
 CANCEL TRACE コマンド . . . . . 2-41  
 CANCEL WATCH コマンド . . . . . 2-45  
 CANCEL WINDOW コマンド . . . . . 2-47  
 DEACTIVATE BREAK コマンド . . . . . 2-59  
 DELETE/KEY コマンド . . . . . 2-83  
 DELETE コマンド . . . . . 2-80  
 EXTRACT コマンド . . . . . 2-137  
 SEARCH コマンド . . . . . 2-171  
 SET IMAGE コマンド . . . . . 2-203  
 SET MODULE コマンド . . . . . 2-220  
 SET TASK コマンド . . . . . 2-251  
 SHOW DISPLAY コマンド . . . . . 2-289  
 SHOW KEY コマンド . . . . . 2-297  
 SHOW PROCESS コマンド . . . . . 2-310  
 SHOW TASK コマンド . . . . . 2-335  
 SHOW WINDOW コマンド . . . . . 2-344  
 ANALYZE/CRASH\_DUMP コマンド . . . . . 2-12  
 ANALYZE/PROCESS\_DUMP コマンド . . . . . 2-14  
 /APPEND 修飾子 . . . . . 2-137  
 /ARGUMENTS 修飾子 . . . . . 2-160, 2-162  
 /ASCIC 修飾子 . . . . . 2-86, 2-116, 2-146  
 /ASCID 修飾子 . . . . . 2-86, 2-116, 2-146  
 /ASCII 修飾子 . . . . . 2-86, 2-116, 2-147  
 ASCII 文字列型 . . . . . 2-85, 2-115, 2-266  
 /ASCIIW 修飾子 . . . . . 2-86, 2-116, 2-147  
 /ASCIZ 修飾子 . . . . . 2-86, 2-116, 2-147  
 /AST 修飾子 . . . . . 2-19  
 AST (非同期システム・トラップ)  
 AST 処理条件の表示 . . . . . 2-280  
 CALL コマンド . . . . . 2-18

## AST (非同期システム・トラップ) (続き)

許可 . . . . . 2-108  
 静的ウォッチポイント . . . . . 2-273  
 無効化 . . . . . 2-93  
 ATTACH コマンド . . . . . 2-16

## B

/BINARY 修飾子 . . . . . 2-103, 2-109, 2-112, 2-116,  
 2-147  
 /BOTTOM 修飾子 . . . . . 2-168  
 /BRANCH 修飾子 . . . . . 2-4, 2-7, 2-26, 2-41, 2-59,  
 2-62, 2-185, 2-257, 2-350  
 /BRIEF 修飾子 . . . . . 2-297, 2-310  
 /BYTE 修飾子 . . . . . 2-86, 2-103, 2-116, 2-147

## C

/CALLABLE\_EDT 修飾子 . . . . . 2-198  
 /CALLABLE\_LSEDT 修飾子 . . . . . 2-198  
 /CALLABLE\_TPU 修飾子 . . . . . 2-198  
 /CALLS 修飾子 . . . . . 2-220, 2-335  
 CALL コマンド . . . . . 2-18  
 および AST . . . . . 2-18  
 および浮動小数点パラメータ . . . . . 2-21  
 /CALL 修飾子 . . . . . 2-4, 2-7, 2-26, 2-41, 2-59,  
 2-62, 2-186, 2-257, 2-350  
 CANCEL ALL コマンド . . . . . 2-24  
 CANCEL BREAK コマンド . . . . . 2-26  
 CANCEL DISPLAY コマンド . . . . . 2-30  
 CANCEL MODE コマンド . . . . . 2-32  
 CANCEL RADIX コマンド . . . . . 2-34  
 CANCEL SCOPE コマンド . . . . . 2-36  
 CANCEL SOURCE コマンド . . . . . 2-37  
 CANCEL TRACE コマンド . . . . . 2-41  
 CANCEL TYPE/OVERRIDE コマンド . . . . . 2-44  
 CANCEL WATCH コマンド . . . . . 2-45  
 CANCEL WINDOW コマンド . . . . . 2-47  
 /CLEAR 修飾子 . . . . . 2-98  
 /COMMAND 修飾子 . . . . . 2-70, 2-162  
 /CONDITION\_VALUE 修飾子 . . . . . 2-109, 2-116  
 CONNECT コマンド . . . . . 2-49  
 デバッガの制御下にあるプロセスのサブプロセス  
 以外 . . . . . 2-51  
 Ctrl/C キー・シーケンス . . . . . 2-53  
 Ctrl/W キー・シーケンス . . . . . 2-55, 2-100  
 Ctrl/Y-DEBUG  
 HP DECwindows Motif for OpenVMS ユーザ・  
 インタフェースでは使用できない . . . . . 2-56

## Ctrl/Y-DEBUG (続き)

保持デバッガでは使用できない . . . . . 2-56  
Ctrl/Y キー・シーケンス . . . . . 2-56  
Ctrl/Z キー・シーケンス . . . . . 2-58  
/CURRENT 修飾子 . . . . . 2-234

## D

/D\_FLOAT 修飾子 . . . . . 2-86, 2-116, 2-147  
/DATE\_TIME 修飾子 . . . . . 2-86, 2-116, 2-147  
DEACTIVATE BREAK コマンド . . . . . 2-59  
DEACTIVATE TRACE コマンド . . . . . 2-62  
DEBUG コマンド . . . . . 2-56  
/DECIMAL 修飾子 . . . . . 2-103, 2-109, 2-112,  
2-116, 2-147  
DECLARE コマンド . . . . . 2-67  
/DEFAULT 修飾子 . . . . . 2-117, 2-147  
/DEFINED 修飾子 . . . . . 2-331  
DEFINE/KEY コマンド . . . . . 2-73  
DEFINE/PROCESS\_SET コマンド . . . . . 2-77  
DEFINE/TRANSLATION\_ATTR=CONCEALED コ  
マンド . . . . . 2-242  
DEFINE コマンド . . . . . 2-70  
省略時の修飾子の設定 . . . . . 2-196  
省略時の修飾子の表示 . . . . . 2-288  
/DEFINITIONS 修飾子 . . . . . 2-117  
DELETE/KEY コマンド . . . . . 2-82  
DELETE コマンド . . . . . 2-80  
DELETE/TYPE 大文字小文字を区別する言  
語 . . . . . 2-89  
DEPOSIT コマンド . . . . . 2-85  
%DESCR 組み込みシンボル . . . . . 2-18  
/DIRECTORY 修飾子 . . . . . 2-297  
/DIRECT 修飾子 . . . . . 2-331  
DISABLE AST コマンド . . . . . 2-93  
DISCONNECT コマンド . . . . . 2-94  
DISPLAY コマンド . . . . . 2-96  
/DISPLAY 修飾子 . . . . . 2-37, 2-240, 2-323  
/DOWN 修飾子 . . . . . 2-134, 2-151, 2-168  
DO 句  
形式 . . . . . 1-2  
終了 . . . . . 2-128, 2-155  
DUMP コマンド . . . . . 2-103  
/DYNAMIC 修飾子 . . . . . 2-98, 2-226, 2-310

## E

/ECHO 修飾子 . . . . . 2-74  
EDIT コマンド . . . . . 2-106  
/EDIT 修飾子 . . . . . 2-37, 2-240, 2-323  
ENABLE AST コマンド . . . . . 2-108  
/ERROR 修飾子 . . . . . 2-177  
EVALUATE/ADDRESS コマンド . . . . . 2-112  
EVALUATE コマンド . . . . . 2-109  
/EVENT 修飾子 . . . . . 2-4, 2-7, 2-26, 2-41, 2-59,  
2-62, 2-186, 2-258  
/EXACT 修飾子 . . . . . 2-37, 2-240

EXAMINE コマンド . . . . . 2-115  
/EXCEPTION 修飾子 . . . . . 2-5, 2-8, 2-26, 2-41,  
2-60, 2-63, 2-186, 2-258, 2-350  
EXITLOOP コマンド . . . . . 2-132  
EXIT コマンド . . . . . 2-128  
/EXIT 修飾子 . . . . . 2-106  
EXPAND コマンド . . . . . 2-133  
/EXTENDED\_FLOAT 修飾子 . . . . . 2-87, 2-117,  
2-147  
EXTRACT コマンド . . . . . 2-136

## F

/F\_FLOAT 修飾子 . . . . . 2-87, 2-117, 2-147  
/FLOAT 修飾子 . . . . . 2-87, 2-117, 2-147  
FOR コマンド . . . . . 2-138  
/FPCR 修飾子 . . . . . 2-117  
/FULL 修飾子 . . . . . 2-310, 2-331, 2-335

## G

/G\_FLOAT 修飾子 . . . . . 2-87, 2-117, 2-147  
/GENERATE 修飾子 . . . . . 2-98  
GO コマンド . . . . . 2-140

## H

/H\_FLOAT 修飾子 . . . . . 2-87, 2-117, 2-148  
/HANDLER 修飾子 . . . . . 2-5, 2-27, 2-60, 2-186  
/HEAP\_ANALYZER 修飾子 . . . . . 2-160, 2-162  
HELP コマンド . . . . . 2-143  
/HEXADECIMAL 修飾子 . . . . . 2-103, 2-109, 2-112,  
2-117, 2-148  
/HIDE 修飾子 . . . . . 2-98  
/HOLD 修飾子 . . . . . 2-251, 2-335  
HP DECwindows Motif for OpenVMS  
MONITOR コマンド . . . . . 2-146

## I

/IDENTIFIER 修飾子 . . . . . 2-171  
/IF\_STATE 修飾子 . . . . . 2-74  
IF コマンド . . . . . 2-145  
/INPUT 修飾子 . . . . . 2-177, 2-230, 2-346  
/INSTRUCTION 修飾子  
ACTIVATE BREAK コマンド . . . . . 2-5  
ACTIVATE TRACE コマンド . . . . . 2-8, 2-63  
CANCEL BREAK コマンド . . . . . 2-27  
CANCEL TRACE コマンド . . . . . 2-42  
DEACTIVATE BREAK コマンドでの . . . . . 2-60  
DEPOSIT コマンドでの . . . . . 2-87  
EXAMINE コマンドでの . . . . . 2-118  
MONITOR コマンドでの . . . . . 2-148  
SELECT コマンドでの . . . . . 2-177  
SET BREAK コマンドでの . . . . . 2-187  
SET TRACE コマンドでの . . . . . 2-258  
STEP コマンドでの . . . . . 2-350



/INTO 修飾子 . . . . . 2-187, 2-258, 2-270, 2-351  
/INT 修飾子 . . . . . 2-148

## J

/JSB 修飾子 . . . . . 2-188, 2-259, 2-351

## L

/LATEST 修飾子 . . . . . 2-37, 2-241  
/LEFT 修飾子 . . . . . 2-134, 2-151, 2-168  
%LINE 組み込みシンボル  
GO コマンド . . . . . 2-140  
/LINE 修飾子 . . . . . 2-5, 2-8, 2-27, 2-42, 2-60,  
2-63, 2-118, 2-188, 2-259  
/LOCAL 修飾子 . . . . . 2-70, 2-80, 2-331  
/LOCK\_STATE 修飾子 . . . . . 2-74  
/LOG 修飾子 . . . . . 2-74, 2-83  
/LONG\_FLOAT 修飾子 . . . . . 2-87, 2-118, 2-148  
/LONG\_LONG\_FLOAT 修飾子 . . . . . 2-87, 2-118,  
2-148  
/LONGWORD 修飾子 . . . . . 2-87, 2-103, 2-118,  
2-148  
/LONG 修飾子 . . . . . 2-148  
LSE (Language Sensitive Editor) . . . . . 2-106

## M

/MARK\_CHANGE 修飾子 . . . . . 2-99  
/MODIFY 修飾子 . . . . . 2-188, 2-259  
/MODULE 修飾子 . . . . . 2-37, 2-234, 2-241  
MONITOR コマンド . . . . . 2-146  
MOVE コマンド . . . . . 2-150

## N

/NEW 修飾子 . . . . . 2-163  
/NEXT 修飾子 . . . . . 2-171

## O

/OCTAL 修飾子 . . . . . 2-103, 2-109, 2-112, 2-118,  
2-148  
/OCTAWORD 修飾子 . . . . . 2-87, 2-118, 2-148  
OpenVMS Alpha システム・コード・デバッグ  
CONNECT コマンド . . . . . 2-49  
REBOOT コマンド . . . . . 2-158  
/OPERANDS 修飾子 . . . . . 2-118, 2-217  
/ORIGINAL 修飾子 . . . . . 2-38, 2-241  
/OUTPUT 修飾子 . . . . . 2-177, 2-230, 2-347  
/OVERRIDE 修飾子 . . . . . 2-34, 2-44, 2-230, 2-267,  
2-314, 2-341  
/OVER 修飾子 . . . . . 2-189, 2-259, 2-270, 2-351

## P

/PACKED 修飾子 . . . . . 2-87, 2-119  
/PAGE 修飾子 . . . . . 2-254  
PC (プログラム・カウンタ)  
SHOW CALLS 表示 . . . . . 2-284  
/POP (省略時の設定)/NOPOP . . . . . 2-99  
/POP 修飾子 . . . . . 2-228  
/PREDEFINED 修飾子  
ACTIVATE BREAK コマンド . . . . . 2-5  
ACTIVATE TRACE コマンド . . . . . 2-8, 2-63  
CANCEL ALL コマンド . . . . . 2-24  
CANCEL BREAK コマンド . . . . . 2-27  
CANCEL TRACE コマンド . . . . . 2-42  
DEACTIVATE BREAK コマンド . . . . . 2-60  
SHOW BREAK コマンドでの . . . . . 2-282  
SHOW TRACE コマンドでの . . . . . 2-339  
/PRIORITY 修飾子 . . . . . 2-252, 2-335  
/PROCESS\_SET 修飾子 . . . . . 2-77  
/PROCESS 修飾子 . . . . . 2-99  
/PROGRAM 修飾子 . . . . . 2-178  
/PROMPT 修飾子 . . . . . 2-178  
/PSL 修飾子 . . . . . 2-119  
/PSR 修飾子 . . . . . 2-119  
/PSW 修飾子 . . . . . 2-119  
/PS 修飾子 . . . . . 2-119  
/PUSH 修飾子 . . . . . 2-100

## Q

/QUADWORD 修飾子 . . . . . 2-88, 2-103, 2-119,  
2-148  
QUIT コマンド . . . . . 2-155

## R

REBOOT コマンド . . . . . 2-158  
/REFRESH 修飾子 . . . . . 2-100  
%REF 組み込みシンボル . . . . . 2-18  
/RELATED 修飾子 . . . . . 2-220, 2-304  
/REMOVE 修飾子 . . . . . 2-100, 2-148  
REPEAT コマンド . . . . . 2-159  
RERUN コマンド . . . . . 2-160  
/RESTORE 修飾子 . . . . . 2-252  
/RETURN 修飾子 . . . . . 2-189, 2-260, 2-352  
/RIGHT 修飾子 . . . . . 2-134, 2-151, 2-168  
RST (実行時シンボル・テーブル)  
シンボルの表示 . . . . . 2-330  
シンボル・レコードの挿入 . . . . . 2-220  
モジュールの表示 . . . . . 2-304  
RUN コマンド  
デバッグ・コマンド . . . . . 2-162

## S

/S_FLOAT 修飾子	2-119
/SAVE_VECTOR_STATE 修飾子	2-19
SAVE コマンド	2-165
/SAVE 修飾子	2-160
/SCREEN_LAYOUT 修飾子	2-137
SCROLL コマンド	2-167
/SCROLL 修飾子	2-178
SDA コマンド	2-174
SEARCH コマンド	2-170
省略時の修飾子の設定	2-238
省略時の修飾子の表示	2-319
SELECT コマンド	2-176
/SEMANTIC_EVENT 修飾子	2-352
/SET_STATE 修飾子	2-74
SET ABORT_KEY コマンド	2-181
SET ATSIGN コマンド	2-183
SET BREAK/UNALIGNED_DATA コマンドおよび システム・サービス・ルーチン	2-192
SET BREAK コマンド	2-184
SET DEFINE コマンド	2-196
SET EDITOR コマンド	2-198
SET EVENT_FACILITY コマンド	2-201
SET IMAGE コマンド	2-203
シンボル定義への作用	2-71
SET KEY コマンド	2-205
SET LANGUAGE コマンド	2-207, 2-210
SET LOG コマンド	2-211
SET MARGINS コマンド	2-213
SET MODE [NO]DYNAMIC コマンド	2-216
SET MODE [NO]G_FLOAT コマンド	2-216
SET MODE [NO]INTERRUPT コマンド	2-216
SET MODE [NO]KEYPAD コマンド	2-217
SET MODE [NO]LINE コマンド	2-217
SET MODE [NO]OPERANDS コマンド	2-217
SET MODE [NO]SCREEN コマンド	2-217
SET MODE [NO]SCROLL コマンド	2-217
SET MODE [NO]SYMBOLIC コマンド	2-218
SET MODE [NO]WAIT コマンド	2-218
SET MODE コマンド	2-216
SET MODULE コマンド	2-220
SET OUTPUT [NO]LOG コマンド	2-223
SET OUTPUT [NO]SCREEN_LOG コマ ンド	2-223
SET OUTPUT [NO]TERMINAL コマン ド	2-223
SET OUTPUT [NO]VERIFY コマンド	2-223
SET OUTPUT コマンド	2-223
SET PROCESS コマンド	2-225
SET PROMPT コマンド	2-228
SET RADIX コマンド	2-230
SET SCOPE コマンド	2-233
SET SEARCH コマンド	2-238
SET SOURCE コマンド	2-240
取り消されたルート・ディレクトリ論理 名	2-242

SET STEP コマンド	2-246
SET TASK/ACTIVE コマンド, POSIX Threads 用 以外	2-251
SET TASK コマンド	2-250
SET TERMINAL コマンド	2-254
SET THREAD コマンド	2-250
SET TRACE コマンド	2-256
SET TYPE/OVERRIDE コマンド	2-266
SET TYPE コマンド	2-266
SET WATCH コマンド	2-269
SET WINDOW コマンド	2-277
/SFPCR 修飾子	2-119
/SHARE 修飾子	2-189, 2-260, 2-304, 2-352
SHOW RADIX コマンド	2-314
/SHORT 修飾子	2-149
SHOW ABORT_KEY コマンド	2-279
SHOW AST コマンド	2-280
SHOW ATSIGN コマンド	2-281
SHOW BREAK コマンド	2-282
個々には表示されない命令	2-282
SHOW CALLS コマンド	2-284
SHOW DEFINE コマンド	2-288
SHOW DISPLAY コマンド	2-289
SHOW EDITOR コマンド	2-291
SHOW EVENT_FACILITY コマンド	2-292
SHOW EXIT_HANDLERS コマンド	2-293
SHOW IMAGE コマンド	2-294
SHOW KEY コマンド	2-296
SHOW LANGUAGE コマンド	2-299
SHOW LOG コマンド	2-300
SHOW MARGINS コマンド	2-301
SHOW MODE コマンド	2-303
SHOW MODULE コマンド	2-304
SHOW OUTPUT コマンド	2-308
SHOW PROCESS コマンド	2-309
SHOW SCOPE コマンド	2-316
SHOW SEARCH コマンド	2-319
SHOW SELECT コマンド	2-321
SHOW SOURCE コマンド	2-323
SHOW STACK コマンド	2-325
SHOW STEP コマンド	2-329
SHOW SYMBOL コマンド	2-330
SHOW TASK コマンド	2-334
SHOW TERMINAL コマンド	2-338
SHOW THREAD コマンド	2-334
SHOW TRACE コマンド	
個々の命令を非表示	2-339
SHOW TRACE コマンド	2-339
SHOW TYPE コマンド	2-341
SHOW WATCH コマンド	2-343
SHOW WINDOW コマンド	2-344
/SILENT 修飾子	2-190, 2-260, 2-270, 2-352
/SIZE 修飾子	2-100
/SOURCE 修飾子	2-119, 2-178, 2-190, 2-260, 2-270, 2-352
SPAWN/NOWAIT コマンドおよび保持デバッ カ	2-347

SPAWN コマンド . . . . . 2-346  
 SSI および静的ウォッチポイント . . . . . 2-273  
 \$START\_ALIGN\_FAULT\_REPORT ルーチンおよび  
     SET BREAK/UNALIGNED DATA . . . . . 2-192  
 /START\_POSITION 修飾子 . . . . . 2-199  
 START HEAP\_ANALYZER コマンド . . . . . 2-349  
 /STATE 修飾子 . . . . . 2-83, 2-205, 2-297, 2-336  
 /STATIC 修飾子 . . . . . 2-270  
 /STATISTICS 修飾子 . . . . . 2-336  
 STEP/SEMANTIC\_EVENT コマンド . . . . . 2-355  
 STEP コマンド . . . . . 2-350  
     省略時の修飾子の設定 . . . . . 2-246  
     省略時の修飾子の表示 . . . . . 2-329  
 STOP コマンド . . . . . 2-358  
 /STRING 修飾子 . . . . . 2-171  
 /SYMBOLIC 修飾子 . . . . . 2-120  
 SYMBOLIZE コマンド . . . . . 2-360  
 /SYSEMULATE 修飾子 . . . . . 2-5, 2-27, 2-60, 2-190  
 /SYSTEM 修飾子 . . . . . 2-190, 2-261, 2-353

## T

/TASK 修飾子 . . . . . 2-88, 2-120, 2-148  
 /TEMPORARY 修飾子 . . . . . 2-190, 2-261, 2-271  
 /TERMINATE 修飾子 . . . . . 2-75  
 /TERMINATING 修飾子 . . . . . 2-5, 2-8, 2-27, 2-42,  
     2-60, 2-63, 2-191, 2-261  
 /TIME\_SLICE 修飾子 . . . . . 2-252, 2-336  
 /TOP 修飾子 . . . . . 2-168  
 TYPE コマンド . . . . . 2-362  
 /TYPE 修飾子 . . . . . 2-88, 2-120, 2-331

## U

/UNALIGNED\_DATA 修飾子 . . . . . 2-5, 2-27, 2-60,  
     2-191  
 /UP 修飾子 . . . . . 2-134, 2-151, 2-168  
 /USE\_CLAUSE 修飾子 . . . . . 2-331  
 /USER 修飾子 . . . . . 2-5, 2-8, 2-24, 2-27, 2-42,  
     2-60, 2-63, 2-282, 2-339

## V

/VALUE 修飾子 . . . . . 2-71  
 %VAL 組み込みシンボル . . . . . 2-18  
 /VARIANT 修飾子 . . . . . 2-120  
 /VISIBLE 修飾子 . . . . . 2-226, 2-252, 2-310

## W

WAIT コマンド . . . . . 2-365  
 /WAIT 修飾子 . . . . . 2-347  
 /WCHAR\_T 修飾子 . . . . . 2-88, 2-121  
 WHEN 句 . . . . . 1-2  
 WHILE コマンド . . . . . 2-366  
 /WIDTH 修飾子 . . . . . 2-254  
 /WORD 修飾子 . . . . . 2-88, 2-103, 2-121, 2-149

/WRAP 修飾子 . . . . . 2-254

## X

/X\_FLOAT 修飾子 . . . . . 2-121

## ア

アット・マーク(@)  
     SET ATSIGN コマンド . . . . . 2-183  
     SHOW ATSIGN コマンド . . . . . 2-281  
     プロシージャ実行コマンド . . . . . 2-2  
 アドレス式  
     DEPOSIT コマンド . . . . . 2-85  
     EVALUATE/ADDRESS コマンド . . . . . 2-112  
     EXAMINE コマンド . . . . . 2-115  
     SET BREAK コマンド . . . . . 2-184  
     SET TRACE コマンド . . . . . 2-256  
     SET WATCH コマンド . . . . . 2-269  
     SYMBOLIZE コマンド . . . . . 2-360

## イ

イベント機能 . . . . . 2-201, 2-292

## ウ

ウィンドウ  
     画面モード  
         定義済み . . . . . 2-344  
         定義の削除 . . . . . 2-47  
         定義の作成 . . . . . 2-277  
         表示 . . . . . 2-344  
 ウォッチポイント  
     起動 . . . . . 2-10, 2-65  
     静的, AST . . . . . 2-273  
     設定 . . . . . 2-269  
     取り消し . . . . . 2-45  
     表示 . . . . . 2-343  
     プログラムの再実行による保存 . . . . . 2-10, 2-65,  
         2-160  
 ウォッチポイント, 静的 . . . . . 2-274

## オ

大文字小文字を区別する言語, DEPOSIT  
     /TYPE . . . . . 2-89  
 オペランド  
     命令 . . . . . 2-118, 2-217  
 オペレーティング・システムのデバッグ  
     CONNECT コマンド . . . . . 2-49

## カ

解釈	
プログラム実行	2-216
格納	
DEPOSIT コマンド	2-85
型	
SET TYPE コマンド	2-266
/TYPE 修飾子	2-88, 2-120, 2-331
上書き	2-44, 2-266, 2-341
現在の	2-266, 2-341
表示	2-341
画面サイズ	
設定	2-254
表示	2-338
画面モード	2-217
空フレーム・プロシージャと SHOW CALLS	
	2-285, 2-287, 2-327
感嘆符(!)	
コメント区切り文字	1-3

## キ

キー状態	2-73, 2-296
基数	
現在の	2-230
指定	2-230
取り消し	2-34
表示	2-314
キー定義	
削除	2-82
作成	2-73
表示	2-296
キーボード・モード	2-73, 2-217, 2-296
強制終了機能	2-53, 2-181, 2-279
行モード	2-217
共用可能イメージ	
SET BREAK/INTO コマンド	2-189
SET IMAGE コマンド	2-203
SET STEP INTO コマンド	2-247
SET TRACE/INTO コマンド	2-260
SHOW IMAGE コマンド	2-294
STEP/INTO コマンド	2-352

## ケ

言語	
現在の	2-207, 2-210
設定	2-207, 2-210
表示	2-299
言語式	
DEPOSIT コマンド	2-85
EVALUATE コマンド	2-109
FOR コマンド	2-138
IF コマンド	2-145
MONITOR コマンド	2-146
REPEAT コマンド	2-158, 2-159

### 言語式 (続き)

WHILE コマンド	2-366
検査	
EXAMINE コマンド	2-115
現在の	
イメージ	2-203, 2-294
型	2-266, 2-341
基数	2-230, 2-314
言語	2-207, 2-210, 2-299
表示	2-176, 2-321
有効範囲	2-233, 2-316
検索リスト	
ソース・ファイル	2-37, 2-240, 2-323
有効範囲	2-233, 2-316

## コ

コマンド行編集キー	1-3
コマンド形式	
デバッグ	1-1
コマンド・プロシージャ	
コマンドの表示	2-223
実行	2-2
終了	2-2, 2-128, 2-155
省略時のディレクトリ	2-183, 2-281
パラメータの引き渡し	2-67
表示の再作成	2-136
コメント, 形式	1-3
コロンの(:)	
範囲区切り文字	2-115

## シ

実行	
開始または再開	
CALL コマンドによる	2-18
GO コマンドによる	2-140
STEP コマンドによる	2-350
中断	
ウォッチポイントで	2-269
ブレークポイントで	2-184
ヒープ・アナライザの起動	2-349
マルチプロセス・プログラム	2-216
モニタ	
SHOW CALLS コマンドによる	2-284
トレースポイントで	2-256
システム空間	
SET BREAK コマンド	2-190
SET STEP コマンド	2-247
SET TRACE コマンド	2-261
STEP コマンド	2-353
システム・サービス, SSI によるサポー	
ト	2-274
システム・サービス呼び出しと静的ウォッチポイン	
ト	2-273
システム・ダンプ・デバッグ	2-12
集合体	
DEPOSIT コマンド	2-85

集合体 (続き)	
EXAMINE コマンド	2-115
SET WATCH コマンド	2-272
値の表示	2-115
値の変更	2-85
終了	
デバッグ・セッション	2-128, 2-155
終了ハンドラ	
実行	2-128
デバッグ	2-128
表示	2-293
出力構成	
設定	2-223
表示	2-308
シンボリック・モード	2-218
シンボル	
SET SCOPE コマンド	2-233
検索規則	2-234
シンボリック・モード	2-218
定義	2-71
表示	2-72, 2-330
呼び出しスタックに基づいた検索	2-234
シンボル化	
アドレス	2-360
レジスタ	2-360

## ス

スクロール・モード	2-217
-----------	-------

## セ

静的ウォッチポイント	2-274
静的ウォッチポイントおよび AST	2-273
静的ウォッチポイントとシステム・サービス呼び出し	2-273
セミコロン(;) コマンド区切り文字	1-3

## ソ

属性	
表示	2-176, 2-321
ソース・ディレクトリ	
検索リスト	2-37, 2-240, 2-241
表示	2-323
ソース表示	
SEARCH コマンド	2-170
SET SCOPE/CURRENT コマンド	2-234
SET STEP コマンド	2-246
TYPE コマンド	2-362
使用できない	2-240
マージン	2-301
呼び出しスタック上のルーチン	2-234
ソース・ファイル	
位置	2-37
記憶位置	2-240, 2-323
使用できない	2-240

ソース・ファイル (続き)	
正しいバージョンの	2-240, 2-241, 2-323

## タ

タスキング (マルチスレッド) プログラム	
SET EVENT_FACILITY コマンド	2-201
SET TASK コマンド	2-250
SET THREAD コマンド	2-250
SHOW EVENT_FACILITY コマンド	2-292
SHOW TASK コマンド	2-334
SHOW THREAD コマンド	2-334

## テ

デバッグ	
オンライン・ヘルプ	2-143
デバッグ・コマンド	
オンライン・ヘルプ	2-143
形式	1-1
概要	1-1
反復	2-138, 2-158, 2-159, 2-366

## ト

動的プロセス設定	2-226
動的モード	2-216
取り消されたルート・ディレクトリ論理名	2-242
トレースポイント	
起動	2-7
設定	2-256
遅延検出	2-257
取り消し	2-41
表示	2-339
プログラムの再実行による保存	2-8, 2-63, 2-160
無効化	2-62
例外	2-256

## ハ

ハイフン(-)	
行継続文字	1-2
バックスラッシュ(\)	
グローバル・シンボル指定子	2-233
パラメータ	
デバッグ・コマンド・プロシージャ	2-67
範囲	
コロンの(:)	2-115

## ヒ

引数, プログラム	2-160, 2-162
評価	
%CURVAL 組み込みシンボル	2-110
式	2-109
メモリ・アドレス	2-112

## 表示, デバッガ, 画面モード

移動	2-150
拡大	2-133
現在の	2-176
作成	2-96
縮小	2-133
除去	2-100
スクロール	2-167
属性	2-176, 2-321
選択	2-176
取り消し	2-30
抜き出し	2-136
非表示	2-98
表示	2-96, 2-289
ペーストボード	2-101
保存	2-165
リスト	2-289

## フ

フォーリン・コマンド	2-162
ブレイクポイント	
起動	2-4
境界に合っていないデータの	2-191
設定	2-184
遅延検出	2-185
取り消し	2-26
表示	2-282
プログラムの再実行による保存	2-6, 2-61, 2-160
無効化	2-59
例外	2-184
プログラム	
引数	2-160, 2-162
プロセス	
デバッガ制御からの解放	2-94
デバッガとの切り離し	2-94
デバッガを接続する	2-49
プロンプト	
デバッガ	2-228

## へ

ベクタ化されたプログラム	
CALL/[NO]SAVE_VECTOR_STATE コマン	
ド	2-19
EXAMINE/OPERANDS コマンド	2-118
ヘルプ	
デバッガの	2-143
変数	
モニタ	2-146

## ホ

保持デバッガおよび SPAWN/NOWAIT	2-347
------------------------	-------

## マ

マージン, ソース表示	2-213, 2-301
待ちモード	2-218
マルチプロセス・プログラム	
CALL コマンド	2-18
CONNECT コマンド	2-49
DEFINE/PROCESS_SET コマンド	2-77
DISCONNECT コマンド	2-94
EXIT コマンド	2-128
GO コマンド	2-140
QUIT コマンド	2-155
SET MODE [NO]INTERRUPT コマン	
ド	2-216
SET PROCESS コマンド	2-225
SHOW PROCESS コマンド	2-309
STEP コマンド	2-350
STOP コマンド	2-358
WAIT コマンド	2-365

## メ

命令	
SET SCOPE/CURRENT コマンド	2-234
オペランド	2-118, 2-217
表示	
呼び出しスタック上のルーチン	2-234
メッセージ, デバッガ	1-5

## モ

モジュール	
関連情報	2-304
設定	2-220
モード	
CANCEL MODE コマンド	2-32
SET MODE [NO]DYNAMIC コマン	
ド	2-216
SET MODE [NO]G_FLOAT コマンド	2-216
SET MODE [NO]INTERRUPT コマン	
ド	2-216
SET MODE [NO]KEYPAD コマンド	2-217
SET MODE [NO]LINE コマンド	2-217
SET MODE [NO]OPERANDS コマン	
ド	2-217
SET MODE [NO]SCREEN コマンド	2-217
SET MODE [NO]SCROLL コマンド	2-217
SET MODE [NO]SYMBOLIC コマン	
ド	2-218
SET MODE [NO]WAIT コマンド	2-218
SHOW MODE	2-303

## 問題点と制限事項

ウォッチポイント, 静的	2-274
ウォッチポイントのサポート	2-272
システム・サービス	2-274
静的ウォッチポイント	2-274

## ユ

### 有効範囲

SEARCH コマンド	2-170
SET SCOPE コマンド	2-233
TYPE コマンド	2-362
機械語命令ディスプレイの	2-234
現在の	2-233
検索リスト	2-36, 2-233, 2-316
省略時の	2-36, 2-316
省略時の設定	2-234
シンボル検索のための	2-36, 2-233, 2-316
設定	2-233
ソース表示の	2-234
取り消し	2-36
表示	2-316
呼び出しスタックとの関係	2-234
レジスタ表示	2-234

## ヨ

### 呼び出しスタック

機械語命令ディスプレイ	2-234
シンボル検索	2-233, 2-234
ソース表示	2-234
表示	2-284, 2-325
レジスタ表示	2-234

## ル

### ルーチン

SET SCOPE コマンド	2-233
表示	
ソース・コードの, 呼び出しスタック上	
の,	2-234
命令の, 呼び出しスタック上の	2-234
レジスタ値の, 呼び出しスタック上	
の	2-234
複数の起動	2-233
呼び出し	2-18
呼び出しスタック	2-233, 2-284

### ルート・ディレクトリ論理名

取り消し	2-242
------	-------

## レ

### 例外

#### 中断

例外ブレークポイント	2-186
------------	-------

### 例外ブレークポイントまたは例外トレースポイント

起動	2-5, 2-8, 2-63
----	----------------

### 例外ブレークポイントまたは例外トレースポイント (続き)

設定	2-186, 2-258
取り消し	2-26, 2-41
無効化	2-60

### レジスタ

SET SCOPE/CURRENT コマンド	2-234
シンボル化	2-360
表示	
呼び出しスタック上のルーチン	2-234

## ロ

### ログ・ファイル

デバッグ	2-223
名前	2-211, 2-300

## ワ

### ワークステーション

VMS を使用する場合のデバッグ・ウィンドウのポ ップアップ	2-228
ターミナル・エミュレータ画面サイズ	2-254

### 割り込み

コマンド実行	2-53
プログラム実行	2-49, 2-53, 2-56





HP OpenVMS デバッガ・コマンド・ディクショナリ

---

2005 年 4 月 発行

日本ヒューレット・パッカート株式会社

〒140-8641 東京都品川区東品川 2-2-24 天王洲セントラルタワー

電話 (03)5463-6600 (大代表)

---

AA-QUTWD-TE

