THE POWER TO KNOW.

Base SAS® 9.3 Utilities: リファレンス

SAS® ドキュメント

The correct bibliographic citation for this manual is as follows: SAS Institute Inc. 2011. *Base SAS® 9.3 Utilities: Reference*. Cary, NC: SAS Institute Inc.

Base SAS® 9.3 Utilities: Reference

ISBN 978–1–60764–905–2 (electronic book)
ISBN 978–1–60764–905–2

SAS Institute Inc., SAS Campus Drive, Cary, North Carolina 27513.

ISBN 978–1–60764–905–2
Printing 1, 2011 July

ISBN 978–1–60764–905–2
Printing 1, 2011 July

SAS® Publishing provides a complete selection of books and electronic products to help customers use SAS software to its fullest potential. For more information about our e-books, e-learning products, CDs, and hard-copy books, visit the SAS Publishing Web site at **support.sas.com/publishing** or call 1-800-727-3228.

目次

本書について

SAS 言語の構文規則

SAS 言語の構文規則の概要

SAS では、SAS 言語要素の構文ドキュメントに共通の規則を使用しています。これらの規則により、SAS 構文の構成要素を簡単に識別できます。規則は、次の項目に分類されます。

- 構文の構成要素
- スタイル規則
- 特殊文字
- SAS ライブラリと外部ファイルの参照

構文コンポーネント

言語要素の多くでは、その構文の構成要素はキーワードと引数から構成されます。キーワードのみ必要な言語要素もあります。また、キーワードに等号(=)が続く言語要素もあります。

キーワード
プログラムの作成ときに使用する SAS 言語要素名です。キーワードはリテラルであり、通常、構文の先頭の単語です。CALL ルーチンでは、最初の 2 つの単語がキーワードです。

次の SAS 構文の例では、構文の最初の単語がキーワードです。

CHAR (*string, position*)

CALL RANBIN (*seed, n, p, x*);

ALTER (*alter-password*)

BEST *w.*

REMOVE <*data-set-name*>

次の例では、CALL ルーチンの最初の 2 つの単語がキーワードです。

CALL RANBIN(*seed*, *n*, *p*, *x*)

引数なしで 1 つのキーワードから構成される SAS ステートメント構文もあります。

DO;
... SAS code ...
END;

2 つのキーワード値のいずれか 1 つの指定が必要なシステムオプションもあります。

DUPLEX | **NODUPLEX**

引数
数値定数、文字定数、変数、式のいずれかです。引数は、キーワードに続くか、キーワードの後ろの等号に続きます。SAS では、引数を使用して、言語要素を処理します。引数が必須の場合もオプションの場合もあります。構文では、オプションの引数にはかぎかっこが付けられます。

次の例では、*string* と *position* がキーワード CHAR に続きます。これらの引数は、CHAR 関数の必須引数です。

CHAR (*string, position*)

引数ごとに値が指定されます。次の例の SAS コードでは、引数 *string* の値として 'summer'、引数 *position* の値として 4 が指定されています。`x=char('summer', 4);`

次の例では、*string* と *substring* は必須引数ですが、*modifiers* と *startpos* はオプションの引数です。

FIND(*string, substring* <,*modifiers*> <,*startpos*>

注: 通常、SAS ドキュメントのサンプルコードは、小文字の固定幅フォントを使用して表記されます。コードの作成には、大文字も、小文字も、大文字と小文字の両方も使用できます。

スタイル規則

SAS 構文の説明に使用されるスタイル規則には、大文字太字、大文字、斜体の規則も含まれます。

大文字太字
関数名やステートメント名などの SAS キーワードを示します。次の例では、キーワード ERROR の表記には大文字太字が使用されています。

ERROR<*message*>;

大文字
リテラルの引数を示します。

次の CMPMODEL=システムオプションの例では、BOTH、CATALOG、XML がリテラルです。

CMPMODEL = BOTH | CATALOG | XML

斜体
ユーザー指定の引数または値を示します。斜体表記の項目は、ユーザー指定値であり、次のいずれかを表します。

- 非リテラルの引数。次の LINK ステートメントの例では、引数 *label* はユーザー指定値であるため、斜体で表記されています。

  LINK *label*;

- 引数に割り当てられる非リテラル値。

  次の FORMAT ステートメントの例では、引数 DEFAULT に変数の *default-format* が割り当てられます。

  FORMAT = *variable-1* <, ..., *variable-n format* ><DEFAULT = *default-format*>;

斜体表記の項目は、選択可能な引数リストの総称でもあります(*attribute-list* など)。複数の斜体表記の項目が使用される場合、項目は *item-1, ..., item-n* という形式で表記されます。

特殊文字

SAS 言語要素の構文には、次の特殊文字も使用されます。

=
等号は、一部の言語要素(システムオプションなど)のリテラル値を示します。

次の MAPS システムオプションの例では、等号は MAPS の値を設定します。

MAPS = *location-of-maps*

< >
かぎかっこはオプションの引数を示します。かぎかっこ付きでない引数は必須引数です。

次の CAT 関数の例では、少なくとも項目が 1 つ必要です。

CAT (*item-1 <, ..., item-n>*)

|
縦棒は、値グループから 1 つの値を選択できることを示します。縦棒で区切られている値は、相互排他です。

次の CMPMODEL=システムオプションの例では、属性を 1 つのみ選択できます。

CMPMODEL = BOTH | CATALOG | XML

...
省略記号は、省略記号に続く引数や引数グループの繰り返しを示します。省略記号とその後の引数にかぎかっこが付けられている場合、その引数はオプションです。

次の CAT 関数の例では、省略記号はオプションの項目を複数指定できることを示しています。

CAT (*item-1 <, ..., item-n>*)

'*value*' or "*value*"
単一引用符や二重引用符付きの引数は、その値も単一引用符または二重引用符を付ける必要があることを示します。

次の FOOTNOTE ステートメントの例では、引数 *text* には引用符が付けられています。

FOOTNOTE <*n*> <*ods-format-options* '*text*' | "*text*">;

;
セミコロンは、ステートメントまたは CALL ルーチンの終わりを示します。

次の例では、それぞれのステートメントはセミコロンで終了しています。
```
data namegame; length color name $8; color = 'black'; name =
'jack'; game = trim(color) || name; run;
```

SAS ライブラリと外部ファイルの参照

多くの SAS ステートメントなどの言語要素では、SAS ライブラリと外部ファイルを参照します。論理名(ライブラリ参照名またはファイル参照名)から参照を作成するのか、引用符付きの物理ファイル名を使用するかを選択できます。論理名を使用する場合、通

常、関連付けに SAS ステートメント(LIBNAME または FILENAME)を使用するのか、動作環境のコントロール言語を使用するのかを選択します。複数の方法を使用して、SAS ライブラリと外部ファイルを参照できます。動作環境によっては使用できない方法があります。

SAS ドキュメントでは、外部ファイルを使用する例には斜体のフレーズ *file-specification* を使用します。また、SAS ライブラリを使用する例には斜体フレーズ *SAS-library* を使用します。*SAS-library* は引用符付きであることに注意してください。

```
infile file-specification obs = 100;
libname libref 'SAS-library';
```

Base SAS ユーティリティの新機能 9.3

SAS 言語リファレンス: ディクショナリ

SAS 9.3 より前は、このドキュメントは *SAS 言語リファレンス: ディクショナリ*の一部でした。SAS 9.3 から *SAS 言語リファレンス: ディクショナリ*は 7 つのドキュメントに分割されました。

- *SAS データセットオプション: リファレンス*
- *SAS 出力形式と入力形式: リファレンス*
- *SAS 関数と CALL ルーチン: リファレンス*
- *SAS ステートメント: リファレンス*
- *SAS システムオプション: リファレンス*
- *SAS コンポーネントオブジェクト: リファレンス* (ハッシュオブジェクトと Java オブジェクトに関してのドキュメント)
- *Base SAS Utilities: リファレンス* (SAS DATA ステップデバッガおよび SAS Utility マクロ%DS2CSV に関してのドキュメント)

推奨資料

推奨資料のリストです。

- *SAS コンポーネントオブジェクト: リファレンス*
- *SAS 出力形式と入力形式: リファレンス*
- *SAS 関数と CALL ルーチン: リファレンス*
- *SAS 言語リファレンス: 解説編*
- *SAS マクロ言語: リファレンス*
- *SAS ステートメント: リファレンス*
- *SAS システムオプション: リファレンス*

SAS の刊行物の総一覧については、support.sas.com/bookstore にてご確認ください。
必要な書籍についてのご質問は、下記までお寄せください。

SAS Books
SAS Campus Drive
Cary, NC 27513-2414
電話: 1-800-727-3228
ファクシミリ: 1-919-677-8166
電子メール: sasbook@sas.com
Web アドレス: support.sas.com/bookstore

1 部

マクロユーティリティ

1 章

ユーティリティのディクショナリ

ディクショナリ

%DS2CSV マクロ

SAS データセットをカンマ区切りファイル(CSV)に変換します。

制限事項: このマクロは DATA ステップでは使用できません。オープンコードでのみ実行できるマクロです。

構文

%DS2CSV(*argument-1=value-1*, *argument-2=value-2 <…,argument-n=value-n>*)

入力と出力に影響する引数

csvfile=*external-filename*
フォーマットされた出力が書き出される CSV ファイルの名前を指定します。指定したファイルが存在しない場合、自動で作成されます。

注 CSVFREF 引数を使っている場合、CSVFILE 引数は使用できません。

csvfref=*fileref*
フォーマットされた出力が書き出される CSV ファイルの場所を示す SAS ファイル参照名を指定します。指定したファイルが存在しない場合、自動で作成されます。

注 CSVFILE 引数を使っている場合、CSVFREF 引数は使用できません。

openmode=REPLACE|APPEND
新規の CSV 出力で既存の指定ファイルの情報を上書きするか、既存のファイルの最後に追加するかを指定します。デフォルトの値は REPLACE です。現在の内容を置き換えたくない場合は、OPENMODE=APPEND を指定すると、新規の CSV 形式の出力は既存のファイルの最後に追加されます。

注 z/OS の区分データセット(PDS)に出力する場合、OPENMODE=APPEND は有効ではありません。

MIME および HTTP ヘッダに影響する引数

MIME および HTTP ヘッダの詳細については、それぞれ順に Internet Request for Comments (RFC) documents RFC 1521 (http://asg.web.cmu.edu/rfc/rfc1521.html)および RFC 1945 (http://asg.web.cmu.edu/rfc/rfc1945.html)を参照してください。

conttype=Y | N
Content type ヘッダを書くかどうかを指定します。デフォルトはヘッダを書く設定です。

制限事項 この引数は RUNMODE=S の場合のみ有効です。

contdisp=Y | N
Content disposition ヘッダを書くかどうかを指定します。デフォルトはヘッダを書く設定です。

制限事項 この引数は RUNMODE=S の場合のみ有効です。

注 CONTDISP=N を指定した場合、SAVEFILE 引数は無視されます。

mimehdr1=*MIME/HTTP-header*
最初に書き出される MIME または HTTP ヘッダに使用されるテキストを指定します。このヘッダは content type と disposition ヘッダの後に書かれます。デフォルトでは、このヘッダは書かれない設定です。

制限事項 この引数は RUNMODE=S の場合のみ有効です。

mimehdr2=*MIME/HTTP-header*
2 つめに書き出される MIME または HTTP ヘッダに使用されるテキストを指定します。このヘッダは content type と disposition ヘッダの後に書かれます。デフォルトでは、このヘッダは書かれない設定です。

制限事項 この引数は RUNMODE=S の場合のみ有効です。

mimehdr3=*MIME/HTTP-header*
RUNMODE=S が指定されているときに、3 つめに書き出される MIME または HTTP ヘッダに使用されるテキストを指定します。このヘッダは content type と disposition ヘッダの後に書かれます。デフォルトでは、このヘッダは書かれない設定です。

制限事項 この引数は RUNMODE=S の場合のみ有効です。

mimehdr4=*MIME/HTTP-header*
4 つめに書き出される MIME または HTTP ヘッダに使用されるテキストを指定します。このヘッダは content type と disposition ヘッダの後に書かれます。デフォルトでは、このヘッダは書かれない設定です。

制限事項 この引数は RUNMODE=S の場合のみ有効です。

mimehdr5=*MIME/HTTP-header*
5 つめに書き出される MIME または HTTP ヘッダに使用されるテキストを指定します。このヘッダは content type と disposition ヘッダの後に書かれます。デフォルトでは、このヘッダは書かれない設定です。

runmode=S | B
%DS2CSV マクロをバッチモードで実行するかサーバーモードで実行するかを指定します。この引数のデフォルトは RUNMODE=S です。

- *Server mode* (RUNMODE=S)は Application Dispatcher プログラムやストリーミング出力ストアドプロセスなどで使われます。サーバーモードでは DS2CSV によって適した MIME または HTTP ヘッダが生成されます。Application Dispatcher についての詳細は、**http://support.sas.com/rnd/web/intrnet/dispatch.html** にある Application Dispatcher のドキュメントを参照してください。
- *Batch mode* (RUNMODE=B)ということは、SAS プログラムエディタで DS2CSV をサブミットしているか、それを SAS プログラムに含んでいるかになります。

  注: バッチモードを指定した場合、HTTP ヘッダは書かれません。

制限事項 RUNMODE=S は SAS/IntrNet および Stored Process Server で使用されている場合のみ有効です。

savefile=*filename*
ウェブブラウザの**名前を付けて保存**ダイアログボックスに表示するファイル名を指定します。デフォルトの値はデータセット名に".csv"を付加したものです。

制限事項 この引数は RUNMODE=S の場合のみ有効です。

注 この引数は CONTDISP=N が指定されている場合のみ有効です。

CSV 作成に影響する引数

colhead=Y | N
CSV ファイルに列ヘッダを含むかどうかを指定します。使われる列ヘッダは LABELS 引数の設定によります。デフォルトでは、列ヘッダは CSV ファイルの最初のレコードとして含まれます。

data=*SAS-data-set-name*
CSV ファイルに変換したいデータを含む SAS データセットを指定します。この引数は必須です。ただし、データセット名を省略した場合は、DS2CSV は一番最近に作成された SAS データセットを使用します。

formats=Y | N
データセットに定義済みの変数の形式を CSV ファイルの値に適用するかどうかを指定します。デフォルトでは、値が CSV ファイルに追加される前にすべての形式が適用されます。形式はデータセットに保存されているもののみ、適用可能です。

labels=Y | N
データセットに定義されている SAS 変数ラベルを列ヘッダに使用するかどうかを指定します。DS2CSV マクロはデフォルトで変数ラベルを使用します。変数が SAS ラベルを持っていない場合は変数名を使用します。labels=N を指定すると列ヘッダには SAS ラベルの代わりに変数名が使用されます。

参照項目 列ヘッダに関しての詳細は colhead (5 ページ)を参照してください。

pw=*password*
パスワード保護のあるデータセットにアクセスするのに必要なパスワードを指定します。この引数はデータセットに READ または PW パスワードがある場合に必要になります。(データセットに WRITE または ALTER パスワードしかない場合はこの引数の指定は必要ありません。)

sepchar=*separator-character*
区切り文字として使用する文字を指定します。2 文字の 16 進コードを指定するか、または省略してデフォルト設定を使います。デフォルト設定は ASCII システムでは 2C で EBCDIC システムでは 6B です。(これらの設定はそれぞれのシステムでカンマ(,)を表します。)

var=*var1 var2 ...*
CSV ファイルにに含む変数とそれらの順序を指定します。データセットにあるすべての変数を含む場合は、この引数は指定しないでください。変数のサブセットのみを含む場合は、変数名をシングルスペースで区切ってリストします。変数名のリストにカンマは使わないでください。

制限事項 値の範囲は無効です。例:**`var1-var4`**.

where=*where-expression*
SAS データセットのオブザベーションを選択するのに有効な WHERE 句を指定します。この引数を使うと *where-expression* に指定する条件に基づいてデータをサブセット化できます。

詳細

DS2CSV マクロは SAS データセットをカンマ区切りファイル(CSV)に変換します。他の種類の出力ファイルを作成する場合には、区切り文字に 16 進数コードを指定できます(たとえば、CSV ファイルなど)。

例

次の例では%DS2CSV マクロを使って SASHELP.RETAIL データセットを CSV ファイルに変換します。

```
%ds2csv (data=sashelp.retail, runmode=b, csvfile=c:\temp\retail.csv);
```

2 部

DATA ステップデバッガ

2 章

DATA ステップデバッガの使用

紹介

デバッグとは

デバッグとはプログラムから論理エラーを除去するプロセスです。構文エラーとは違い、論理エラーでプログラムの実行が停止することはありません。その代わり、プログラムで予期しない結果が生じる原因となります。たとえば、在庫の流れを把握する DATA ステップを作成した場合、プログラムでは 在庫切れになっているにもかかわらず倉庫が一杯である状態では、プログラムに論理エラーがあります。

DATA ステップをデバッグするには、次のタスクのどれかを実行します:

- ステップの数行を他の DATA ステップにコピーして実行し、ステートメントの結果を出力する。
- PUT ステートメントを DATA ステップの特定の場所に挿入してサブミットし、SAS ログに表示されている値を検証する。
- DATA ステップデバッガを使用する。

SAS ログでもデータエラーを見つけることはできますが、DATA ステップデバッガではより簡単でインタラクティブにデータステップの論理エラーや時にはデータエラーも発見することができます。

DATA ステップデバッガについて

DATA ステップデバッガは Base SAS ソフトウェアの一部でウィンドウとコマンド群で構成されます。コマンド発行時に、DATA ステップステートメントを1つずつ実行および一時停止することができ、ウィンドウには変数値の結果が表示されます。表示される結果を検証して、論理エラーの箇所を限定できます。デバッガはインタラクティブなので、1つのデバッグセッションでコマンド発行と結果検証を必要なだけ繰り返すことができます。デバッガを起動するには、DATA ステートメントに DEBUG オプションを加えてプログラムを実行します。

DATA ステップデバッガでは次のタスクを実行できます。

- ステートメントを1つずつまたはグループで実行
- 1つ以上のステートメントの実行をスキップ
- DATA ステップの反復中または指定の条件において、特定のステートメントで実行を一時停止し、コマンドで実行の再開
- 特定の変数の値をモニターし、その値が変更する箇所で実行を一時停止
- 変数の値を表示し、新しい値を付与
- 変数の属性を表示
- 各デバッガコマンドのヘルプを表示
- ファンクションキーにデバッガコマンドを割り当てる
- マクロ機能を使用してカスタマイズされたデバッガコマンドを生成

基本的な使用

デバッガセッションの操作方法

DEBUG オプション付きの DATA ステップをサブミットすると、SAS はステップをコンパイルし、デバッガウィンドウを表示し、実行開始のデバッガコマンドを入力するまで一時停止します。たとえば、GO コマンドの実行を開始した場合、SAS は DATA ステップの各ステートメントを実行します。DATA ステップの特定の行で実行を一時停止させたい場合、BREAK コマンドを使って選択のステートメントにブレークポイントを設定します。そして GO コマンドを発行します。GO コマンドはブレークポイントに到達するまで、実行を開始または再開します。

DATA ステップを1ステートメントずつまたは数ステートメントずつ実行するには、STEP コマンドを使用します。デフォルトでは STEP コマンドは ENTER キーにマッピングされています。

デバッグセッションでは、DATA ステップのステートメントはデバッグセッション外で実行される回数と同じだけ実行されます。最後の反復終了後、DEBUGGER LOG ウィンドウにメッセージが表示されます。

デバッグセッションでは、DATA ステップの実行終了後は、DATA ステップ実行の再開はできません。SAS セッションで DATA ステップを再度サブミットしてください。ただし、実行終了後に変数の最後の値を検証することはできます。

一回に 1 つの DATA ステップのみデバッグできます。デバッガは DATA ステップでのみ使用でき、PROC ステップでは使用できません。

ウィンドウの使用

DATA ステップデバッガには DEBUGGER LOG ウィンドウと DEBUGGER SOURCE の2つのメインのウィンドウがあります。DEBUG オプションをつけて DATA ステップを実行するとウィンドウが表示されます。

The DEBUGGER LOG ウィンドウは発行するデバッガーコマンドとその結果を記録します。最後の行はデバッガコマンドを発行するデバッガコマンドラインです。デバッガコマンドラインには、より大きい(>)プロンプトが表示されています。

DEBUGGER SOURCE ウィンドウには、デバッグしている DATA ステップを構成する SAS ステートメントが表示されます。ウィンドウはプログラムのデバッグ中に DATA ステップのどの位置かを表示します。ウィンドウでは、SAS ステートメントは SAS ログと同じように行番号を持ちます。

コマンドラインにウィンドウ環境のコマンドを入力できます。ファンクションキーでコマンドを実行することもできます。

コマンドの入力

コマンドとその説明のリストは、“カテゴリ別の DATA ステップデバッガコマンド” (27 ページ)を参照してください。

デバッガコマンドラインに DATA ステップデバッガコマンドを入力します。コマンド入力の際には次のルールに従ってください。

- コマンドは一行に入力します(DO グループは除く)。
- DO グループは複数行にわたることもあります。
- 複数のコマンドを入力する場合は、セミコロンでコマンドを区切ります。

```
examine _all_; set letter='bill'; examine letter
```

式の処理

デバッガ式には“SAS 式の演算子” (*SAS 言語リファレンス: 解説編* 6 章)に説明されているすべての SAS 演算子が使用できます。デバッガ式には関数を含むことはできません。.

デバッガ式は1行内でおさまるようにしてください。式は2行にわたることはできません。

コマンドをファンクションキーに割り当てる

ファンクションキーにデバッガコマンドを割り当てるには、KEYS ウィンドウを開きます。カーソルを割り当てるファンクションキーの定義カラムにおき、DSD のあとにコマンドを入力します。ファンクションキーに複数のコマンドを割り当てるには、コマンドを(セミコロンで区切って)引用符で囲みます。変更を保存するのを忘れないようにしてください。ファンクションキーに割り当てられたコマンドの例です。

- `dsd step3`
- `dsd 'examine cost saleprice; go 120;'`

マクロ機能とデバッガの併用

デバッグツールとしてマクロを使用する

デバッガの DEBUGGER LOG コマンドラインからマクロを起動するには、SAS マクロ機能を使います。マクロを定義して、デバッガコマンドラインから %LET 等のマクロプログラムステートメントを使用できます。

マクロは一連のデバッガコマンドを保存するのに便利です。DEBUGGER LOG コマンドラインでマクロを実行すると、一連のデバッガコマンドが生成されます。マクロと引数を使用して、様々な状況において異なる一連のデバッガコマンドを生成することもできます。

マクロを用いてカスタマイズしたデバッグコマンドを作成する

DEBUGGER LOG のコマンドラインにマクロを定義すればカスタマイズのデバッグコマンドが作成できます。そしてコマンドラインからマクロを起動します。 たとえば、変数 COST を検証して、5 つのステートメントを実行して、そして変数 DURATION を検証するには、次のマクロを定義します（このケースでは、マクロは EC と呼ばれています）。この例では、EXAMINE コマンドにはエリアスが使われています。

```
%macro ec; ex cost; step 5; ex duration; %mend ec;
```

コマンドを発行するには、DEBUGGER LOG コマンドラインからマクロ EC を起動します。

```
%ec
```

DEBUGGER LOG は COST の値を表示し、次の 5 ステートメントを実行し、そして DURATION の値を表示します。

注: DEBUGGER LOG コマンドラインでマクロを定義する場合、現在のデバッグセッション中でだけ、そのマクロを使用できます。なぜなら、マクロが永久保存されていないからです。永久保存されたマクロを作成するには、プログラムエディタを使用します。

マクロで生成された DATA ステップのデバッグ

マクロを使って DATA ステップを生成できますが、マクロ生成された DATA ステップのデバッグは困難な場合があります。SAS ログはマクロのコピーは表示しますが、マクロが生成した DATA ステップは表示しません。このときに DEBUG オプションを使用すると、マクロが生成する文字列はデバッガーに連続するストリームとして認識されます。結果、実行が一時停止できるラインブレークがない状態になります。

マクロで生成される DATA ステップをデバッグするには、

1. プログラムの実行時に MPRINT と MFILE システムオプションを使用する。
2. 既存の外部ファイルにファイル参照名 MPRINT を割り当てる。MFILE でプログラムの出力を外部ファイルに誘導する。プログラムを再実行すると、ファイルに最新の出力が前回の出力に追加されます。
3. SAS セッションでマクロを起動する。
4. 外部ファイルを開くには、プログラムエディタウィンドウで INCLUDE コマンドを発行するか、ファイルメニューを使用します。

5. DATA ステートメントに DEBUG オプションを追加すると、デバッグセッションが開始します。
6. 論理エラーを発見したら、そのステートメントを生成する部分のマクロを修正します。

例

例 1: 結果が出力されないときのシンプルな DATA ステップのデバッグ問題の発見

問題の発見

このプログラムは旅行ツアーのグループの情報を作成します。このデータファイルには2種類のレコードが含まれています。1つはツアーコードで、もう1つは顧客情報です。プログラムは顧客のツアー番号、名前、年齢、性別のリストを作成します。

```
/* first execution */
data tours (drop=type);
input @1 type $ @;
if type='H' then do;
input @3 Tour $20.;
return;
end;
else if type='P' then do;
input @3 Name $10. Age 2. +1 Sex $1.;
output;
end;
datalines;
H Tour 101
P Mary E 21 F
P George S 45 M
P Susan K 3 F
H Tour 102
P Adelle S 79 M
P Walter P 55 M
P Fran I 63 F
;

proc print data=tours;
title 'Tour List';
run;
```

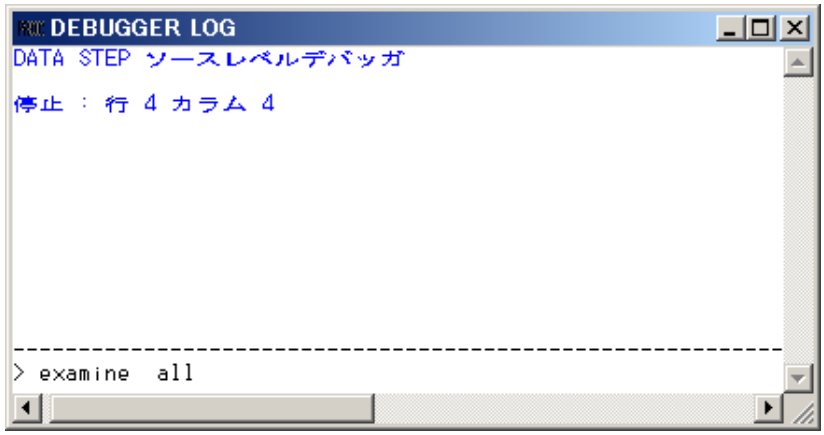
OBS	Tour	Name	Age	Sex
1		Mary E	21	F
2		George S	45	M
3		Susan K	3	F
4		Adelle S	79	M
5		Walter P	55	M
6		Fran I	63	F

プログラムエラーなしで実行されましたが、出力結果が予想外でした。出力結果には変数 Tour の値が含まれていません。SAS ログを見てもプログラムのデバッグはできません。なぜなら、データは有効でログにはエラーがないからです。論理エラーを確認するには、DATA ステップデバッガを使って DATA ステップを再度実行します。

最初の反復後にデータ値を検証する

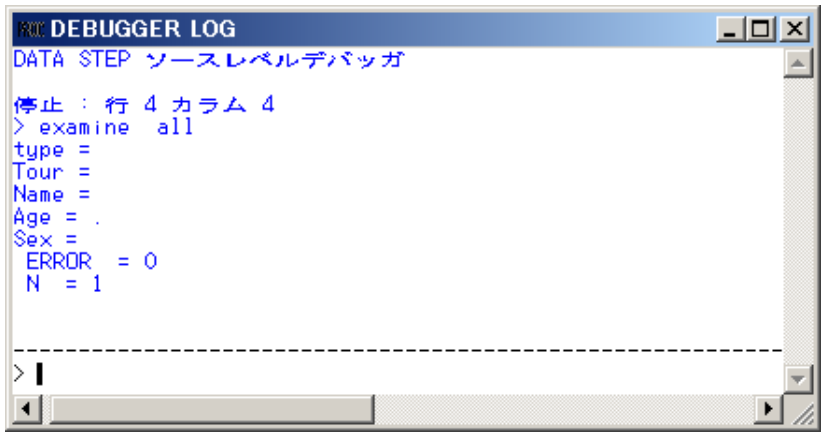
DATA ステップのデバッグは、論理エラーの仮説をたてて、プログラムの様々なポイントで変数値を検証して検定します。たとえば、実行を開始する前にデバッガコマンドラインから EXAMINE コマンドを発行して、プログラムデータ vector のすべての変数の値を表示するようにします。

```
examine _all_
```



注: ほとんどのデバッガコマンドには省略形があり、ファンクションキーにコマンドを割り当てることもできます。このセクションの例では、コマンドを完全形で表示します。すべてのコマンドのリストは、“カテゴリ別の DATA ステップデバッガコマンド” (27 ページ)を参照してください。

ENTER キーを押すと、次が表示されます。



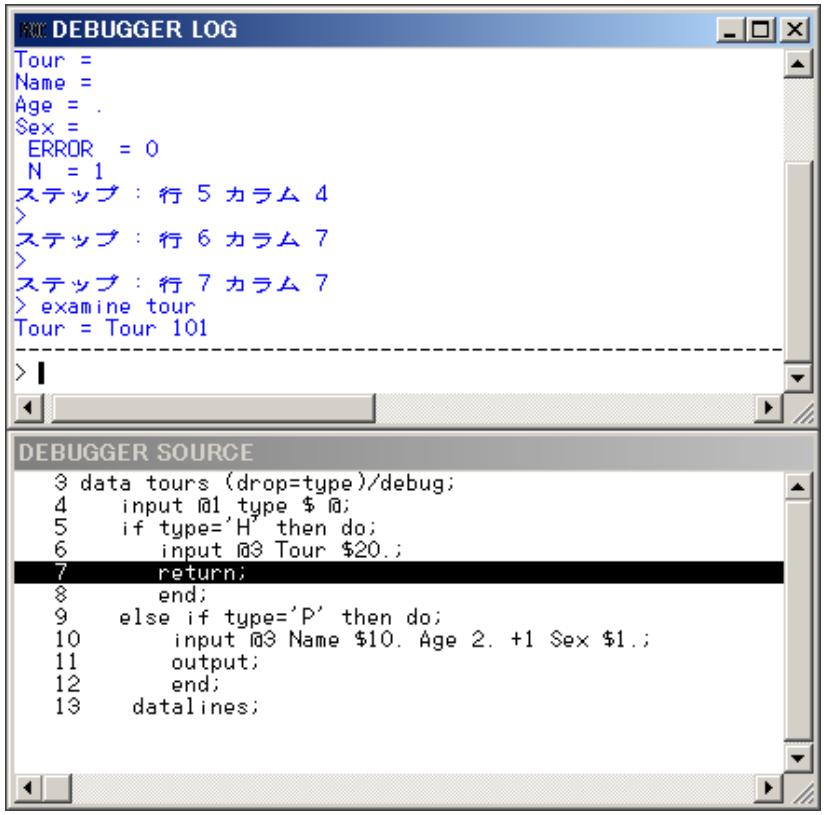
すべての変数の値は、DEBUGGER LOG ウィンドウに表示されます。SAS は INPUT ステートメントをコンパイルしましたが、実行はしていません。

DATA ステップステートメントを1つずつ実行するには、STEP コマンドを使用します。デフォルトで STEP コマンドは ENTER キーに割り当てられています。ENTER キーを繰り返し押して、DATA ステップの最初の反復を実行し、DEBUGGER SOURCE ウィンドウでプログラムの RETURN ステートメントがハイライトしたら、ストップします。

Tour の情報がプログラムの出力から欠損していたため、EXAMINE コマンドを入力して DATA ステップの最初の反復での変数 Tour の値を表示します。

```
examine tour
```

結果は次のように表示されます。



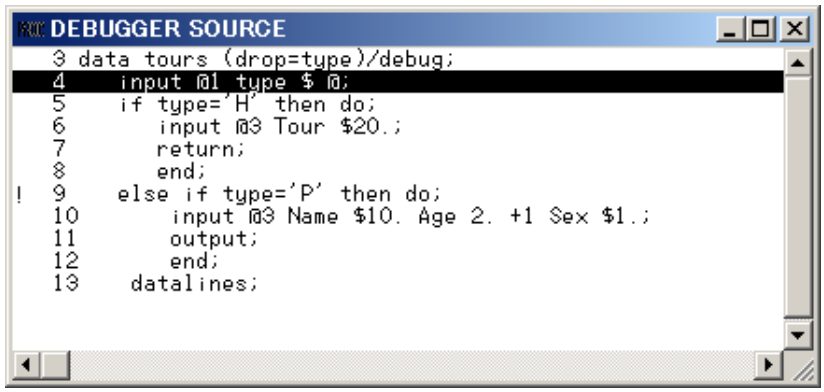
変数 Tour は値 Tour 101 を持っており、Tour が読み取られていることを示します。DATA ステップの最初の反復は、意図されたように動作しています。ENTER を押して DATA ステップの一番上にいきます。

2回目の反復後にデータ値を検証する

指定する特定の行で Data ステップの実行を一時停止するには、BREAK コマンド(ブレークポイントの設定ともいいます)を使います。この例では、ブレークポイントを 9 行目に設定するとこにより、ELSE ステートメントを実行する前に一時停止します。

```
break 9
```

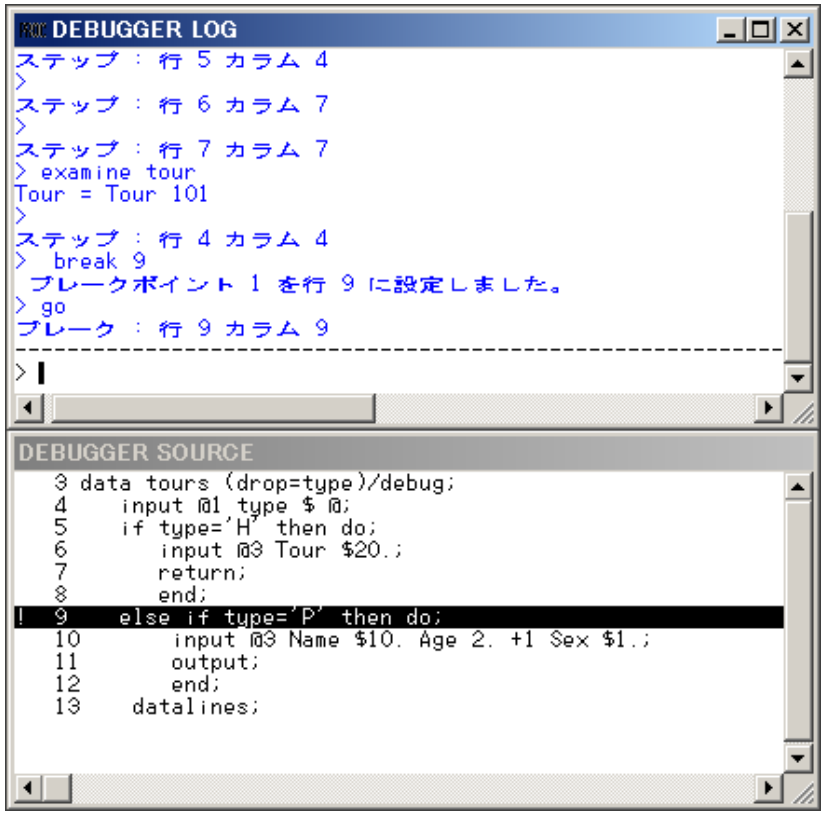
ENTER を押すと、ブレークポイントを示す感嘆符が DEBUGGER SOURCE ウィンドウの 9 行目に表示されます。



GO コマンドを実行すると、ブレークポイント(このケースでは 9 行目)に到達するまで DATA ステップが実行されます。

```
go
```

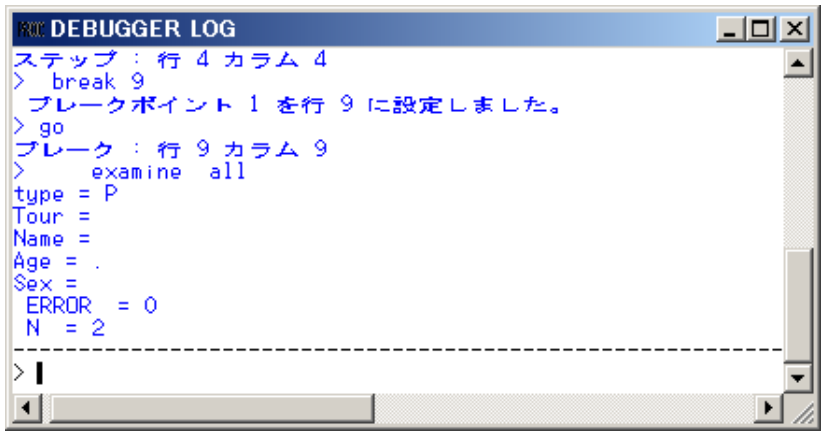
結果は次のように表示されます。



SAS は 7 行目の ELSE ステートメントの直前で実行を一時停止します。この時点で、ステータスを見るためにすべての変数の値を検証します。

```
examine _all_
```

値は次のように表示されます。



Tour の値をみようとしますが、表示されていません。各反復の最初にプログラムデータ vector は欠損値にリセットされるので、Tour の値を保持していません。論理的な問題を解決するためには、SAS プログラムに RETAIN ステートメントを含めなくてはなりません。

デバッガの終了

デバッグセッションを終了するには、デバッガコマンドラインで QUIT コマンドを発行します。

```
quit
```

デバッグウィンドウが消えて、元の SAS セッションが再開されます。

DATA ステップの修正

RETAIN ステートメントを追加して、元のプログラムを修正します。DATA ステップから DEBUG オプションを削除して、プログラムを再度サブミットします。

```
 /* corrected version */
data tours (drop=type);
retain Tour;
input @1 type $ @;
if type='H' then do;
input @3 Tour $20.;
return;
end;
else if type='P' then do;
input @3 Name $10. Age 2. +1 Sex $1.;
output;
end;
datalines;
H Tour 101
P Mary E 21 F
P George S 45 M
P Susan K 3 F
H Tour 102
P Adelle S 79 M
P Walter P 55 M
P Fran I 63 F
;

run;

proc print;
title 'Tour List';
run;
```

今度は Tour の値が出力に表示されています。

Tour List

OBS	Tour	Name	Age	Sex
1	Tour 101	Mary E	21	F
2	Tour 101	George S	45	M
3	Tour 101	Susan K	3	F
4	Tour 102	Adelle S	79	M
5	Tour 102	Walter P	55	M
6	Tour 102	Fran I	63	F

例 2: 出力形式の処理

この例では、Format ステートメントを使用して日付を出力するプログラムのデバッグ方法を示します。次のプログラムは特定の国への旅行日数をリストするレポートを作成します。

```
data tours;
length Country $ 10;
input Country $10. Start : mmddyy. End : mmddyy.;
Duration=end-start;
datalines;
Italy 033012 041312
Brazil 021912 022812
Japan 052212 061512
Venezuela 110312 11801
Australia 122112 011513
;

proc print data=tours;
format start end date9.;
title 'Tour Duration';
run;
```

結果ビューア - SAS Output

Tour Duration

OBS	Country	Start	End	Duration
1	Italy	30MAR2012	13APR2012	14
2	Brazil	19FEB2012	28FEB2012	9
3	Japan	22MAY2012	15JUN2012	24
4	Venezuela	03NOV2012	18JAN2012	-290
5	Australia	21DEC2012	15JAN2013	25

Venezuela へのツアーの Duration の値は-290 日という負の数字を示しています。エラーを確認するには、DATA ステップデバッガを使って DATA ステップを再度実行します。SAS は次のデバッガウィンドウを表示します。

実行を開始する前に DEBUGGER LOG コマンドラインから EXAMINE コマンドを発行して、プログラムデータ vector のすべての変数の値を表示するようにします。

```
examine _all_
```

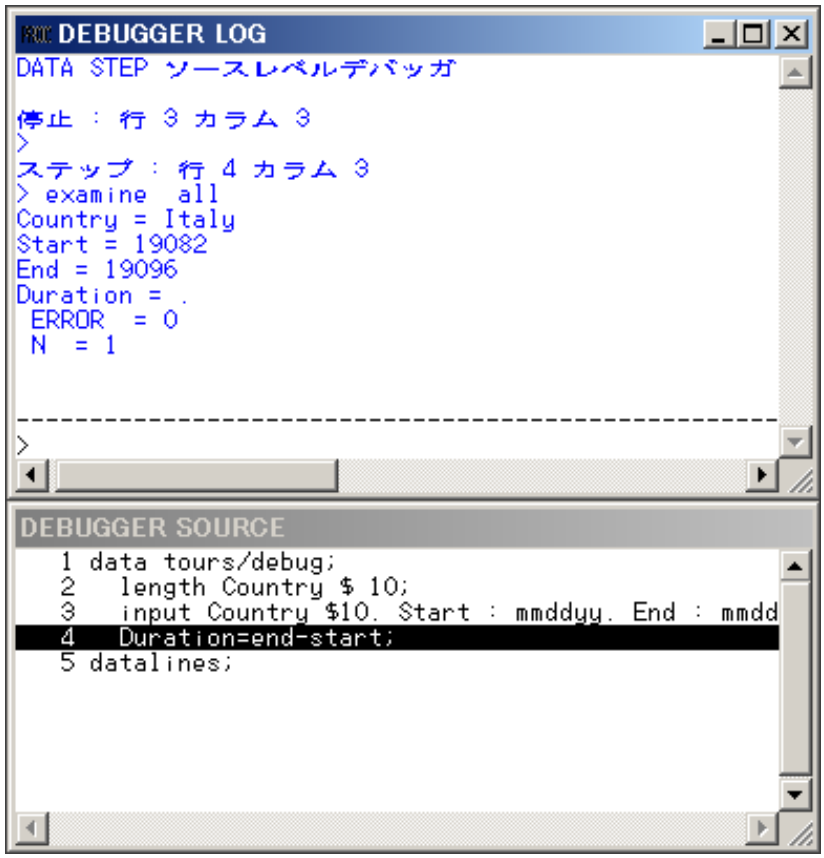
すべての変数の初期値は、DEBUGGER LOG ウィンドウに表示されます。SAS はまだ INPUT ステートメントを実行していません。

ENTER キーを押すと STEP コマンドが発行されます。SAS は INPUT ステートメントを実行し、割り当てステートメントをハイライトします。

EXAMINE コマンドを発行して、すべての変数の現在の値を表示するようにします。

```
examine _all_
```

結果は次のように表示されます。



Venezuela ツアーに問題があるので、Country の値が Venezuela のときに割り当てステートメントの前で実行を一時停止します。次のようにブレークポイントを設定します。

```
break 4 when country='Venezuela'
```

GO コマンドを実行して、プログラムの実行を再開します。

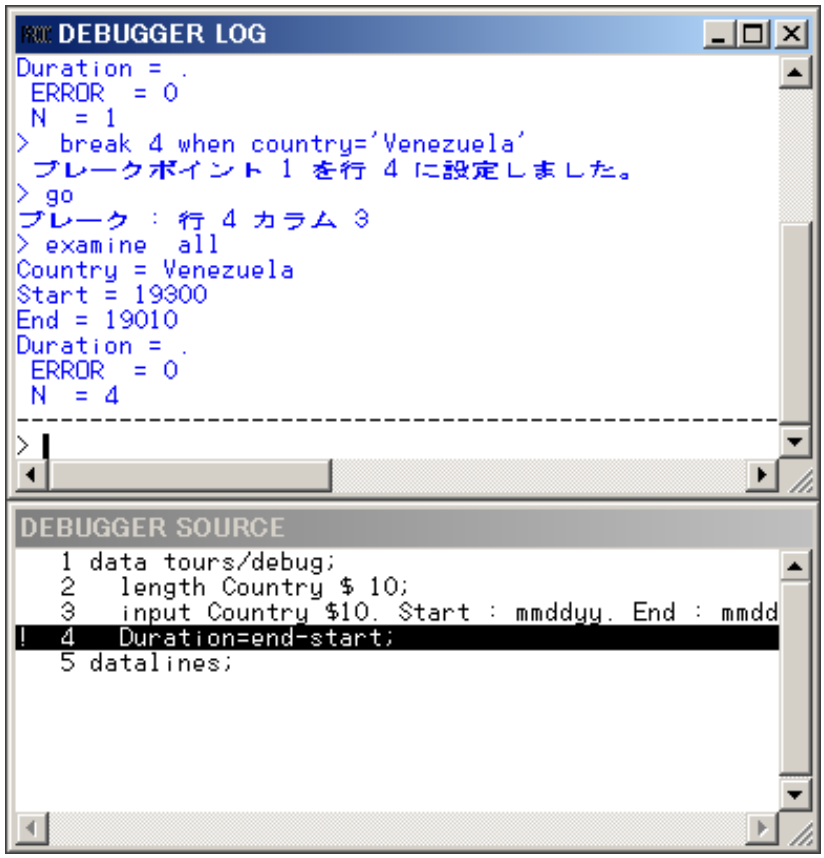
```
go
```

国名が Venezuela のとき、実行は停止します。Venezuela 旅行のツアー Start と End 日程を検証します。割り当てステートメントはハイライトされているため(ステートメントはまだ実行されていないということ)、Duration に値はありません。

EXAMINE コマンドを実行して、実行後の変数の値の変化を表示します。command to view the value of the variables after execution:

```
examine _all_
```

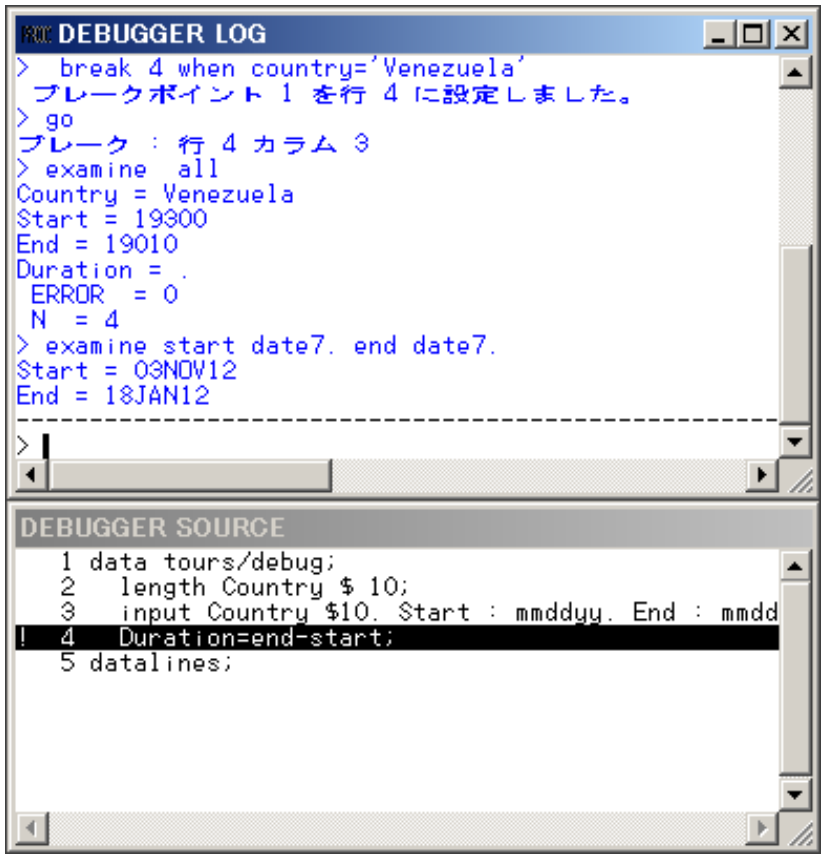
結果は次のように表示されます。



SAS 形式の日付を表示するには、DATEw. format を使用して EXAMINE コマンドを発行します。

```
examine start date7. end date7.
```

結果は次のように表示されます。



ツアー終了が January 18, 2012 ではなく November 18, 2012 となっているため、変数 End にエラーがあります。プログラムのソースデータを検証すると、End の値のミスタイプが見つかります。SET コマンドを使用して、暫定的に End の値を November 18 に設定して、期待する結果が得られるか見てみます。DDMMMYY*w*. 形式で SET コマンドを発行します。

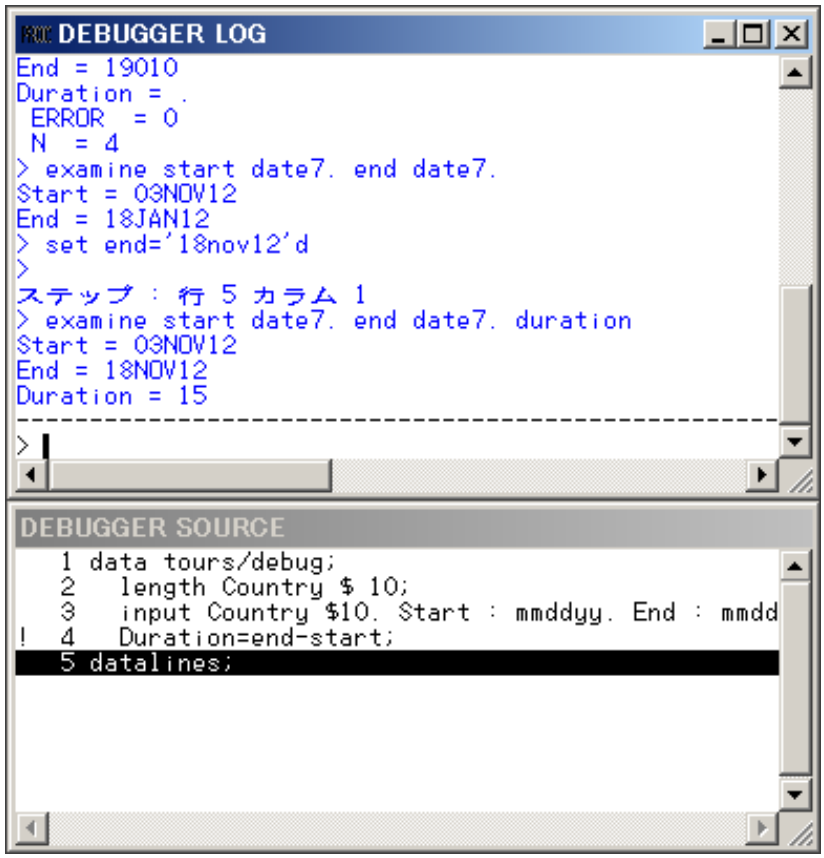
```
set end='18nov00'd
```

ENTER キーを押すと STEP コマンドが発行され、割り当てステートメントが実行されます。

EXAMINE コマンドを発行して、ツアー日付と Duration フィールドを表示します。

```
examine start date7. end date7. duration
```

結果は次のように表示されます。



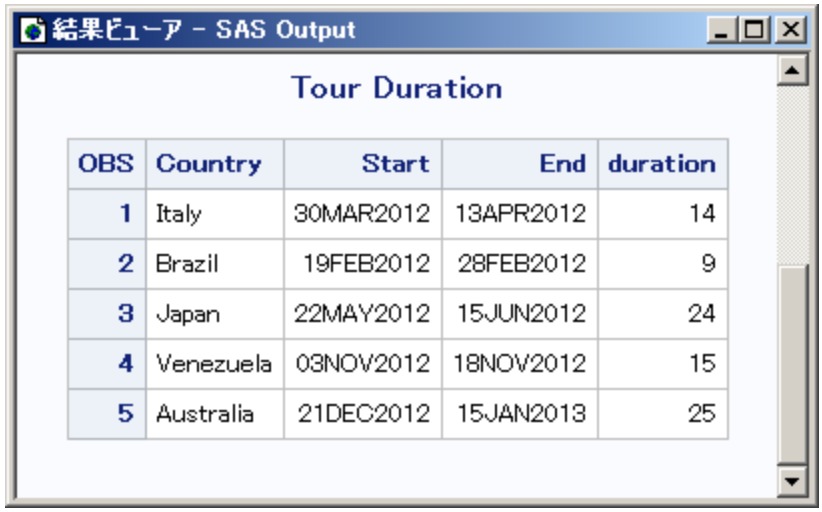
Start と End と Duration フィールドには正しいデータが入っています。

DEBUGGER LOG コマンドラインで QUIT コマンドを発行してデバッグセッションを終了します。SAS プログラムの元のデータを修正して、DBBUG オプションを削除して、プログラムを再度サブミットします。

```
/* corrected version */

data tours;
length Country $ 10;
input Country $10. Start : mmddyy. End : mmddyy.;
duration=end-start;
datalines;
Italy 033012 041312
Brazil 021912 022812
Japan 052212 061512
Venezuela 110312 111812
Australia 122112 011513
;

proc print data=tours;
format start end date9.;
title 'Tour Duration';
run;
```


結果ビューア - SAS Output

Tour Duration

OBS	Country	Start	End	duration
1	Italy	30MAR2012	13APR2012	14
2	Brazil	19FEB2012	28FEB2012	9
3	Japan	22MAY2012	15JUN2012	24
4	Venezuela	03NOV2012	18NOV2012	15
5	Australia	21DEC2012	15JAN2013	25

例 3: DO ループのデバッグ

反復する DO や DO WHILE や DO UNTIL ステートメントは DATA ステップの一回の反復で何回も反復させることができます。DO ループをデバッグする際、BREAK コマンドで AFTER オプションを使うことでループの複数回の反復を検証することができます。AFTER オプションではブレークポイントに到達するまでにループ反復する回数を指定します。BREAK コマンドはプログラムの実行を一時停止します。たとえば、このデータセットで考えてみます。

```
data new / debug;
set old;
do i=1 to 20;
newtest=oldtest+i;
output;
end;
run;
```

割り当てステートメント(この例では 4 行目)に DO ループの 5 回の反復ごとにブレークポイントを設定するには、このコマンドを発行します。

```
break 4 after 5
```

GO コマンドを発行したら、デバッガは DO ループの反復 `i` の値が 5 か 10 か 15 か 20 の場合に、実行を一時停止します。

反復 DO ループでは、AFTER オプションにはループの反復の回数を丁度分割できる値を選択します。たとえば、この DATA ステップでは、は 5 で 20 を丁度分割できます。2 回目の反復では、`i` の値はまた 5、10、15、そして 20 となります。

もしも丁度分割できる値を選択しない場合(この例では 3)、AFTER オプションでの `i` の値が 3、6、9、12、15 および 18 のときにデバッガは一時停止します。DO ループの 2 回目の反復のときは、`i` の値は 1、4、7、10、13 および 16 になります。

例 4: 変数のフォーマット指定された値の検証

EXAMINE コマンドで値を表示するときには、SAS 形式またはユーザー指定の形式を使用できます。たとえば、変数 BEGIN は SAS 日付値を持っているとします。曜日と日付を表示するには、EXAMINE に WEEKDATE*w.* 形式を使用します。

```
examine begin weekdate17.
```

BEGIN の値が 033012 のとき、デバッガの表示は次のようになります。

```
Sun, Mar 30, 2012
```

他の例では、SIZE という名前に形式を作成することもできます。

```
proc format;
value size 1-5='small'
6-10='medium'
11-high='large';
run;
```

変数 STOCKNUM に形式 SIZE を適用する DATA ステップをデバッグするには、形式を EXAMINE と一緒に使います。

```
examine stocknum size.
```

たとえば、STOCKNUM の値が 7 のとき、デバッガの表示は次のようになります。

```
STOCKNUM = medium
```

3 章

DATA ステップデバッガコマンドのディクショナリ

カテゴリ別の DATA ステップデバッガコマンド

カテゴリ	言語要素	説明
DATA ステップ変数の操作	CALCULATE (p. 30)	デバッガ式を評価して、結果を表示
	DESCRIBE (p. 32)	変数の属性を表示します。
	EXAMINE (p. 33)	変数の値を表示します。
	SET (p. 38)	特定の変数に新たな値を割り当てます。
ウィンドウの管理	HELP (p. 35)	デバッガコマンドの情報を表示します。
	SWAP (p. 40)	SOURCE ウィンドウと LOG ウィンドウを切り替えます。
デバッガのカスタマイズ	ENTER (p. 33)	ENTER キーにデバッガコマンドを割り当てます。

カテゴリ	言語要素	説明
デバッガの終了	QUIT (p. 38)	デバッガセッションを終了します。
デバッグリクエストの操作	BREAK (p. 28)	プログラムの実行を実行ステートメントで一時停止します。
	DELETE (p. 31)	ブレークポイントを削除または DATA ステップにおいて変数のステータスを監視
	LIST (p. 37)	引数にある項目のすべての出現個所を表示します。
	TRACE (p. 40)	デバッガが DATA ステップ実行の連続レコードを表示するかどうかを制御します。
	WATCH (p. 41)	指定した変数の値が変わると実行を一時停止します。
プログラム実行の制御	GO (p. 34)	DATA ステップの実行を開始または再開します。
	JUMP (p. 36)	一時停止したプログラムの実行を再開します。
	STEP (p. 39)	アクティブなプログラムでステートメントを1つずつ実行します。

ディクショナリ

BREAK

プログラムの実行を実行ステートメントで一時停止します。

カテゴリ: デバッグリクエストの操作

別名: B

構文

BREAK *location* <AFTER *count*> <WHEN *expression*> <DO *group* >

必須引数

location

ブレークポイントを設定する場所を指定します。*Location* は次のうちの1つになります。

label

ステートメントラベルです。ブレークポイントはラベルの次のステートメントに設定されます。

line-number

ブレークポイントを設定するプログラムの行番号です。

*

現在の行

オプション引数

AFTER *count*
ステートメントが *count* 回実行される毎に、ブレークポイントを有効にします。カウントは連続します。DO ループ内でステートメントに AFTER オプションが適用されると、カウントはループの反復から次の反復へと継続されていきます。デバッガは反復の開始前に *count* の値を 1 にリセットしません。

BREAK コマンドに AFTER と WHEN の両方がある場合、AFTER が最初に評価されます。AFTER カウントが満たされたら、WHEN 式が評価されます。

ヒント AFTER オプションは DO ループのデバッグに便利です。

WHEN *expression*
式が正しい場合に、ブレークポイントを有効にします。

DO *group*
DO ステートメントと END ステートメントに囲まれたデバッガコマンド群です。DO *group* の構文は以下になります。

DO; *command-1*<…;*command-n*;> **END**;

command
デバッガコマンドを指定します。複数コマンドはセミコロンで区切ります。

DO グループは2行以上になったり、IF-THEN/ELSE ステートメントを含んだりする場合があります。

IF *expression* **THEN** *command;* <ELSE *command;*>

IF *expression* **THEN DO** *group;* <ELSE DO *group;*>

IF は式を評価します。条件が真の場合、デバッガコマンドまたは THEN 句の DO グループが実行されます。条件が真ではない場合、オプションの ELSE コマンドが代替アクションを指示します。IF には次の引数を使用できます。

expression
デバッガ式を指定します。評価結果が非ゼロおよび非欠損の場合、この式は真になります。評価結果がゼロまたは欠損の場合、この式は偽になります。

command
デバッガコマンドを 1 つだけ指定します。

DO *group*
DO グループを指定します。

詳細

BREAK コマンドは指定のステートメントで DATA ステップの実行を一時停止します。BREAK コマンドの実行は*ブレークポイントの設定*とも言われます。

デバッガはブレークポイントを検知すると、次のことをします。

- AFTER *count* の値をチェックして、存在する場合、*count* がブレークポイントアクティベーションに到達しているならば、実行を一時停止
- WHEN 式を評価して、存在する場合は評価した条件が真ならば実行を一時停止
- AFTER も WHEN 句も存在しない場合は、実行を一時停止
- 実行が停止している場所の行番号を表示
- DO グループに存在するコマンドを実行
- >プロンプトでコントロールをユーザーに戻す

ブレークポイントが1つ以上のステートメントを含むソース行に設定されている場合、ブレークポイントはソース行の各ステートメントに適用となります。ブレークポイントがマクロ起動を含む行に設定されている場合、デバッガはマクロで生成される各ステートメントで一時停止します。

例

- 現在のプログラムの 5 行目にブレークポイントを設定

```
b 5
```

- ステートメントラベルが **eoflabel** のステートメントの後ろにブレークポイントを設定

```
b eoflabel
```

- 45 行目が 3 回実行される毎に有効となるブレークポイントを 45 行目に設定

```
b 45 after 3
```

- 45 行目が 3 回実行されて、かつ DIVISOR と DIVIDEND の値が 0 の場合に有効となるブレークポイントを 45 行目に設定

```
b 45 after 3
when (divisor=0 and dividend=0)
```

- プログラムの 45 行目にブレークポイントを設定して、変数 NAME と変数 AGE の値を評価

```
b 45 do; ex name age; end;
```

- プログラムの 15 行目にブレークポイントを設定 DIVISOR の値が 3 より大きい場合、STEP を実行そうでない場合は DIVIDEND の値を表示

```
b 15 do; if divisor>3 then st;
else ex dividend; end;
```

関連項目:

コマンド:

- “DELETE” (31 ページ)
- “WATCH” (41 ページ)

CALCULATE

デバッガ式を評価して、結果を表示

カテゴリ: DATA ステップ変数の操作

構文

CALC *expression*

必須引数

expression
デバッガ式を指定します。

制限事項 デバッガ式は関数を持てません。

詳細

CALCULATE コマンドはデバッガ式を評価して結果を表示します。結果は数値になります。

例

- 1.1 と 1.2 と 3.4 を足して合計に 0.5 をかける

```
calc (1.1+1.2+3.4)*0.5
```

- STARTAGE と DURATION を足す

```
calc startage+duration
```

- 変数 SALE の値から変数 DOWNPAY の値を引いて、その値に変数 RATE の値をかける その値を 12 で割って、50 を足す

```
calc (((sale-downpay)*rate)/12)+50
```

関連項目:

“式の処理” (11 ページ)

DELETE

ブレークポイントを削除または DATA ステップにおいて変数のステータスを監視

カテゴリ: デバッグリクエストの操作

別名: D

構文

DELETE BREAK *location*

DELETE WATCH *variable(s)* | _ALL_

必須引数

BREAK
ブレークポイントを削除

別名 B

location
削除したいブレークポイントの場所をを指定します。*location* には、次の値を指定できます。

ALL
DATA ステップにあるすべてのブレークポイント

label
ステートメントラベルの後のステートメント

line-number
プログラムの行番号

*
現在の行のブレークポイント

WATCH
ウォッチ対象変数のステータスを削除します

別名 W

variable(s)
ステータスが削除されるウォッチ対象変数の名前です

ALL
すべてのウォッチ対象変数のステータスを削除すると指定します

例

- ステートメントラベルでブレークポイントを削除します

```
eoflabel
:
d b eoflabel
```

- 現在の DATA ステップの変数 ABC のウォッチステータスを削除します

```
d w abc
```

関連項目:

コマンド:

- “BREAK” (28 ページ)
- “WATCH” (41 ページ)

DESCRIBE

変数の属性を表示します。

カテゴリ: DATA ステップ変数の操作

別名: DESC

構文

DESCRIBE *variable(s)* | _ALL_

必須引数

variable(s)
DATA ステップの変数を示します

ALL
DATA ステップに定義されているすべての変数を示します

詳細

DESCRIBE コマンドは指定した変数の属性を表示します（複数可）。

DESCRIBE は名前、タイプ、変数の長さ、そしてある場合は入力形式と出力形式または変数ラベルを表示します。

例

- 変数 ADDRESS の属性を表示します

```
desc address
```

- 配列エレメント ARR{*i* + *j*}の属性を表示します

```
desc arr{i+j}
```

ENTER

ENTER キーにデバッガコマンドを割り当てます。

カテゴリ: デバッガのカスタマイズ

構文

ENTER *command-1 <… ; command-n>*

必須引数

command
デバッガコマンドを指定します

デフォルト STEP 1

詳細

ENTER コマンドは ENTER キーにデバッガコマンドを割り当てます（複数可）。新しいコマンドを ENTER キーに割り当てると、既存のコマンド割り当ては置き換えられます。

複数のコマンドを割り当てる場合は、コマンドををセミコロンで区切ります。

例

- ENTER キーにコマンド STEP 5 を割り当てます

```
enter st 5
```

- ENTER キーに変数 CITY に対してコマンド EXAMINE とコマンド DESCRIBE をに割り当てます

```
enter ex city; desc city
```

EXAMINE

変数の値を表示します。

カテゴリ: DATA ステップ変数の操作

別名: E

構文

EXAMINE *variable-1 <format-1> <...variable-n <format-n> >*

EXAMINE _ALL_ *<format>*

必須引数

variable
DATA ステップ変数を示します。

ALL
現在の DATA ステップに定義されているすべての変数を示します。

オプション引数

format
SAS 形式かユーザーが作成した形式かを示します。

詳細

EXAMINE コマンドは指定した変数の値を表示します(複数可)。デバッガは変数に現在関連付けられている出力形式で値を表示します。

例

- 変数 N と変数 STR の値を表示します

```
ex n str
```

- 配列 TESTARR のエレメント *i* を表示します

```
ex testarr{i}
```

- 配列 CRR のエレメント *i*+1, *j**2、および *k*-3 を表示します

```
ex crr{i+1}; ex crr{j*2}; ex crr{k-3}
```

- SAS 日付変数 T_DATE を DATE7.出力形式で表示します

```
ex t_date date7.
```

- 配列 NEWARR のすべてのエレメントの値を表示します

```
ex newarr{*}
```

関連項目:

コマンド:

- “DESCRIBE” (32 ページ)

GO

DATA ステップの実行を開始または再開します。

カテゴリ: プログラム実行の制御

別名: G

構文

GO *<line-number | label>*

引数なし

引数を省略した場合、GO は DATA ステップの実行を再開し、ブレークポイントに到達するまで、またはウォッチ対象変数の値が変化するまで、または DATA ステップが実行を完了するまでステートメントを継続して実行します。

オプション引数

line-number
次に実行を一時停止させるプログラム行の番号を指定します

label
ステートメントラベルです実行は次のステートメントラベルの後で一時停止します。

詳細

GO コマンドは実行を開始または再開します。すべてのオブザベーションを読み込む、または GO コマンドに指定されたブレークポイントに到達する、または BREAK コマンドで以前に設定されたブレークポイントに到達するまで、実行は継続されます。

例

- プログラムの実行を再開して、ステートメントを継続して実行します。

```
g
```

- プログラムの実行を再開して、104 行でステートメントの実行を一時停止します

```
g 104
```

関連項目:

コマンド:

- “JUMP” (36 ページ)
- “STEP” (39 ページ)

HELP

デバッガコマンドの情報を表示します。

カテゴリ: ウィンドウの管理

構文

HELP

引数なし

HELP コマンドはデバッガコマンド要覧を表示します。コマンド名を選択すると、そのコマンドの構文や使用法の情報が表示されます HELP コマンドはウィンドウのコマンド行またはメニューまたはファンクションキーで入力しなければなりません。

JUMP

一時停止したプログラムの実行を再開します。

カテゴリ: プログラム実行の制御

別名: J

構文

JUMP *line-number* | *label*

必須引数

line-number
一時停止しているプログラムが再開するプログラム行の番号を示します。

label
ステートメントラベルです。実行はラベルの後のステートメントで再開されます。

詳細

JUMP コマンドは、途中のステートメントを実行することなく、特定の場所にジャンプしてプログラムを実行します。JUMP 実行後は、GO または STEP で実行を再開しなくてはなりません。DATA ステップのどの実行ステートメントにもジャンプすることができます。

注意:

JUMP コマンドを使用して DO ループ内のステートメントまたは LINK-RETURN グループの対象となっているラベルのステートメントにジャンプしないでください。 そのような場合、ループの最初や LINK ステートメントに設定されているコントロールを飛ばしてしまうことになり、予期しない結果が生じる原因となります。

JUMP は次の 2 つのケースで有益です

- 他の場所に集中して問題を起こしているコードの箇所を飛ばしたい場合このケースでは、JUMP コマンドを使って、DATA ステップで問題のある箇所の後ろのポイントに移動します。このケースでは、JUMP コマンドを使って、DATA ステップで問題のある箇所の後ろのポイントに移動します。
- 問題を引き起こした一連のステートメントを再実行したい場合このケースでは、JUMP を使って DATA ステップで問題あるステートメントの前の箇所に移動し、そして SET コマンドを使って関係のある変数の値を当時の値にリセットします。そしてそれらのステートメントを STEP または GO で再実行します。

例

- 5 行目にジャンプ

```
j 5
```

関連項目:

コマンド:

- “GO” (34 ページ)
- “STEP” (39 ページ)

LIST

引数にある項目のすべての出現個所を表示します。

カテゴリ: デバッグリクエストの操作

別名: L

構文

LIST _ALL_ | BREAK | DATASETS | FILES | INFILES | WATCH

必須引数

ALL
すべての項目の値を表示します。

BREAK
ブレークポイントの表示

別名 B

DATASETS
現在の DATA ステップで使用されている SAS データセットをすべて表示します。

FILES
現在の DATA ステップが書き込むすべての外部ファイルを表示します。

INFILES
現在の DATA ステップが読み込むすべての外部ファイルを表示します。

WATCH
ウォッチ対象の変数を表示します。

別名 W

例

- 全てのブレークポイント、SAS データセット、外部ファイル、そしてウォッチ対象の変数をリスト表示します。

```
l _all_
```

- 現在の DATA ステップのすべてのブレークポイントを表示します。

```
l b
```

関連項目:

コマンド:

- “BREAK” (28 ページ)
- “DELETE” (31 ページ)
- “WATCH” (41 ページ)

QUIT

デバッガセッションを終了します。

カテゴリ: デバッガの終了

別名: Q

構文

QUIT

引数なし

QUIT コマンドはデバッガセッションを終了させ、SAS セッションに戻ります。

詳細

SAS はデバッグしている DATA ステップによってビルドされたデータセットを作成します。しかし、デバッガを終了するのに QUIT を使用した場合、SAS は現在のオブザベーションを現在のデータセットに追加しません。

デバッガセッション中はいつでも QUIT コマンドを使用できます。デバッガセッションの終了後は、新しいデバッグセッションを開始するには DEBUG オプションを付けて DATA ステップを再サブミットしなければなりません。終了後にセッションを再開することはできません。

SET

特定の変数に新たな値を割り当てます。

カテゴリ: DATA ステップ変数の操作

別名: なし

構文

SET *variable=expression*

必須引数

variable

DATA ステップ変数名または配列参照名を指定します。

expression
デバッガ式を示します。

ヒント *expression* には、等号の左側で使用する変数名を指定します。1 つの変数が等号の両側に表示される場合、右側のオリジナル値を使用して式を評価し、等号の左側にある変数に結果が格納されます

詳細

SET コマンドは指定した変数に値を割り当てます。プログラム実行時にエラーを検知した場合、このコマンドを使って変数に新しい値を割り当てられます。これにより、デバッグセッションを継続できます。

例

- 変数 A に値 3 を設定します:

```
set a=3
```

- 変数 B に値 **12345** を割り当て、それを以前の B の値に連結させます:

```
set b='12345' || b
```

- 配列エレメント ARR{1}を式 a+3 の結果に設定します:

```
set arr{1}=a+3
```

- 配列エレメント CRR{1,2,3}を式 crr{1,1,2} + crr{1,1,3}の結果に設定します:

```
set crr{1,2,3} = crr{1,1,2} + crr{1,1,3}
```

- 変数 A を式 a+c*3 の結果に設定します:

```
set a=a+c*3
```

STEP

アクティブなプログラムでステートメントを1つずつ実行します。

カテゴリ: プログラム実行の制御

別名: ST

構文

STEP <*n*>

引数なし
STEP は 1 つのステートメントを実行します。

オプション引数

n
実行するステートメントの数を指定します。

詳細

STEP コマンドは DATA ステップのステートメントを、実行が一時停止していたステートメントから実行します。

STEP コマンドを発行すると、デバッガーは:

- 指定した数のステートメントを実行します。
- 行番号を表示します。
- >プロンプトを表示してコントロールをユーザーに戻します。

注: デフォルトで、ENTER キーを押すと STEP コマンドが実行できます。

関連項目:

コマンド:

- “GO” (34 ページ)
- “JUMP” (36 ページ)

SWAP

SOURCE ウィンドウと LOG ウィンドウを切り替えます。

カテゴリ: ウィンドウの管理

別名: なし

構文

SWAP

引数なし

SWAP コマンドは、デバッガの起動中に **LOG** ウィンドウと **SOURCE** ウィンドウを切り替えます。デバッグセッションを開始すると、デフォルトでは **LOG** ウィンドウがアクティブになります。DATA ステップの実行中に SWAP コマンドを使用して **SOURCE** ウィンドウと **LOG** ウィンドウの切り替えができ、プログラムの文字列をスクロールして表示したり、またプログラムの実行をモニタリングしたりできます。SWAP コマンドはウィンドウのコマンドライン、またはメニュー、またはファンクションキーから入力します。

TRACE

デバッガが DATA ステップ実行の連続レコードを表示するかどうかを制御します。

カテゴリ: デバッグリクエストの操作

別名: T

デフォルト: OFF

構文

TRACE <ON | OFF>

引数なし

引数なしで TRACE コマンドを使って、トレーシングがオンかオフかが確認できます。

オプション引数

ON
デバッガーが DATA ステップ実行時の連続レコードを表示するための準備を開始します。次の DATA ステップの実行を再開するステートメント(たとえば GO)で **DEBUGGER LOG** ウィンドウに DATA ステップ実行中のすべてのアクションを記録します。

OFF
表示を停止します。

比較

TRACE は TRACE コマンドの現在の状況を表示します。

例

- TRACE が ON か OFF か判別します:

```
trace
```

- デバッガ実行のレコードを表示する準備をします:

```
trace on
```

WATCH

指定した変数の値が変わると実行を一時停止します。

カテゴリ: デバッグリクエストの操作

別名: W

構文

WATCH *variable(s)*

必須引数

variable(s)
DATA ステップ変数を指定します(複数可)。

詳細

WATCH コマンドは指定した変数をモニターして、その値が変化したときにプログラムの実行を一時停止します。

ウォッチしている変数の値が変化する度に、デバッガは次のことをします。

- 実行を一時停止します
- 実行が一時停止している行番号を表示します。
- 変数の古い値を表示します。

- 変数の新しい値を表示します。
- >プロンプトを表示してコントロールをユーザーに戻します。

例

- 変数 DIVISOR の値の変化をモニターします。

```
w divisor
```

キーワード