THE POWER TO KNOW.

SAS® 9.4 コンポーネント オブジェクト

リファレンス

第 2 版

SAS® ドキュメント

The correct bibliographic citation for this manual is as follows: SAS Institute Inc. 2013. *SAS® 9.4 Component Objects: Reference, Second Edition*. Cary, NC: SAS Institute Inc.

SAS® 9.4 Component Objects: Reference, Second Edition

SAS Institute Inc., SAS Campus Drive, Cary, North Carolina 27513-2414.

August 2014

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

構文のコンポーネント

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。複数の引数を含む構文で区切り記号を使用する場合と使用しない場合を説明するために、引数の構文の形式が複数示されています。

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。

これらの例の SAS 構文では、キーワードには太字が使用されています。

CHAR (*string, position*)

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

ALTER (*alter-password*)

BEST *w.*

REMOVE <*data-set-name*>

この例では、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**

プロシジャステートメントによっては、ステートメント構文中に複数のキーワードが含まれます。

CREATE <UNIQUE> **INDEX** *index-name* **ON** *table-name* (*column-1* <, *column-2*, …>)

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、オプションの引数は山かっこ(< >)で囲まれます。

この例では、*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。この例の SAS コードでは、引数 *string* の値は 'summer'、引数 *position* の値は 4 です。

```
x=char('summer', 4);
```

この例では、*string* および *substring* は必須引数ですが、*modifiers* と *startpos* はオプションです。

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

argument(s)

引数は必ず 1 つ必要であり、複数の引数が許可されます。引数の間はスペースで区切ります。カンマ(,)などの区切り記号は、引数間に必要ありません。

たとえば、MISSING ステートメントは、この形式で複数の引数を含みます。

MISSING *character(s)*;

<LITERAL_ARGUMENT> *argument-1* <<LITERAL_ARGUMENT> *argument-2 ...* >

引数は必ず 1 つ必要であり、リテラル引数がこの引数に関連付けられます。リテラルと引数のペアは複数指定できます。リテラルと引数の間に区切り記号は必要ありません。省略記号(...)は、追加のリテラルと引数が許可されることを示します。

たとえば、BY ステートメントはこの引数を含みます。

BY <DESCENDING> *variable-1* <<DESCENDING> *variable-2* …>;

argument-1 <option(s)> <argument-2 <option(s)> ...>

引数は必ず 1 つ必要であり、1 つ以上のオプションがこの引数に関連付けられます。複数の引数と関連するオプションを指定できます。引数とオプションの間に区切り記号は必要ありません。省略記号(...)は、追加の引数と関連するオプションが許可されることを示します。

たとえば、FORMAT プロシジャの PICTURE ステートメントは、この形式で複数の引数を含みます。

PICTURE name <(*format-option(s)*)>
<*value-range-set-1* <(*picture-1-option(s)*)>
<*value-range-set-2* <(*picture-2-option(s)*)> …>>;

argument-1=value-1 <argument-2=value-2 ...>
引数には値を割り当てる必要があり、複数の引数を指定できます。省略記号(...)は、追加の引数が許可されることを示します。引数間に区切り記号は必要ありません。

たとえば、LABEL ステートメントは、この形式で複数の引数を含みます。

LABEL *variable-1=label-1 <variable-2=label-2 …>*;

argument-1 <, argument-2, ...>
引数は必ず 1 つ必要であり、カンマまたは別の区切り記号で区切って複数の引数を指定できます。省略記号(...)は、カンマで区切られた引数が続くことを示します。SAS ドキュメントでは両方の形式が使用されます。

次に、この形式で指定された複数の引数の例を示します。

AUTHPROVIDERDOMAIN (*provider-1:domain-1 <, provider-2:domain-2, …>*

INTO :*macro-variable-specification-1 <, :macro-variable-specification-2, …>*

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。この例では、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。

この 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。

- 非リテラル引数。この LINK ステートメントの例では、引数 *label* はユーザー指定値のため、斜体で表示されます。

  LINK *label*;

- 引数に割り当てられる非リテラル値。

  この 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(s) <format* > <DEFAULT = *default-format*>;

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。

この MAPS システムオプションの例では、等号により MAPS の値が設定されます。

MAPS = *location-of-maps*

< >
山かっこはオプションの引数を示します。必須引数は山かっこで囲みません。

この 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-2*, …>)

|
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。

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

CMPMODEL=BOTH | CATALOG | XML

...
省略記号は、引数の繰り返しが可能なことを示します。引数と省略記号が山かっこで囲まれている場合、その引数はオプションです。繰り返される引数には、その引数の前や後ろに、区切り記号を入れる必要があります。

この CAT 関数の例では、複数の *item* 引数が許可され、カンマで区切る必要があります。

CAT (*item-1* <, *item-2*, …>)

'*value*'または"*value*"
一重引用符や二重引用符付きの引数は、その値にも一重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。

この FOOTNOTE ステートメントの例では、引数 *text* に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。

この例では、各ステートメントが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への参照

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、参照の作成に 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* を引用符で囲んで使用します。

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

SAS 9.4Component Objects ユーティリティの新機能

ハッシュオブジェクトのキー集計のトラッキング

DECLARE ステートメントまたは _NEW_ 演算子で *keysum* 引数タグを使用して、すべてのキーのキー集計をトラッキングする変数の名前を指定することができます。

ハッシュオブジェクトで複数のキー間を反復

DO_OVER メソッドを DO ループの反復内で使用して、重複キー間を移動します。DO_OVER メソッドは最初のメソッド呼び出しでキーを読み込み、キーが最後に到達するまで重複キーリストを移動しつづけます。反復の途中でキーを切り替えた場合、RESET_DUP メソッドを使用してポインターをリストの最初にリセットしなければなりません。

ロックダウン状態での制限

LOCKDOWN ステートメントと LOCKDOWN システムオプションは SAS 9.4 メンテナンスリリース1で初めて導入されました。ロックダウンを設定すると、クライアントサーバー環境(例えば SAS Enterprise Guide)を使用中の場合、SAS サーバーの管理者は SAS クライアントがアクセスできるディレクトリとファイルのセットを作成できます。他のディレクトリやファイルはすべてアクセスできなくなります。SAS がロックダウン状態の場合で DATA ステップ Java オブジェクトが存在しない場合、ディレクトリやファイルの制限に加えて、いくつかの言語要素も使用できなくなります。詳細については、“SAS Processing Restrictions for Servers in a Locked-Down State” (*SAS Language Reference: Concepts*)を参照してください。

1 章

SAS コンポーネントオブジェクトについて

DATA ステップコンポーネントオブジェクト

SAS では、DATA ステップで使用するために事前定義された、次の 5 つのコンポーネントオブジェクトが用意されています。

ハッシュオブジェクトとハッシュ反復子オブジェクト
ルックアップキーに基づいて、データをすばやく効率的に保存、検索および取得できます。詳細については、“Using the Hash Object ” (*SAS Language Reference: Concepts*)および および“Using the Hash Iterator Object ” (*SAS Language Reference: Concepts*)を参照してください。

Java オブジェクト
Java クラスをインスタンス化して結果オブジェクトのフィールドとメソッドにアクセスする、Java ネイティブインターフェイス (JNI)に似たメカニズムを提供します。Java オブジェクトの詳細については、“Using the Java Object” (*SAS Language Reference: Concepts*) を参照してください。

ロガーおよびアペンダオブジェクト
ログイベントを記録し、それらのイベントを適切な出力先に書き込むことができます。詳細については、“Component Object Reference” (*SAS Logging: Configuration and Programming Reference*) を参照してください。

DATA ステップコンポーネントインターフェイス

DATA ステップコンポーネントオブジェクトインターフェイスでは、DATA ステップで事前定義済みコンポーネントオブジェクトを作成して操作できます。

コンポーネントオブジェクトを宣言して作成するには、DECLARE ステートメントのみを使用するか、DECLARE ステートメントと_NEW_演算子を組み合わせて使用します。

コンポーネントオブジェクトは、属性、メソッドおよび演算子で構成されるデータ要素です。**属性**は、オブジェクトに関連付けられた情報を指定するプロパティです。**メソッド**は、オブジェクトが実行できる操作を定義します。コンポーネントオブジェクトでは、**演算子**によって特殊な機能を使用できます。

コンポーネントオブジェクトの属性とメソッドにアクセスするには、DATA ステップオブジェクトのドット表記を使用します。

注: DATA ステップコンポーネントオブジェクトのステートメント、属性、メソッド、演算子は、これらのオブジェクト用に定義されている場合のみ使用できます。これらの事前定義された DATA ステップオブジェクトで SAS Component Language 機能を使用することはできません。

ドット表記と DATA ステップコンポーネントオブジェクト

定義

ドット表記は、メソッド呼び出しおよび属性値の設定とクエリに対するショートカットとして使用できます。ドット表記を使用することで、SAS プログラムが読みやすくなります。

DATA ステップコンポーネントオブジェクトでドット表記を使用するには、DECLARE ステートメントのみを使用するか、DECLARE ステートメントと_NEW_演算子を組み合わせて使用し、コンポーネントオブジェクトを宣言してインスタンス化する必要があります。詳細については、“Using DATA Step Component Objects” (*SAS Language Reference: Concepts*) および“Component Object Reference” (*SAS Logging: Configuration and Programming Reference*)を参照してください。

構文

ドット表記の構文を次に示します。

object.attribute

または

object.method(<*argument_tag-1*: *value-1*<,… *argument_tag-n*: *value-n*>>;

引数は次のように定義されます。

オブジェクト
DATA ステップコンポーネントオブジェクトの変数名を指定します。

attribute
割り当てまたはクエリ用のオブジェクト属性を指定します。

オブジェクトの属性を設定する場合、コードの形式は次のようになります。

```
object.attribute = value;
```

オブジェクトの属性をクエリする場合、コードの形式は次のようになります。

```
value = object.attribute;
```

method
呼び出すメソッド名を指定します。

argument_tag
メソッドに渡される引数を識別します。引数タグはかっこで囲みます。メソッドに引数タグが含まれるかどうかに関わらず、かっこは必須です。

すべての DATA ステップコンポーネントオブジェクトのメソッドの形式は次のようになります。

```
return_code=object.method(<argument_tag-1:value-1
<, ...argument_tag-n value-n>>);
```

リターンコードは、メソッドの成功または失敗を示します。ゼロのリターンコードは成功を表し、ゼロ以外の値は失敗を表します。メソッド呼び出しでリターンコード変数を指定せず、そのメソッドが失敗した場合、対応するエラーメッセージがログに出力されます。

value
引数の値を指定します。

コンポーネントオブジェクト使用時のルール

- DATA ステップ変数の割り当てと同じ方法でオブジェクトを割り当てることができます。ただし、オブジェクトの種類は一致する必要があります。最初のコードセットは有効ですが、2 番目のコードセットではエラーが発生します。

```
declare hash h();
declare hash t();
t=h;
```

```
declare hash t();
declare javaobj j();
j=t;
```

- オブジェクトの配列は宣言できません。次のコードではエラーが生成されます。

```
declare hash h1();
declare hash h2();
array h h1-h2;
```

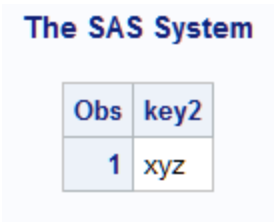
- コンポーネントオブジェクトはハッシュオブジェクト内にデータとして保存できますが、キーとしては保存できません。

```
data _null_;
declare hash h1();
declare hash h2();

length key1 key2 $20;

h1.defineKey('key1');
h1.defineData('key1', 'h2');
h1.defineDone();

key1 = 'abc';
h2 = _new_ hash();
h2.defineKey('key2');
h2.defineDone();

key2 = 'xyz';
h2.add();
```

```
h1.add();

key1 = 'def';
h2 = _new_ hash();
h2.defineKey('key2');
h2.defineDone();

key1 = 'abc';
rc = h1.find();
h2.output(dataset: 'work.h2');
run;

proc print data=work.h2;
run;
```

データセット WORK.H2 が表示されます。

図 1.1 *データセット WORK.H2*



The SAS System

Obs	key2
1	xyz

- 等号記号(=)以外の比較演算子を含むコンポーネントオブジェクトは使用できません。H1 と H2 がハッシュオブジェクトの場合、次のコードではエラーが生成されます。

```
if h1>h2 then
```

- コンポーネントオブジェクトを宣言してインスタンス化した後に、スカラ値を割り当てることはできません。J が Java オブジェクトの場合、次のコードではエラーが生成されます。

```
j=5;
```

- まだ使用されている可能性のあるオブジェクト参照、または参照によってすでに削除されているオブジェクト参照を削除しないように注意する必要があります。次のコードでは、元の H1 オブジェクトは H2 への参照によってすでに削除されているため、2 番目の DELETE ステートメントでエラーが生成されます。元の H2 は直接参照できなくなります。

```
declare hash h1();
declare hash h2();
declare hash t();
t=h2;
h2=h1;
h2.delete();
t.delete();
```

- 引数タグの構文ではコンポーネントオブジェクトを使用できません。次の例では、ADD メソッドでの H2 ハッシュオブジェクトの使用でエラーが生成されます。

```
declare hash h2();
declare hash h();
h.add(key: h2);
h.add(key: 99, data: h2);
```

- Java による SAS ログへのテキスト出力の 1 バイト目でのパーセント文字(%)の使用は、SAS によって予約されています。Java テキスト行の 1 バイト目に%を出力する必要がある場合は、そのすぐ後に別のパーセントを追加してエスケープします(%%)。

2 章

ハッシュオブジェクトとハッシュ反復子オブジェクトのディクショナリ

ディクショナリ

ADD メソッド

特定のキーに関連付けられた指定データをハッシュオブジェクトに追加します。

適用対象: ハッシュオブジェクト

構文

rc=*object*.**ADD**(<<KEY: *keyvalue-1*>, …<KEY: *keyvalue-n*>, <DATA: *datavalue-1* >, … <DATA: *datavalue-n*>>);

引数

rc
メソッドが成功したか失敗したかを示します。

ゼロのリターンコードは成功を表し、ゼロ以外の値は失敗を表します。メソッド呼び出しでリターンコード変数を指定せず、そのメソッドが失敗した場合、該当するエラーメッセージがログに書き込まれます。

オブジェクト
ハッシュオブジェクト名を指定します。

KEY: *keyvalue*
DEFINEKEY メソッド呼び出しで指定された、対応するキー変数に一致する型のキー値を指定します。

"KEY: *keyvalue*"ペアの数は、DEFINEKEY メソッドを使用して定義されたキー変数の数によって変わります。

DATA: *datavalue*
DEFINEDATA メソッド呼び出しで指定された、対応するデータ変数に一致する型のデータ値を指定します。

"DATA: *datavalue*"ペアの数は、DEFINEDATA メソッドを使用して定義されたデータ変数の数によって変わります。

詳細

次の 2 つの方法のいずれかで ADD メソッドを使用して、ハッシュオブジェクト内にデータを保存できます。

1 つ目は、次のコードに示すように、キーおよびデータ項目を定義してから ADD メソッドを使用する方法です。

```
data _null_;
length k $8;
length d $12;
/* Declare hash object and key and data variable names */
if _N_ = 1 then do;
declare hash h();
rc = h.defineKey('k');
rc = h.defineData('d');
rc = h.defineDone();
end;
/* Define constant key and data values */
k = 'Joyce';
d = 'Ulysses';
/* Add key and data values to hash object */
rc = h.add();
run;
```

2 つ目は、次のコードに示すように、ショートカットを使用して、ADD メソッド呼び出しでキーおよびデータを直接指定する方法です。

```
data _null_;
length k $8;
length d $12;
/* Define hash object and key and data variable names */
if _N_ = 1 then do;
declare hash h();
rc = h.defineKey('k');
rc = h.defineData('d');
rc = h.defineDone();
/* avoid uninitialized variable notes */
call missing(k, d);
end;
/* Define constant key and data values and add to hash object */
rc = h.add(key: 'Joyce', data: 'Ulysses');
run;
```

すでにハッシュオブジェクト内にあるキーを追加した場合、ADD メソッドはゼロ以外の値を返し、キーがすでにハッシュオブジェクト内にあることを示します。指定したキーに関連付けられたデータを新しいデータに置き換えるには、REPLACE メソッドを使用します。

DEFINEDATA メソッドでデータ変数を指定しない場合、データ変数は自動的にキーと同じであるとみなされます。

KEY:および DATA:引数タグを使用して直接キーとデータを指定する場合、両方の引数タグを使用する必要があります。

ADD メソッドでは、データ変数の値はデータ項目の値に設定されません。単にハッシュオブジェクトの値のみが設定されます。

関連項目:

- “Storing and Retrieving Data” (*SAS Language Reference: Concepts*)

メソッド:

- “DEFINEDATA メソッド” (19 ページ)
- “DEFINEKEY メソッド” (22 ページ)
- “REF メソッド” (50 ページ)

CHECK メソッド

指定したキーがハッシュオブジェクト内に保存されているかどうかを確認します。

適用対象: ハッシュオブジェクト

構文

rc=*object*.**CHECK**(<KEY: *keyvalue-1*, … KEY: *keyvalue-n*>);

引数

rc
メソッドが成功したか失敗したかを示します。

ゼロのリターンコードは成功を表し、ゼロ以外の値は失敗を表します。メソッド呼び出しでリターンコード変数を指定せず、そのメソッドが失敗した場合、該当するエラーメッセージがログに書き込まれます。

オブジェクト
ハッシュオブジェクト名を指定します。

KEY: *keyvalue*
DEFINEKEY メソッド呼び出しで指定された、対応するキー変数に一致する型のキー値を指定します。

"KEY: *keyvalue*"ペアの数は、DEFINEKEY メソッドを使用して定義されたキー変数の数によって変わります。

詳細

次の 2 つの方法のいずれかで CHECK メソッドを使用して、ハッシュオブジェクト内のデータを検索できます。

1 つ目は、次のコードに示すように、キーを指定してから CHECK メソッドを使用する方法です。

```
data _null_;
length k $8;
length d $12;
/* Declare hash object and key and data variable names */
if _N_ = 1 then do;
declare hash h();
rc = h.defineKey('k');
rc = h.defineData('d');
rc = h.defineDone();

/* avoid uninitialized variable notes */
call missing(k, d);
end;
/* Define constant key and data values and add to hash object */
rc = h.add(key: 'Joyce', data: 'Ulysses');
/* Verify that JOYCE key is in hash object */
k = 'Joyce';
rc = h.check();
if (rc = 0) then
put 'Key is in the hash object.';
run;
```

2 つ目は、次のコードに示すように、ショートカットを使用して、CHECK メソッド呼び出しでキーを直接指定する方法です。

```
data _null_;
length k $8;
length d $12;
/* Declare hash object and key and data variable names */
if _N_ = 1 then do;
declare hash h();
rc = h.defineKey('k');
rc = h.defineData('d');
rc = h.defineDone();

/* avoid uninitialized variable notes */
call missing(k, d);
end;
```

```
/* Define constant key and data values and add to hash object */
rc = h.add(key: 'Joyce', data: 'Ulysses');
/* Verify that JOYCE key is in hash object */
rc = h.check(key: 'Joyce');
if (rc =0) then
put 'Key is in the hash object.';
run;
```

比較

CHECK メソッドは、キーがハッシュオブジェクト内にあるかどうかを示す値のみを返します。キーに関連付けられたデータ変数は更新されません。FIND メソッドも、キーがハッシュオブジェクト内にあるかどうかを示す値を返します。ただし、キーがハッシュオブジェクト内にある場合、FIND メソッドはさらにデータ変数にデータ項目の値を設定し、メソッド呼び出し後にそのデータ項目を使用できるようにします。

関連項目:

メソッド:

- “FIND メソッド” (26 ページ)
- “DEFINEKEY メソッド” (22 ページ)

CLEAR メソッド

ハッシュオブジェクトインスタンスを削除せずに、ハッシュオブジェクトからすべての項目を削除します。

適用対象: ハッシュオブジェクト

構文

rc=*object*.**CLEAR**();

引数

rc
メソッドが成功したか失敗したかを示します。

ゼロのリターンコードは成功を表し、ゼロ以外の値は失敗を表します。メソッド呼び出しでリターンコード変数を指定せず、そのメソッドが失敗した場合、該当するエラーメッセージがログに書き込まれます。

オブジェクト
ハッシュオブジェクト名を指定します。

詳細

CLEAR メソッドでは、既存のハッシュオブジェクトを削除して新しいオブジェクトを作成することなく、そのオブジェクトから項目を削除して再利用できます。ハッシュオブジェクトインスタンスを完全に削除する場合は、DELETE メソッドを使用します。

注: CLEAR メソッドは、DATA ステップ変数の値を変更しません。単にハッシュオブジェクトの値をクリアします。

例: ハッシュオブジェクトのクリア

次の例では、ハッシュオブジェクトを宣言し、ハッシュオブジェクト内の項目数を取得してから、ハッシュオブジェクトを削除せずにクリアします。

```
data mydata;
do i = 1 to 10000;
output;
end;
run;
data _null_;
length i 8;

/* Declares the hash object named MYHASH using the data set MyData. */
dcl hash myhash(dataset: 'mydata');
myhash.definekey('i');
myhash.definedone();
call missing (i);
/* Uses the NUM_ITEMS attribute, which returns the */
/* number of items in the hash object. */
n = myhash.num_items;
put n=;
/* Uses the CLEAR method to delete all items within MYHASH. */
rc = myhash.clear();
/* Writes the number of items in the log. */
n = myhash.num_items;
put n=;
run;
```

最初の PUT ステートメントは、ハッシュテーブル MYHASH をクリアする前にその項目数を書き込みます。

```
n=10000
```

2 番目の PUT ステートメントは、ハッシュテーブル MYHASH をクリアした後にその項目数を書き込みます。

```
n=0
```

関連項目:

メソッド:

- “DELETE メソッド、ハッシュオブジェクトとハッシュ反復子オブジェクト” (24 ページ)

DECLARE ステートメントのハッシュオブジェクトとハッシュ反復子オブジェクト

ハッシュオブジェクトまたはハッシュ反復子オブジェクトを宣言します。ハッシュオブジェクトまたはハッシュ反復子オブジェクトのインスタンスを作成し、データを初期化します。

該当要素: DATA ステップ

カテゴリ: アクション

種類: 実行可能

別名: DCL

適用対象: ハッシュオブジェクト, ハッシュ反復子オブジェクト

構文

形式 1: **DECLARE** *object object-reference*;

形式 2: **DECLARE** *object object-reference* <(*argument_tag-1*: *value-1*, …*argument_tag-n*: *value-n*)>;

引数

object
コンポーネントオブジェクトを指定します。次のいずれかの値を使用できます。

hash
ハッシュオブジェクトを指定します。ハッシュオブジェクトは、迅速なデータの保存および取得のメカニズムを提供します。ハッシュオブジェクトは、ルックアップキーに基づいてデータを保存および取得します。

参照項目 “Using the Hash Object ” (*SAS Language Reference: Concepts*)

hiter
ハッシュ反復子オブジェクトを指定します。ハッシュ反復子オブジェクトを使用すると、ハッシュオブジェクトのデータを正または逆のキー順序で取得できます。

参照項目 “Using the Hash Object ” (*SAS Language Reference: Concepts*)

object-reference
ハッシュオブジェクトまたはハッシュ反復子オブジェクトのオブジェクト参照名を指定します。

argument_tag*:*value
ハッシュオブジェクトのインスタンスの作成に使用する情報を指定します。

有効なハッシュオブジェクト引数および値タグには、次の 5 つがあります。

dataset: '*dataset_name* <(*datasetoption*)>'
ハッシュオブジェクトに読み込む SAS データセットの名前を指定します。

SAS データセットの名前には、リテラルまたは文字変数を使用できます。データセット名は一重引用符または二重引用符で囲む必要があります。マクロ変数は二重引用符で囲む必要があります。

DATASET 引数タグでハッシュオブジェクトを宣言するときに、SAS データセットオプションを使用できます。データセットオプションは、データセットオプションが表示される SAS データセットにのみ適用されるアクションを指定します。次の操作を実行できます。

- 変数名の変更
- 処理するオブザベーション番号に基づくオブザベーションのサブセットの選択
- WHERE オプションを使用したオブザベーションの選択
- ハッシュオブジェクトに読み込まれたデータセット、または OUTPUT メソッド呼び出しで指定されている出力データセットの変数の削除または保持
- データセットのパスワードの指定

次の構文が使用されます。

```
dcl hash h (dataset: 'x (where = (i > 10))');
```

SAS データセットオプションのリストについては、次を参照してください。*SAS データセットオプション: リファレンス*

注 データセットに重複するキーが含まれている場合、デフォルトでは、ハッシュオブジェクトの最初のインスタンスが保持され、後続のインスタンスは無視されます。重複するキーがある場合に、ハッシュオブジェクトの最後のインスタンスまたは SAS ログに書き込まれたエラーメッセージを保存するには、DUPLICATE 引数タグを使用します。

duplicate: '*option*'

ハッシュオブジェクトにデータセットを読み込むときに重複するキーを無視するかどうかを判断します。デフォルトでは、最初のキーが保存され、後続の重複はすべて無視されます。オプションには、次のいずれかの値を使用できます。

'replace' | 'r'
最後の重複するキーレコードを保存します。

'error' | 'e'
重複するキーが検出された場合に、ログにエラーを報告します。

REPLACE オプションを使用する次の例では、キー 620 には **brown**、キー 531 には **blue** が保存されます。デフォルトを使用する場合は、620 には **green**、531 には **yellow** が保存されます。

```
data table;
  input key data $;
  datalines;
  531 yellow
  620 green
  531 blue
  908 orange
  620 brown
  143 purple
 run;

data _null_;
length key 8 data $ 8;
if (_n_ = 1) then do;
    declare hash myhash(dataset: "table", duplicate: "r");
    rc = myhash.definekey('key');
    rc = myhash.definedata('data');
    myhash.definedone();
 end;
rc = myhash.output(dataset:"otable");
run;
```

hashexp: *n*

ハッシュオブジェクトの内部テーブルサイズです。ハッシュテーブルのサイズは 2^n です。

HASHEXP の値は、ハッシュテーブルサイズを作成する 2 の累乗指数として使用されます。たとえば、HASHEXP の値 4 は、ハッシュテーブルサイズ 2^4(つまり 16)と同じです。HASHEXP の最大値は 20 です。

ハッシュテーブルサイズは、保存可能な項目数とは等しくありません。ハッシュテーブルを、一連の'バケット'と考えます。ハッシュテーブルサイズ 16 には、16 個の'バケット'があります。各バケットに保持できる項目の数は無限です。ハッシュテーブルの効率は、項目をバケットにマップし、バケットから項目を取得するハッシュ関数の機能にあります。

ハッシュオブジェクトのルックアップルーチンの効率を最大にするには、ハッシュオブジェクトのデータ量に応じてハッシュテーブルサイズを指定する必要があります。最適な結果が得られるまで、さまざまな HASHEXP 値を試してみてください。たとえば、ハッシュオブジェクトに 100 万個の項目が含まれている場合、ハッシュテーブルサイズ 16(HASHEXP = 4)でも機能しますが、効率は良くありません。ハッシュテーブルサイズ 512 または 1024(HASHEXP = 9 または 10)を使用すると、最高の処理速度が得られます。

デフォルト 8。これは、ハッシュテーブルサイズ 2^8 (つまり 256)と同じです。

keysum:'*variable-name*'
すべてのキーのキー集計をトラッキングする変数の名前を指定します。キー集計はキーが何回 FIND メソッド呼び出しに参照されたかのカウントです。

注 キー集計は出力データセットにあります。

例 “例 5: 出力データセットにキー集計を追加する” (18 ページ)

ordered: '*option*'
ハッシュオブジェクトとハッシュ反復子オブジェクトを併用する場合、またはハッシュオブジェクト OUTPUT メソッドを使用する場合、データがキー値の順序で返されるかどうか、またはどのように返されるかを指定します。

option には、次のいずれかの値を使用できます。

'ascending' | 'a'
データはキー値の昇順で返されます。'**ascending**'の指定は、'**yes**'を指定することと同じです。

'descending' | 'd'
データはキー値の降順で返されます。

'YES' | 'Y'
データはキー値の昇順で返されます。'**yes**'の指定は、'**ascending**'を指定することと同じです。

'NO' | 'N'
データは未定義の順序で返されます。

デフォルト NO

ヒント 引数は二重引用符で囲むこともできます。

multidata: '*option*'
各キーに複数のデータ項目を許可するかどうかを指定します。

option には、次のいずれかの値を使用できます。

'YES' | 'Y'
各キーに複数のデータ項目が許可されます。

'NO' | 'N'
各キーにデータ項目が 1 つのみ許可されます。

デフォルト NO

ヒント 引数値は二重引用符で囲むこともできます。

参照項目 “Non-Unique Key and Data Pairs” (*SAS Language Reference: Concepts*)

suminc: '*variable-name*'
ハッシュオブジェクトキーの集計数を維持します。SUMINC 引数タグは、DATA ステップ変数を指定します。この変数には、合計増分が保持されます。合計増分はキーが参照されるたびにキー集計に追加される数です。

参照項目 “Maintaining Key Summaries” (*SAS Language Reference: Concepts*)

例 キー集計は、DATA ステップ変数の現在の値を使用して変更されます。

```
dcl hash myhash(suminc: 'count');
```

参照項目 “Initializing Hash Object Data Using a Constructor” (*SAS Language Reference: Concepts*)および“Declaring and Instantiating a Hash Iterator Object” (*SAS Language Reference: Concepts*)

詳細

基本

SAS プログラムで DATA ステップコンポーネントオブジェクトを使用するには、オブジェクトを宣言して作成(インスタンス化)する必要があります。DATA ステップコンポーネントインターフェイスは、DATA ステップ内から定義済みのコンポーネントオブジェクトにアクセスするメカニズムを提供します。

定義済みの DATA ステップコンポーネントオブジェクトの詳細については、“Using DATA Step Component Objects” (*SAS Language Reference: Concepts*)を参照してください。

ハッシュオブジェクトまたはハッシュ反復子オブジェクトの宣言(フォーム 1)

ハッシュオブジェクトまたはハッシュ反復子オブジェクトを宣言するには、DECLARE ステートメントを使用します。

```
declare hash h;
```

DECLARE ステートメントは、オブジェクト参照 H がハッシュオブジェクトであることを SAS に示します。

新しいハッシュオブジェクトまたはハッシュ反復子オブジェクトを宣言した後に、_NEW_ 演算子を使用してオブジェクトをインスタンス化します。たとえば、次のコード行では、_NEW_演算子でハッシュオブジェクトが作成され、オブジェクト参照 H に割り当てられます。

```
h = _new_ hash( );
```

DECLARE ステートメントを使用したハッシュオブジェクトまたはハッシュ反復子オブジェクトのインスタンス生成(フォーム 2)

DECLARE ステートメントと_NEW_演算子を使用してハッシュオブジェクトまたはハッシュ反復子オブジェクトを宣言し、インスタンス化する 2 ステップのプロセスの代わりに、DECLARE ステートメントを使用して、ハッシュオブジェクトまたはハッシュ反復子オブジェクトを 1 つのステップで宣言し、インスタンス化することができます。たとえば、次のコード行では、DECLARE ステートメントでハッシュオブジェクトが宣言およびインスタンス化され、オブジェクト参照 H に割り当てられます。

```
declare hash h( );
```

前述のコード行は、次のコードを使用するのと同等です。

```
declare hash h;
h = _new_ hash( );
```

*コンストラクタ*は、ハッシュオブジェクトをインスタンス化し、ハッシュオブジェクトデータを初期化するために使用できるメソッドです。たとえば、次のコード行では、DECLARE ステートメントでハッシュオブジェクトが宣言およびインスタンス化され、オブジェクト参照 H に割り当てられます。さらに、引数タグ HASHEXP を使用してハッシュテーブルサイズが値 16(2^4)に初期化されます。

```
declare hash h(hashexp: 4);
```

ハッシュオブジェクトの読み込み時に SAS データセットオプションを使用する

DATASET 引数タグでハッシュオブジェクトを宣言するときに、SAS データセットオプションを使用できます。データセットオプションは、データセットオプションが表示される SAS データセットにのみ適用されるアクションを指定します。次の操作を実行できます。

- 変数名の変更
- 処理するオブザベーション番号に基づくオブザベーションのサブセットの選択
- WHERE オプションを使用したオブザベーションの選択
- ハッシュオブジェクトに読み込まれたデータセット、または OUTPUT メソッド呼び出しで指定されている出力データセットの変数の削除または保持
- データセットのパスワードの指定

次の構文が使用されます。

```
dcl hash h(dataset: 'x (where = (i > 10))');
```

データセットオプションのその他の使用例については、“例 4: ハッシュオブジェクトの読み込み時に SAS データセットオプションを使用する” (18 ページ)を参照してください。データセットオプションのリストについては、*SAS データセットオプション: リファレンス*を参照してください。

比較

ハッシュオブジェクトまたはハッシュ反復子オブジェクトのインスタンスを宣言し、インスタンス化するには、DECLARE ステートメントと_NEW_演算子を使用するか、DECLARE ステートメントのみを使用します。

例

例 1: DECLARE ステートメントと_NEW_演算子を使用したハッシュオブジェクトの宣言とインスタンス生成

この例では、DECLARE ステートメントを使用してハッシュオブジェクトを宣言します。ハッシュオブジェクトのインスタンス化には_NEW_演算子が使用されます。

```
data _null_; length k $15; length d $15; if _N_ = 1 then do; /* Declare and instantiate
      myhash = _new_ hash( );/* Define key and data variables */ rc = myhash.defineKey('
```

例 2: DECLARE ステートメントを使用したハッシュオブジェクトの宣言とインスタンス生成

この例では、DECLARE ステートメントを使用してハッシュオブジェクトの宣言とインスタンス化を 1 つのステップで実行します。

```
data _null_; length k $15; length d $15; if _N_ = 1 then do; /* Declare and instantiate
```

例 3: ハッシュオブジェクトのインスタンスを生成してサイズ順に並べる

この例では、DECLARE ステートメントを使用してハッシュオブジェクトを宣言し、インスタンス化します。ハッシュテーブルサイズは 16(2^4)に設定されています。

```
data _null_; length k $15; length d $15; if _N_ = 1 then do; /* Declare and instantiate
```

例 4: ハッシュオブジェクトの読み込み時に SAS データセットオプションを使用する

次の例では、ハッシュオブジェクトを宣言するときにさまざまな SAS データセットオプションを使用します。

```
data x; retain j 999; do i = 1 to 20; output; end; run; /* Using the WHERE option.*/ dat
```

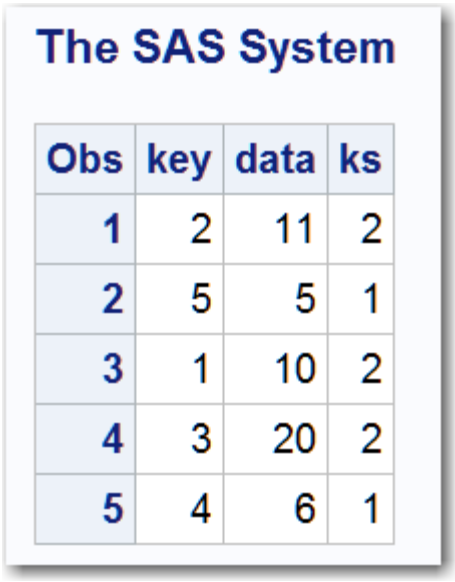
SAS データセットオプションのリストについては、*SAS データセットオプション: リファレンス*を参照してください。

例 5: 出力データセットにキー集計を追加する

次の例では、変数 *ks* はキー集計を持っていて、その変数を出力データセットに追加することを宣言します。

```
data key;
   length key data 8;
   input key data;
   datalines;
      1 10
      2 11
      3 20
      5 5
      4 6
run;

data _null_;
   length key data r i sum 8;
   length ks 8;
   i = 0;
   dcl hash h(dataset:'key', suminc: 'i', keysum: 'ks');
   h.definekey('key');
   h.definedata('key', 'data');
   h.definedone();

   i = 1;
   do key = 1 to 5;
      rc = h.find();
   end;

   do key = 1 to 3;
      rc = h.find();
   end;

   rc = h.output(dataset:'out');
 run;

proc print data=out;
run;
```

アウトプット 2.1 *キー集計データの出力*



The SAS System

Obs	key	data	ks
1	2	11	2
2	5	5	1
3	1	10	2
4	3	20	2
5	4	6	1

関連項目:

- “Using DATA Step Component Objects” (*SAS Language Reference: Concepts*)

演算子:

- “ _NEW_ Operator、ハッシュオブジェクトとハッシュ反復子オブジェクト” (38 ページ)

DEFINEDATA メソッド

ハッシュオブジェクトに保存するデータを、指定のデータ変数に関連付けて定義します。

適用対象: ハッシュオブジェクト

構文

rc=*object*.**DEFINEDATA**('*datavarname-1*'<, …'*datavarname-n*'>);

rc=*object*.**DEFINEDATA**(ALL: 'YES' | "YES");

引数

rc
メソッドが成功したか失敗したかを示します。

ゼロのリターンコードは成功を表し、ゼロ以外の値は失敗を表します。メソッド呼び出しでリターンコード変数を指定せず、そのメソッドが失敗した場合、該当するエラーメッセージがログに書き込まれます。

オブジェクト
ハッシュオブジェクト名を指定します。

'*datavarname*'
データ変数の名前を指定します。

データ変数名は二重引用符で囲むこともできます。

ALL: 'YES' | "YES"
すべてのデータ変数をデータに指定して、データセットをオブジェクトコンストラクタに読み込みます。

DECLARE ステートメントまたは_NEW_演算子で *dataset* 引数タグを使用してデータセットを自動的に読み込む場合、ALL: 'YES'を使ってすべてのデータセット変数をデータに指定できます。

詳細

ハッシュオブジェクトは、ルックアップキーに基づくデータの保存と取得によって機能します。キーとデータは、ドット表記のメソッド呼び出しでハッシュオブジェクトを初期化するときに使う DATA ステップ変数です。キーを定義するには、キー変数名を DEFINEKEY メソッドに渡します。データを定義するには、データ変数名を DEFINEDATA メソッドに渡します。ハッシュオブジェクトの初期化を完了するには、すべてのキー変数とデータ変数を定義してから、DEFINEDONE メソッドを呼び出す必要があります。キーとデータには、任意の数の文字または数値 DATA ステップ変数を使用できます。

注: ADD メソッドまたは REPLACE メソッドにショートカット表記(例: **`h.add(key:99, data:'apple', data:'orange')`**)を使い、DEFINEDATA メソッドに ALL:'YES'オプションを使う場合は、データの指定順序をデータセット内と同じにする必要があります。

注: ハッシュオブジェクトは値をキー変数(例: **`h.find(key:'abc')`**)に割り当てないため、ハッシュオブジェクトとハッシュ反復子が実行するキーとデータの変数割り当てを、SAS コンパイラからは検出できません。したがって、キー変数またはデータ変数への割り当てがプログラムに出現しない場合、変数が初期化されていないことを示すメモが発行されます。このようなメモが発行されないようにするには、次のアクションのいずれかを実行します。

- NONOTES システムオプションを設定します。
- 各キー変数およびデータ変数について、初期割り当てステートメントを(通常は欠損値に)指定します。
- CALL MISSING ルーチンを、すべてのキー変数とデータ変数をパラメータとして使用します。次に、例を示します。

```
length d $20;
length k $20;
if _N_ = 1 then do;
declare hash h();
rc = h.defineKey('k');
rc = h.defineData('d');
rc = h.defineDone();
call missing(k,d);
end;
```

DEFINEDATA メソッドの使用方法の詳細については、“Defining Keys and Data” (*SAS Language Reference: Concepts*)を参照してください。

例

次の例では、ハッシュオブジェクトを作成し、キー変数とデータ変数を定義します。

```
data _null_;
length d $20;
length k $20;
```

```
/* Declare the hash object and key and data variables */
if _N_ = 1 then do;
declare hash h();
rc = h.defineKey('k');
rc = h.defineData('d');
rc = h.defineDone();
/* avoid uninitialized variable notes */
call missing(k, d);
end;
run;
```

関連項目:

- “Defining Keys and Data” (*SAS Language Reference: Concepts*)

メソッド:

- “DEFINEDONE メソッド” (21 ページ)
- “DEFINEKEY メソッド” (22 ページ)

演算子:

- “_NEW_ Operator、ハッシュオブジェクトとハッシュ反復子オブジェクト” (38 ページ)

ステートメント:

- “DECLARE ステートメントのハッシュオブジェクトとハッシュ反復子オブジェクト” (12 ページ)

DEFINEDONE メソッド

キーとデータの定義がすべて完了したことを指定します。

適用対象: ハッシュオブジェクト

構文

rc= *object*.**DEFINEDONE**( );

rc= *object*.**DEFINEDONE**(MEMRC: 'y');

引数

rc
メソッドが成功したか失敗したかを示します。

ゼロのリターンコードは成功を表し、ゼロ以外の値は失敗を表します。メソッド呼び出しでリターンコード変数を指定せず、そのメソッドが失敗した場合、該当するエラーメッセージがログに書き込まれます。

オブジェクト
ハッシュオブジェクト名を指定します。

memrc:'y'
ハッシュオブジェクトへのデータセットの読み込みでメモリエラーが発生した場合に、エラーからの回復を可能にします。

データセット読み込み時のメモリ不足が原因で呼び出しが失敗すると、ゼロ以外のリターンコードが返されます。ハッシュオブジェクトは基礎配列の主メモリを解放します。この種のエラーが発生した後に実行可能な操作は、DELETE メソッドを使った削除のみです。

詳細

DEFINEDONE メソッド呼び出しでコンストラクタに *dataset* 引数タグを使用すると、データセットはハッシュオブジェクトに読み込まれます。

ハッシュオブジェクトは、ルックアップキーに基づくデータの保存と取得によって機能します。キーとデータは、ドット表記のメソッド呼び出しでハッシュオブジェクトを初期化するときに使う DATA ステップ変数です。キーを定義するには、キー変数名を DEFINEKEY メソッドに渡します。データを定義するには、データ変数名を DEFINEDATA メソッドに渡します。ハッシュオブジェクトの初期化を完了するには、すべてのキー変数とデータ変数を定義してから、DEFINEDONE メソッドを呼び出す必要があります。キーとデータには、任意の数の文字または数値 DATA ステップ変数を使用できます。

DEFINEDONE メソッドの使用方法の詳細については、“Defining Keys and Data” (*SAS Language Reference: Concepts*)を参照してください。

関連項目:

- “Defining Keys and Data” (*SAS Language Reference: Concepts*)

メソッド:

- “DEFINEDATA メソッド” (19 ページ)
- “DEFINEKEY メソッド” (22 ページ)

DEFINEKEY メソッド

ハッシュオブジェクトのキー変数を定義します。

適用対象: ハッシュオブジェクト

構文

rc=*object*.**DEFINEKEY**('*keyvarname-1*'<, … '*keyvarname-n*'>);
rc=*object*.**DEFINEKEY**(ALL: 'YES' | "YES");

引数

rc
メソッドが成功したか失敗したかを示します。

ゼロのリターンコードは成功を表し、ゼロ以外の値は失敗を表します。メソッド呼び出しでリターンコード変数を指定せず、そのメソッドが失敗した場合、該当するエラーメッセージがログに書き込まれます。

オブジェクト
ハッシュオブジェクト名を指定します。

'keyvarname'
キー変数の名前を指定します。

キー変数名は二重引用符で囲むこともできます。

ALL: 'YES' | "YES"
すべてのデータ変数をキーに指定して、データセットをオブジェクトコンストラクタに読み込みます。

DECLARE ステートメントまたは_NEW_演算子で *dataset* 引数タグを使用してデータセットを自動的に読み込む場合、ALL: 'YES'を使ってすべてのキー変数を定義できます。

詳細

ハッシュオブジェクトは、ルックアップキーに基づくデータの保存と取得によって機能します。キーとデータは、ドット表記のメソッド呼び出しでハッシュオブジェクトを初期化するときに使う DATA ステップ変数です。キーを定義するには、キー変数名を DEFINEKEY メソッドに渡します。データを定義するには、データ変数名を DEFINEDATA メソッドに渡します。ハッシュオブジェクトの初期化を完了するには、すべてのキー変数とデータ変数を定義してから、DEFINEDONE メソッドを呼び出す必要があります。キーとデータには、任意の数の文字または数値 DATA ステップ変数を使用できます。

DEFINEKEY メソッドの使用方法の詳細については、“Defining Keys and Data” (*SAS Language Reference: Concepts*)を参照してください。

注: ADD、CHECK、FIND、REMOVE、REPLACE などのメソッドにショートカット表記(例: **`h.add(key:99, data:'apple', data:'orange')`**)を使い、DEFINEKEY メソッドに ALL:'YES'オプションを使う場合は、キーとデータの指定順序をデータセット内と同じにする必要があります。

注: ハッシュオブジェクトは値をキー変数(例: **`h.find(key:'abc')`**)に割り当てないため、ハッシュオブジェクトとハッシュ反復子が実行するキーとデータの変数割り当てを、SAS コンパイラからは検出できません。したがって、キー変数またはデータ変数への割り当てがプログラムに出現しない場合、変数が初期化されていないことを示すメモが発行されます。このようなメモが発行されないようにするには、次のアクションのいずれかを実行します。

- NONOTES システムオプションを設定します。
- 各キー変数およびデータ変数について、初期割り当てステートメントを(通常は欠損値に)指定します。
- CALL MISSING ルーチンを、すべてのキー変数とデータ変数をパラメータとして使用します。次に、例を示します。

```
length d $20;
length k $20;
if _N_ = 1 then do;
declare hash h();
rc = h.defineKey('k');
rc = h.defineData('d');
rc = h.defineDone();
call missing(k, d);
end;
```

関連項目:

- “Defining Keys and Data” (*SAS Language Reference: Concepts*)

メソッド:

- “DEFINEDATA メソッド” (19 ページ)
- “DEFINEDONE メソッド” (21 ページ)

演算子:

- “_NEW_ Operator、ハッシュオブジェクトとハッシュ反復子オブジェクト” (38 ページ)

ステートメント:

- “DECLARE ステートメントのハッシュオブジェクトとハッシュ反復子オブジェクト” (12 ページ)

DELETE メソッド、ハッシュオブジェクトとハッシュ反復子オブジェクト

ハッシュオブジェクトまたはハッシュ反復子オブジェクトを削除します。

適用対象: ハッシュオブジェクト, ハッシュ反復子オブジェクト

構文

rc=*object*.**DELETE**( );

引数

rc
メソッドが成功したか失敗したかを示します。

ゼロのリターンコードは成功を表し、ゼロ以外の値は失敗を表します。メソッド呼び出しでリターンコード変数を指定せず、そのメソッドが失敗した場合、対応するエラーメッセージがログに出力されます。

object
ハッシュオブジェクトまたはハッシュ反復子オブジェクトの名前を指定します。

詳細

DATA ステップのコンポーネントオブジェクトは、DATA ステップの終了時に自動的に削除されます。他のハッシュオブジェクトコンストラクタまたはハッシュ反復子オブジェクトコンストラクタでオブジェクト参照変数を再利用する場合は、DELETE メソッドでハッシュオブジェクトまたはハッシュ反復子オブジェクトを削除する必要があります。

削除したハッシュオブジェクトまたはハッシュ反復子オブジェクトを使用しようとすると、エラーがログに書き込まれます。

ハッシュオブジェクト内からすべての項目を削除し、ハッシュオブジェクトを再利用できるように保存するには、“CLEAR メソッド” (11 ページ)を使用します。

EQUALS メソッド

2 つのハッシュオブジェクトが等しいかどうかを確認します。

適用対象: ハッシュオブジェクト

構文

rc=*object*.**EQUALS**(HASH: '*object*', RESULT: *variable name*);

引数

rc
メソッドが成功したか失敗したかを示します。

ゼロのリターンコードは成功を表し、ゼロ以外の値は失敗を表します。メソッド呼び出しでリターンコード変数を指定せず、そのメソッドが失敗した場合、該当するエラーメッセージがログに書き込まれます。

オブジェクト
ハッシュオブジェクト名を指定します。

HASH:'*object*'
1 つ目のハッシュオブジェクトに対して比較する 2 つ目のハッシュオブジェクトの名前を指定します。

RESULT: *variable name*
結果を保持する数値変数の名前を指定します。ハッシュオブジェクトが等しい場合、結果変数は 1 です。それ以外の場合の結果変数はゼロです。

詳細

次の例では、H1 を H2 ハッシュオブジェクトに対して比較します。

```
 length eq k 8;
declare hash h1();
h1.defineKey('k');
h1.defineDone();

declare hash h2();
h2.defineKey('k');
h2.defineDone();

rc = h1.equals(hash: 'h2', result: eq);
if eq then
put 'hash objects equal';
else
put 'hash objects not equal';
```

次のすべての条件に該当する場合、2 つのハッシュオブジェクトは等しいと定義されます。

- 2 つのハッシュオブジェクトが同サイズである(HASHEXP サイズが等しい)
- 2 つのハッシュオブジェクトに同数の項目がある(H1.NUM_ITEMS = H2.NUM_ITEMS)
- 2 つのハッシュオブジェクトのキー構造とデータ構造が同一である

- H1 と H2 ハッシュオブジェクトの比較を順不同で反復した場合に、H1 からの一連のレコードが対応する H2 のレコードと同じキーおよびデータのフィールドを持っている。すなわち、各ハッシュオブジェクトの同じ位置に各レコードがあり、それらのレコードが対応するもう一方のハッシュオブジェクトのレコードと同一である。

例: 2 つのハッシュオブジェクトの比較

次の例では、EQUALS の 1 つ目のリターンコードはゼロ以外の値を返し、2 つ目はゼロ値を返します。

```
data x;
length k eq 8;
declare hash h1();
h1.defineKey('k');
h1.defineDone();

declare hash h2();
h2.defineKey('k');
h2.defineDone();

k = 99;
h1.add();
h2.add();
rc = h1.equals(hash: 'h2', result: eq);
put eq=;

k = 100;
h2.replace();

rc = h1.equals(hash: 'h2', result: eq);
put eq=;

run;
```

FIND メソッド

指定したキーがハッシュオブジェクトに保存されているかどうかを確認します。

適用対象: ハッシュオブジェクト

構文

rc=*object*.**FIND**(<KEY: *keyvalue-1*, …KEY: *keyvalue-n*>);

引数

rc
メソッドが成功したか失敗したかを示します。

ゼロのリターンコードは成功を表し、ゼロ以外の値は失敗を表します。メソッド呼び出しでリターンコード変数を指定せず、そのメソッドが失敗した場合、該当するエラーメッセージがログに書き込まれます。

オブジェクト
ハッシュオブジェクト名を指定します。

KEY: *keyvalue*
DEFINEKEY メソッド呼び出しで指定された、対応するキー変数に一致する型のキー値を指定します。

"KEY: *keyvalue*"ペアの数は、DEFINEKEY メソッドを使用して定義されたキー変数の数によって変わります。

詳細

次の 2 つの方法のいずれかで FIND メソッドを使用して、ハッシュオブジェクト内のデータを検索できます。

1 つ目は、次のコードに示すように、キーを指定してから FIND メソッドを使用する方法です。

```
data _null_;
length k $8;
length d $12;
/* Declare hash object and key and data variables */
if _N_ = 1 then do;
declare hash h();
rc = h.defineKey('k');
rc = h.defineData('d');
rc = h.defineDone();
/* avoid uninitialized variable notes */
call missing(k, d);
end;
/* Define constant key and data values */
rc = h.add(key: 'Joyce', data: 'Ulysses');
/* Find the key JOYCE */
k = 'Joyce';
rc = h.find();
if (rc = 0) then
put 'Key is in the hash object.';
run;
```

2 つ目は、次のコードに示すように、ショートカットを使用して、FIND メソッド呼び出しでキーを直接指定する方法です。

```
data _null_;
length k $8;
length d $12;
/* Declare hash object and key and data variables */
if _N_ = 1 then do;
declare hash h();
rc = h.defineKey('k');
rc = h.defineData('d');
rc = h.defineDone();
/* avoid uninitialized variable notes */
call missing(k, d);
end;
/* Define constant key and data values */
rc = h.add(key: 'Joyce', data: 'Ulysses');
/* Find the key JOYCE */
rc = h.find(key: 'Joyce');
if (rc = 0) then
put 'Key is in the hash object.';
run;
```

ハッシュオブジェクトの各キーに複数のデータ項目がある場合は、“FIND_NEXT メソッド” (29 ページ)と“FIND_PREV メソッド” (30 ページ)を FIND メソッドと組み合わせ、複数データ項目のリスト内を移動します。

比較

FIND メソッドは、キーがハッシュオブジェクト内に存在するかどうかを示す値を返します。FIND メソッドでは、キーがハッシュオブジェクトに存在する場合、同時にデータ項目の値がデータ変数に設定されます。これによって、メソッド呼び出し後に値が使えるようになります。CHECK メソッドは、キーがハッシュオブジェクト内にあるかどうかを示す値のみを返します。データ変数は更新されません。

例: FIND メソッドを使用したハッシュオブジェクト内のキー検索

次の例では、ハッシュオブジェクトを作成します。2 つのデータ値を追加します。FIND メソッドを使用してハッシュオブジェクト内のキーを検索します。データ値は、キーに関連付けられたデータセット変数に返されます。

```
data _null_;
length k $8;
length d $12;
/* Declare hash object and key and data variable names */
if _N_ = 1 then do;
declare hash h();
rc = h.defineKey('k');
rc = h.defineData('d');
/* avoid uninitialized variable notes */
call missing(k, d);
rc = h.defineDone();
end;
/* Define constant key and data values and add to hash object */
rc = h.add(key: 'Joyce', data: 'Ulysses');
rc = h.add(key: 'Homer', data: 'Odyssey');
/* Verify that key JOYCE is in hash object and */
/* return its data value to the data set variable D */
rc = h.find(key: 'Joyce');
put d=;
run;
```

d=Ulysses が SAS ログに書き込まれます。

関連項目:

- “Storing and Retrieving Data” (*SAS Language Reference: Concepts*)

メソッド:

- “CHECK メソッド” (9 ページ)
- “DEFINEKEY メソッド” (22 ページ)
- “FIND_NEXT メソッド” (29 ページ)
- “FIND_PREV メソッド” (30 ページ)
- “REF メソッド” (50 ページ)

FIND_NEXT メソッド

現在のキーの複数項目リストで現在のリスト項目を次の項目に設定し、対応するデータ変数にデータを設定します。

適用対象: ハッシュオブジェクト

構文

rc=*object*.**FIND_NEXT**( );

引数

rc
メソッドが成功したか失敗したかを示します。

ゼロのリターンコードは成功を表し、ゼロ以外の値は失敗を表します。メソッド呼び出しでリターンコード変数を指定せず、そのメソッドが失敗した場合、対応するエラーメッセージがログに出力されます。

オブジェクト
ハッシュオブジェクト名を指定します。

詳細

FIND メソッドは、キーがハッシュオブジェクト内に存在するかどうかを判断します。HAS_NEXT メソッドは、キーに複数のデータ項目が関連付けられているかどうかを判断します。キーに別のデータ項目があることが判明した場合、FIND_NEXT メソッドを使用してそのデータ項目を取得できます。これによりデータ変数にデータ項目の値が設定され、メソッド呼び出し後に使用できるようになります。データ項目リスト内では、HAS_NEXT メソッドと FIND_NEXT メソッドを使用することでリスト内を移動できます。

例

この例では、FIND_NEXT メソッドを使用して、複数のキーに複数のデータ項目があるデータセットを反復処理します。キーに複数のデータ項目がある場合、後続の項目は **dup** とマークされます。

```
data dup;
length key data 8;
input key data;
datalines;
1 10
2 11
1 15
3 20
2 16
2 9
3 100
5 5
1 5
4 6
5 99
;
```

```
data _null_;
dcl hash h(dataset:'dup', multidata: 'y');
h.definekey('key');
h.definedata('key', 'data');
h.definedone();
/* avoid uninitialized variable notes */
call missing (key, data);
do key = 1 to 5;
rc = h.find();
if (rc = 0) then do;
put key= data=;
rc = h.find_next();
do while(rc = 0);
put 'dup ' key= data;
rc = h.find_next();
end;
end;
end;
run;
```

次の行が SAS ログに書き込まれます。

```
key=1 data=10
dup key=1 5
dup key=1 15
key=2 data=11
dup key=2 9
dup key=2 16
key=3 data=20
dup key=3 100
key=4 data=6
key=5 data=5
dup key=5 99
```

関連項目:

- “Non-Unique Key and Data Pairs” (*SAS Language Reference: Concepts*)

メソッド:

- “FIND メソッド” (26 ページ)
- “FIND_PREV メソッド” (30 ページ)
- “HAS_NEXT メソッド” (33 ページ)

FIND_PREV メソッド

現在のキーの複数項目リストで現在のリスト項目を前の項目に設定し、対応するデータ変数にデータを設定します。

適用対象: ハッシュオブジェクト

構文

rc=*object*.**FIND_PREV**( );

引数

rc
メソッドが成功したか失敗したかを示します。

ゼロのリターンコードは成功を表し、ゼロ以外の値は失敗を表します。メソッド呼び出しでリターンコード変数を指定せず、そのメソッドが失敗した場合、対応するエラーメッセージがログに出力されます。

オブジェクト
ハッシュオブジェクト名を指定します。

詳細

FIND メソッドは、キーがハッシュオブジェクト内に存在するかどうかを判断します。HAS_PREV メソッドは、キーに複数のデータ項目が関連付けられているかどうかを判断します。キーに前のデータ項目があることが判明した場合、FIND_PREV メソッドを使用してそのデータ項目を取得できます。これによりデータ変数にデータ項目の値が設定され、メソッド呼び出し後に使用できるようになります。データ項目リスト内では、HAS_NEXT メソッドと FIND_NEXT メソッドに加えて HAS_PREV メソッドと FIND_PREV メソッドを使用することで、リスト内を移動できます。例については、“HAS_NEXT メソッド” (33 ページ)を参照してください。

関連項目:

- “Non-Unique Key and Data Pairs” (*SAS Language Reference: Concepts*)

メソッド:

- “FIND メソッド” (26 ページ)
- “FIND_NEXT メソッド” (29 ページ)
- “HAS_PREV メソッド” (35 ページ)

FIRST メソッド

基となるハッシュオブジェクトの開始値を返します。

適用対象: ハッシュ反復子オブジェクト

構文

rc=*object*.**FIRST**( );

引数

rc
メソッドが成功したか失敗したかを示します。

ゼロのリターンコードは成功を表し、ゼロ以外の値は失敗を表します。メソッド呼び出しでリターンコード変数を指定せず、そのメソッドが失敗した場合、対応するエラーメッセージがログに書き込まれます。

オブジェクト
ハッシュ反復子オブジェクト名を指定します。

詳細

FIRST メソッドは、ハッシュオブジェクト内の最初のデータ項目を返します。ハッシュオブジェクトをインスタンス化するときに、DECLARE ステートメントまたは_NEW_演算子で **ordered: 'yes'**または **ordered: 'ascending'**引数タグを使用した場合、ハッシュオブジェクト内のデータ項目はキー値の昇順で並べ替えられるため、'最小'キー(最小の数値またはアルファベット順の最初)のデータ項目が返されます。NEXT メソッドへの繰り返される呼び出しは、ハッシュオブジェクト内を反復して移動し、キーの昇順でデータ項目を返します。反対に、ハッシュオブジェクトをインスタンス化するときに、DECLARE ステートメントまたは_NEW_演算子で **ordered: 'descending'**引数タグを使用した場合、ハッシュオブジェクト内のデータ項目はキー値の降順で並べ替えられるため、'最大'キー(最大の数値またはアルファベット順の最後)のデータ項目が返されます。NEXT メソッドへの繰り返される呼び出しは、ハッシュオブジェクト内を反復して移動し、キーの降順でデータ項目を返します。

ハッシュオブジェクト内の最後のデータ項目を返すには、LAST メソッドを使用します。

注: FIRST メソッドは、データ変数にデータ項目の値を設定し、メソッド呼び出し後にそのデータ項目を使用できるようにします。

例: ハッシュオブジェクトデータの取得

次の例では、売上データを含むデータセットを作成します。製品を売上順に表示します。データがハッシュオブジェクト内に読み込まれ、データの取得に FIRST メソッドと NEXT メソッドが使用されます。

```
data work.sales;
input prod $1-6 qty $9-14;
datalines;
banana 398487
apple 384223
orange 329559
;
data _null_;
/* Declare hash object and read SALES data set as ordered */
if _N_ = 1 then do;
length prod $10;
length qty $6;
declare hash h(dataset: 'work.sales', ordered: 'yes');
declare hiter iter('h');
/* Define key and data variables */
h.defineKey('qty');
h.defineData('prod');
h.defineDone();
/* avoid uninitialized variable notes */
call missing(qty, prod);
end;
/* Iterate through the hash object and output data values */
rc = iter.first();
do while (rc = 0);
put prod=;
rc = iter.next();
end;
run;
```

次の行が SAS ログに書き込まれます。

```
prod=orange
prod=apple
prod=banana
```

関連項目:

- “Using the Hash Iterator Object ” (*SAS Language Reference: Concepts*)

メソッド:

- “LAST メソッド” (37 ページ)

演算子:

- “_NEW_ Operator、ハッシュオブジェクトとハッシュ反復子オブジェクト” (38 ページ)

ステートメント:

- “DECLARE ステートメントのハッシュオブジェクトとハッシュ反復子オブジェクト” (12 ページ)

HAS_NEXT メソッド

現在のキーの複数データ項目リスト内に次の項目があるかどうかを判断します。

適用対象: ハッシュオブジェクト

構文

rc=*object*.**HAS_NEXT**(RESULT: R);

引数

rc
メソッドが成功したか失敗したかを示します。

ゼロのリターンコードは成功を表し、ゼロ以外の値は失敗を表します。メソッド呼び出しでリターンコード変数を指定せず、そのメソッドが失敗した場合、該当するエラーメッセージがログに書き込まれます。

オブジェクト
ハッシュオブジェクト名を指定します。

RESULT:R
数値変数 R を指定します。この変数は、データ項目リスト内に別のデータ項目がない場合はゼロ値、データ項目リスト内に別のデータ項目がある場合はゼロ以外の値を受け取ります。

詳細

1 つのキーに複数のデータ項目がある場合、HAS_NEXT メソッドを使用して、現在のキーの複数データ項目リスト内に次の項目があるかどうかを判断できます。別の項目がある場合、このメソッドは数値変数 R でゼロ以外の値を返します。別の項目がない場合、ゼロを返します。

FIND メソッドは、キーがハッシュオブジェクト内に存在するかどうかを判断します。HAS_NEXT メソッドは、キーに複数のデータ項目が関連付けられているかどうかを判断します。キーに別のデータ項目があることが判明した場合、FIND_NEXT メソッドを使用してそのデータ項目を取得できます。これによりデータ変数にデータ項目の値が設定され、メソッド呼び出し後に使用できるようになります。データ項目リスト内では、HAS_NEXT メソッドと FIND_NEXT メソッドに加えて HAS_PREV メソッドと FIND_PREV メソッドを使用することで、リスト内を移動できます。

例: データ項目の検索

この例では、複数のキーに複数のデータ項目があるハッシュオブジェクトを作成します。さらに、HAS_NEXT メソッドを使用してすべてのデータ項目を検索します。

```
data testdup;
length key data 8;
input key data;
datalines;
1 100
2 11
1 15
3 20
2 16
2 9
3 100
5 5
1 5
4 6
5 99
;
data _null_;
length r 8;
dcl hash h(dataset:'testdup', multidata: 'y');
h.definekey('key');
h.definedata('key', 'data');
h.definedone();
call missing (key, data);
do key = 1 to 5;
rc = h.find();
if (rc = 0) then do;
put key= data=;
h.has_next(result: r);
do while(r ne 0);
rc = h.find_next();
put 'dup ' key= data;
h.has_next(result: r);
end;
end;
end;
run;
```

次の行が SAS ログに書き込まれます。

```
key=1 data=100
dup key=1 5
dup key=1 15
key=2 data=11
dup key=2 9
dup key=2 16
key=3 data=20
dup key=3 100
key=4 data=6
key=5 data=5
dup key=5 99
```

関連項目:

- “Using the Hash Iterator Object ” (*SAS Language Reference: Concepts*)

メソッド:

- “FIND メソッド” (26 ページ)
- “FIND_NEXT メソッド” (29 ページ)
- “FIND_PREV メソッド” (30 ページ)
- “HAS_PREV メソッド” (35 ページ)

HAS_PREV メソッド

現在のキーの複数データ項目リスト内に前の項目があるかどうかを判断します。

適用対象: ハッシュオブジェクト

構文

rc=*object*.**HAS_PREV**(RESULT: R);

引数

rc
メソッドが成功したか失敗したかを示します。

ゼロのリターンコードは成功を表し、ゼロ以外の値は失敗を表します。メソッド呼び出しでリターンコード変数を指定せず、そのメソッドが失敗した場合、該当するエラーメッセージがログに書き込まれます。

オブジェクト
ハッシュオブジェクト名を指定します。

RESULT:R
数値変数 R を指定します。この変数は、データ項目リスト内に別のデータ項目がない場合はゼロ値、データ項目リスト内に別のデータ項目がある場合はゼロ以外の値を受け取ります。

詳細

1 つのキーに複数のデータ項目がある場合、HAS_PREV メソッドを使用して、現在のキーの複数データ項目リスト内に前の項目があるかどうかを判断できます。前の項目

がある場合、このメソッドは数値変数 R でゼロ以外の値を返します。別の項目がない場合、ゼロを返します。

FIND メソッドは、キーがハッシュオブジェクト内に存在するかどうかを判断します。HAS_NEXT メソッドは、キーに複数のデータ項目が関連付けられているかどうかを判断します。キーに前のデータ項目があることが判明した場合、FIND_PREV メソッドを使用してそのデータ項目を取得できます。これによりデータ変数にデータ項目の値が設定され、メソッド呼び出し後に使用できるようになります。データ項目リスト内では、HAS_NEXT メソッドと FIND_NEXT メソッドに加えて HAS_PREV メソッドと FIND_PREV メソッドを使用することで、リスト内を移動できます。例については、"HAS_NEXT メソッド" (33 ページ)を参照してください。

関連項目:

- "Non-Unique Key and Data Pairs" (*SAS Language Reference: Concepts*)

メソッド:

- "FIND メソッド" (26 ページ)
- "FIND_NEXT メソッド" (29 ページ)
- "FIND_PREV メソッド" (30 ページ)
- "HAS_NEXT メソッド" (33 ページ)

ITEM_SIZE 属性

ハッシュオブジェクト内の項目のサイズ(バイト)を返します。

適用対象: ハッシュオブジェクト

構文

variable_name=*object*.**ITEM_SIZE**;

引数

variable_name
ハッシュオブジェクト内の項目のサイズを含む変数名を指定します。

object
ハッシュオブジェクト名を指定します。

詳細

ITEM_SIZE 属性は、項目のサイズ(バイト)を返します。これには、キーおよびデータ変数の他に、いくつかの追加内部情報が含まれます。ハッシュオブジェクトが ITEM_SIZE 属性と NUM_ITEMS 属性で使用するメモリ量の推定値を取得できます。ITEM_SIZE 属性は、ハッシュオブジェクトが必要とする初期オーバーヘッドを反映せず、必要な内部配置も考慮しません。そのため、ITEM_SIZE の使用では正確なメモリ使用量は提供されず、近似値が返されます。

例: ハッシュ項目のサイズを返す

次の例では、ITEM_SIZE を使用して MYHASH 内の項目のサイズを返します。

```
data work.stock; input prod $1-10 qty 12-14; datalines; broccoli 345 corn 389 potato 993
```

itemsize=40 が SAS ログに書き込まれます。

LAST メソッド

基となるハッシュオブジェクトの最終値を返します。

適用対象: ハッシュ反復子オブジェクト

構文

rc=*object*.**LAST**( );

引数

rc
メソッドが成功したか失敗したかを示します。

ゼロのリターンコードは成功を表し、ゼロ以外の値は失敗を表します。メソッド呼び出しでリターンコード変数を指定せず、そのメソッドが失敗した場合、該当するエラーメッセージがログに書き込まれます。

オブジェクト
ハッシュ反復子オブジェクト名を指定します。

詳細

LAST メソッドは、ハッシュオブジェクト内の最後のデータ項目を返します。ハッシュオブジェクトをインスタンス化するときに、DECLARE ステートメントまたは_NEW_演算子で **ordered: 'yes'**または **ordered: 'ascending'**引数タグを使用した場合、ハッシュオブジェクト内のデータ項目はキー値の昇順で並べ替えられるため、'最大'キー(最大の数値またはアルファベット順の最後)のデータ項目が返されます。反対に、ハッシュオブジェクトをインスタンス化するときに、DECLARE ステートメントまたは_NEW_演算子で **ordered: 'descending'**引数タグを使用した場合、ハッシュオブジェクト内のデータ項目はキー値の降順で並べ替えられるため、'最小'キー(最小の数値またはアルファベット順の最初)のデータ項目が返されます。

ハッシュオブジェクト内の最初のデータ項目を返すには、FIRST メソッドを使用します。

注: LAST メソッドは、データ変数にデータ項目の値を設定し、メソッド呼び出し後にそのデータ項目を使用できるようにします。

関連項目:

- “Using the Hash Iterator Object ” (*SAS Language Reference: Concepts*)

メソッド:

- “FIRST メソッド” (31 ページ)

演算子:

- “_NEW_ Operator、ハッシュオブジェクトとハッシュ反復子オブジェクト” (38 ページ)

ステートメント:

- “DECLARE ステートメントのハッシュオブジェクトとハッシュ反復子オブジェクト” (12 ページ)

NEW Operator、ハッシュオブジェクトとハッシュ反復子オブジェクト

ハッシュオブジェクトまたはハッシュ反復子オブジェクトのインスタンスを作成します。

適用対象: ハッシュオブジェクト, ハッシュ反復子オブジェクト

構文

object-reference = **_NEW_***object* (<*argument_tag-1*: *value-1* <, …*argument_tag-n*: *value-n*> >);

引数

object-reference
ハッシュオブジェクトまたはハッシュ反復子オブジェクトのオブジェクト参照名を指定します。

object
コンポーネントオブジェクトを指定します。次のいずれかを指定できます。

hash ハッシュオブジェクトを示します。ハッシュオブジェクトは、迅速なデータの保存および取得のメカニズムを提供します。ハッシュオブジェクトは、ルックアップキーに基づいてデータを保存および取得します。

hiter ハッシュ反復子オブジェクトを示します。ハッシュ反復子オブジェクトを使用すると、ハッシュオブジェクトのデータを正または逆のキー順序で取得できます。

参照項目 “Using DATA Step Component Objects” (*SAS Language Reference: Concepts*) および “Using the Hash Iterator Object ” (*SAS Language Reference: Concepts*)

argument_tag*:*value
ハッシュオブジェクトのインスタンスの作成に使用する情報を指定します。

有効なハッシュオブジェクトの引数タグおよび値は、次のとおりです。

dataset: '*dataset_name* <(*datasetoption*)>'
ハッシュオブジェクトに読み込む SAS データセットの名前を指定します。

SAS データセットの名前には、リテラルまたは文字変数を使用できます。データセット名は一重引用符または二重引用符で囲む必要があります。マクロ変数は二重引用符で囲む必要があります。

DATASET 引数タグでハッシュオブジェクトを宣言するときに、SAS データセットオプションを使用できます。データセットオプションは、データセットオプションが表示される SAS データセットにのみ適用されるアクションを指定します。次の操作を実行できます。

- 変数名の変更
- 処理するオブザベーション番号に基づくオブザベーションのサブセットの選択
- WHERE オプションを使用したオブザベーションの選択

- ハッシュオブジェクトに読み込まれたデータセット、または OUTPUT メソッド呼び出しで指定されている出力データセットの変数の削除または保持
- データセットのパスワードの指定

次の構文が使用されます。

```
dcl hash h;
h = _new_ hash (dataset: 'x (where = (i > 10))');
```

SAS データセットオプションのリストについては、*SAS データセットオプション: リファレンス*を参照してください。

注 データセットに重複するキーが含まれている場合、デフォルトでは、ハッシュオブジェクトの最初のインスタンスが保持され、後続のインスタンスは無視されます。重複するキーがある場合に、ハッシュオブジェクトの最後のインスタンスを保存するか、または SAS ログにエラーメッセージを書き込むには、DUPLICATE 引数タグを使用します。

duplicate: '*option*'
ハッシュオブジェクトにデータセットを読み込むときに重複するキーを無視するかどうかを判断します。デフォルトでは、最初のキーが保存され、後続の重複はすべて無視されます。オプションには、次のいずれかの値を使用できます。

'replace' | 'r'
最後の重複するキーレコードを保存します。

'error' | 'e'
重複するキーが検出された場合に、ログにエラーを報告します。

REPLACE オプションを使用する次の例では、キー 620 には **brown**、キー 531 には **blue** が保存されます。デフォルトを使用する場合は、620 には **green**、531 には **yellow** が保存されます。

```
data table;
  input key data $;
  datalines;
  531 yellow
  620 green
  531 blue
  908 orange
  620 brown
  143 purple
 run;
data _null_;
length key 8 data $ 8;
if (_n_ = 1) then do;
    declare hash myhash;
    myhash = _new_ hash (dataset: "table", duplicate: "r");
    rc = myhash.definekey('key');
    rc = myhash.definedata('data');
    myhash.definedone();
 end;
rc = myhash.output(dataset:"otable");
run;
```

hashexp: *n*
ハッシュオブジェクトの内部テーブルサイズです。ハッシュテーブルのサイズは 2^n です。

HASHEXP の値は、ハッシュテーブルサイズを作成する 2 の累乗指数として使用されます。たとえば、HASHEXP の値 4 は、ハッシュテーブルサイズ 2^4(つまり 16)と同じです。HASHEXP の最大値は 20 です。

ハッシュテーブルサイズは、保存可能な項目数とは等しくありません。ハッシュテーブルを、一連の'バケット'と考えます。ハッシュテーブルサイズ 16 には、16 個の'バケット'があります。各バケットに保持できる項目の数は無限です。ハッシュテーブルの効率は、項目をバケットにマップし、バケットから項目を取得するハッシュ関数の機能にあります。

ハッシュオブジェクトのルックアップルーチンの効率を最大にするには、ハッシュオブジェクトのデータ量に応じてハッシュテーブルサイズを設定する必要があります。最適な結果が得られるまで、さまざまな HASHEXP 値を試してみてください。たとえば、ハッシュオブジェクトに 100 万個の項目が含まれている場合、ハッシュテーブルサイズ 16(HASHEXP = 4)でも機能しますが、効率は良くありません。ハッシュテーブルサイズ 512 または 1024(HASHEXP = 9 または 10)を使用すると、最高の処理速度が得られます。

デフォルト 8。これは、ハッシュテーブルサイズ 2^8 (つまり 256)と同じです。

keysum:'*variable-name*'
すべてのキーのキー集計をトラッキングする変数の名前を指定します。キー集計はキーが何回 FIND メソッド呼び出しに参照されたかのカウントです。

注 キー集計は出力データセットにあります。

ordered: '*option*'
ハッシュオブジェクトとハッシュ反復子オブジェクトを併用する場合、またはハッシュオブジェクト OUTPUT メソッドを使用する場合、データがキー値の順序で返されるかどうか、またはどのように返されるかを指定します。

引数値は二重引用符で囲むこともできます。

option には、次のいずれかの値を使用できます。

'ascending' | 'a'	データはキー値の昇順で返されます。'`ascending`'の指定は、'`yes`'を指定することと同じです。
'descending' | 'd'	データはキー値の降順で返されます。
'YES' | 'Y'	データはキー値の昇順で返されます。'`yes`'の指定は、'`ascending`'を指定することと同じです。
'NO' | 'N'	データは未定義の順序で返されます。

デフォルト NO

multidata: '*option*'
各キーに複数のデータ項目を許可するかどうかを指定します。

引数値は二重引用符で囲むこともできます。

`option` には、次のいずれかの値を使用できます。

'YES' | 'Y'	各キーに複数のデータ項目が許可されます。
'NO' | 'N'	各キーにデータ項目が 1 つのみ許可されます。

デフォルト NO

参照項目 “Non-Unique Key and Data Pairs” (*SAS Language Reference: Concepts*)

suminc: '*variable-name*'
ハッシュオブジェクトキーの集計数を維持します。SUMINC 引数タグは、DATA ステップ変数を指定します。この変数には、合計増分が保持されます。合計増分はキーが参照されるたびにキー集計に追加される数です。たとえば、キー集計は、DATA ステップ変数の現在の値を使用して変更されます。

```
dcl hash myhash(suminc: 'count');
```

詳細については、“Maintaining Key Summaries” (*SAS Language Reference: Concepts*)を参照してください。

参照項目 “Initializing Hash Object Data Using a Constructor” (*SAS Language Reference: Concepts*) および “Declaring and Instantiating a Hash Object” (*SAS Language Reference: Concepts*)

詳細

SAS プログラムで DATA ステップコンポーネントオブジェクトを使用するには、オブジェクトを宣言して作成(インスタンス化)する必要があります。DATA ステップコンポーネントインターフェイスは、DATA ステップ内から定義済みのコンポーネントオブジェクトにアクセスするメカニズムを提供します。

_NEW_演算子を使用してコンポーネントオブジェクトをインスタンス化する場合は、最初に DECLARE ステートメントを使用してコンポーネントオブジェクトを宣言する必要があります。たとえば、次のコード行では、DECLARE ステートメントは、オブジェクト参照 H がハッシュオブジェクトであることを SAS に示します。_NEW_演算子でハッシュオブジェクトが作成され、オブジェクト参照 H に割り当てられます。

```
declare hash h(); h = _new_ hash( );
```

注: DECLARE ステートメントを使用してハッシュオブジェクトまたはハッシュ反復子オブジェクトの宣言とインスタンス化を 1 つのステップで実行できます。

コンストラクタは、コンポーネントオブジェクトをインスタンス化し、コンポーネントオブジェクトデータを初期化するために使用するメソッドです。たとえば、次のコード行では、_NEW_演算子でハッシュオブジェクトがインスタンス化され、オブジェクト参照 H に割り当てられます。さらに、データセット WORK.KENNEL がハッシュオブジェクトに読み込まれます。

```
declare hash h(); h = _new_ hash(datset: "work.kennel");
```

定義済みの DATA ステップコンポーネントオブジェクトおよびコンストラクタの詳細については、“Using DATA Step Component Objects” (*SAS Language Reference: Concepts*) を参照してください。

比較

ハッシュオブジェクトまたはハッシュ反復子オブジェクトのインスタンスを宣言し、インスタンス化するには、DECLARE ステートメントと_NEW_演算子を使用するか、DECLARE ステートメントのみを使用します。

例: _NEW_演算子を使用したハッシュオブジェクトデータのインスタンス化と初期化

この例では、_NEW_演算子を使用して、ハッシュオブジェクトのデータをインスタンス化して初期化し、ハッシュ反復子オブジェクトをインスタンス化します。

ハッシュオブジェクトにはデータが含まれており、データをキー順序で取得するために反復子が使用されます。

```
data kennel; input name $1-10 kenno $14-15; datalines; Charlie      15 Tanner       07 J
```

次の行が SAS ログに書き込まれます。

```
注: There were 7 observations read from the data set WORK.KENNEL.01    Murphy
04    Jake 07    Tanner 09    Pepe 11    Jacques 12    Princess Z 15    Charlie
```

関連項目:

- “Using DATA Step Component Objects” (*SAS Language Reference: Concepts*)

ステートメント:

- “DECLARE ステートメントのハッシュオブジェクトとハッシュ反復子オブジェクト” (12 ページ)

NEXT メソッド

基となるハッシュオブジェクトの次の値を返します。

適用対象: ハッシュ反復子オブジェクト

構文

rc=*object*.**NEXT**( );

引数

rc
メソッドが成功したか失敗したかを示します。

ゼロのリターンコードは成功を表し、ゼロ以外の値は失敗を表します。メソッド呼び出しでリターンコード変数を指定せず、そのメソッドが失敗した場合、該当するエラーメッセージがログに書き込まれます。

オブジェクト
ハッシュ反復子オブジェクト名を指定します。

詳細

NEXT メソッドを繰り返し使用することで、ハッシュオブジェクト内を移動し、データ項目をキーの順序で返すことができます。

FIRST メソッドは、ハッシュオブジェクト内の最初のデータ項目を返します。

ハッシュオブジェクト内の前のデータ項目を返すには、PREV メソッドを使用できます。

注: NEXT メソッドは、データ変数にデータ項目の値を設定し、メソッド呼び出し後にそのデータ項目を使用できるようにします。

注: FIRST メソッドを呼び出さずに NEXT メソッドを呼び出した場合でも、NEXT メソッドはハッシュオブジェクト内の最初の項目から開始します。

関連項目:

- “Using the Hash Iterator Object ” (*SAS Language Reference: Concepts*)

メソッド:

- “FIRST メソッド” (31 ページ)
- “PREV メソッド” (49 ページ)

演算子:

- “_NEW_ Operator、ハッシュオブジェクトとハッシュ反復子オブジェクト” (38 ページ)

ステートメント:

- “DECLARE ステートメントのハッシュオブジェクトとハッシュ反復子オブジェクト” (12 ページ)

NUM_ITEMS 属性

ハッシュオブジェクト内の項目数を返します。

適用対象: ハッシュオブジェクト

構文

variable_name=*object*.**NUM_ITEMS**;

引数

variable_name
ハッシュオブジェクト内の項目数を含む変数名を指定します。

オブジェクト
ハッシュオブジェクト名を指定します。

例: ハッシュオブジェクト内の項目数を返す

この例では、データセットを作成し、そのデータセットをハッシュオブジェクト内に読み込みます。項目がハッシュオブジェクトに追加され、その結果のハッシュオブジェクト内の項目の合計数が NUM_ITEMS 属性によって返されます。

```
data work.stock;
input item $ qty;
datalines;
broccoli 345
corn 389
potato 993
onion 730
;
data _null_;
if _N_ = 1 then do;
length item $10;
length qty 8;
```

```
length totalitems 8;
/* Declare hash object and read STOCK data set as ordered */
declare hash myhash(dataset: "work.stock");
/* Define key and data variables */
myhash.defineKey('item');
myhash.defineData('qty');
myhash.defineDone();
end;
/* Add a key and data value to the hash object */
item = 'celery';
qty = 183;
rc = myhash.add();
if (rc ne 0) then
put 'Add failed';
/* Use NUM_ITEMS to return updated number of items in hash object */
totalitems = myhash.num_items;
put totalitems=;
run;
```

totalitems=5 が SAS ログに書き込まれます。

OUTPUT メソッド

ハッシュオブジェクトにデータが格納された 1 つ以上のデータセットを作成します。

適用対象: ハッシュオブジェクト

構文

rc=*object*.**OUTPUT** (DATASET: '*dataset-1* <(*datasetoption*)> '
<, …<DATASET: '*dataset-n*>'> ('*datasetoption* <(*datasetoption*)> ');

引数

rc
メソッドが成功したか失敗したかを示します。

ゼロのリターンコードは成功を表し、ゼロ以外の値は失敗を表します。メソッド呼び出しでリターンコード変数を指定せず、そのメソッドが失敗した場合、該当するエラーメッセージがログに書き込まれます。

object
ハッシュオブジェクト名を指定します。

DATASET: '*dataset*'
出力データセット名を指定します。

SAS データセットの名前には、文字リテラルまたは文字変数を使用できます。データセット名は二重引用符で囲むこともできます。出力データセットの名前を指定するときには、DATASET 引数タグで SAS データセットオプションを使用できます。マクロ変数は二重引用符で囲む必要があります。

datasetoption
データセットオプションを指定します。

データセットオプションを指定する方法の詳細については、“Syntax” (*SAS Data Set Options: Reference*)を参照してください。

詳細

ハッシュオブジェクトキーは、出力データセットの一部として自動的には保存されません。出力データセットにキーを含めるには、DEFINEDATA メソッドを使用してデータ項目としてキーを定義する必要があります。

ハッシュオブジェクトをインスタンス化するときに、DECLARE ステートメントまたは _NEW_ 演算子で **ordered: 'yes'** または **ordered: 'ascending'** 引数タグを使用する場合、データ項目はキー値の昇順でデータセットに書き込まれます。ハッシュオブジェクトをインスタンス化するときに、DECLARE ステートメントまたは _NEW_ 演算子で **ordered: 'descending'** 引数タグを使用する場合、データ項目はキー値の降順でデータセットに書き込まれます。**ordered** 引数タグを使用しない場合、順序は未定義になります。

出力データセットの名前を指定するときには、DATASET 引数タグで SAS データセットオプションを使用できます。データセットオプションは、データセットオプションが表示される SAS データセットにのみ適用されるアクションを指定します。次の操作を実行できます。

- 変数名の変更
- 処理するオブザベーション番号に基づくオブザベーションのサブセットの選択
- WHERE オプションを使用したオブザベーションの選択
- ハッシュオブジェクトに読み込まれたデータセット、または OUTPUT メソッド呼び出しで指定されている出力データセットの変数の削除または保持

  注: 削除または保持する変数は、DEFINEDATA メソッドまたは DEFINEKEY メソッドを使用することにより、ハッシュ テーブルに含めておく必要があります。そうでない場合、エラーが発生します。
- データセットのパスワードの指定

次の例では、WHERE データセットオプションを使用して、出力データセット OUT の特定のデータを選択します。

```
data x;
 do i = 1 to 20;
   output;
 end;
run;

/* Using the WHERE option. */
data _null_;
  length i 8;
  dcl hash h(dataset:'x');
  h.definekey(all: 'y');
  h.definedone();
  h.output(dataset: 'out (where =( i < 8))');
run;
```

次の例では、RENAME データセットオプションを使用して、出力データセット OUT の変数名 J を K に変更します。

```
data x;
 do i = 1 to 20;
   output;
 end;
run;
/* Using the RENAME option. */
data _null_;
```

```
  length i j 8;
  dcl hash h(dataset:'x');
  h.definekey(all: 'y');
  h.definedone();
  h.output(dataset: 'out (rename =(i=k))');
run;
```

データセットオプションのリストについては、*SAS データセットオプション: リファレンス* を参照してください。

注: OUTPUT メソッドを使用してデータセットを作成すると、ハッシュオブジェクトは出力データセットには含まれません。次の例では、H2 ハッシュオブジェクトは出力データセットから省略されます。

```
data _null_;
   length k 8;
   length d $10;
   declare hash h2();
   declare hash h(ordered: 'y');
   h.defineKey('k');
   h.defineData('k', 'd', 'h2');
   h.defineDone();
   k = 99;
   d = 'abc';
   h.add();
   k = 199;
   d = 'def';
   h.add();
   h.output(dataset:'work.x');
run;
```

例

次のコードでは、天文学データを含むデータセット ASTRO を使用して、メシエ(OBJ)天体が赤経(RA)値の昇順で並べ替えられたハッシュオブジェクトを作成し、OUTPUT メソッドを使用してデータセットにデータを保存します。

```
data astro;
input obj $1-4 ra $6-12 dec $14-19;
datalines;
 M31 00 42.7 +41 16
 M71 19 53.8 +18 47
 M51 13 29.9 +47 12
 M98 12 13.8 +14 54
 M13 16 41.7 +36 28
 M39 21 32.2 +48 26
 M81 09 55.6 +69 04
M100 12 22.9 +15 49
 M41 06 46.0 -20 44
 M44 08 40.1 +19 59
 M10 16 57.1 -04 06
 M57 18 53.6 +33 02
  M3 13 42.2 +28 23
 M22 18 36.4 -23 54
 M23 17 56.8 -19 01
 M49 12 29.8 +08 00
 M68 12 39.5 -26 45
```

```
 M17 18 20.8 -16 11
 M14 17 37.6 -03 15
 M29 20 23.9 +38 32
 M34 02 42.0 +42 47
 M82 09 55.8 +69 41
 M59 12 42.0 +11 39
 M74 01 36.7 +15 47
 M25 18 31.6 -19 15
;
run;
data _null_;
   if _N_ = 1 then do;
      length obj $10;
      length ra $10;
      length dec $10;
      /* Read ASTRO data set as ordered */
   declare hash h(hashexp: 4, dataset:"work.astro", ordered: 'yes');
      /* Define variables RA and OBJ as key and data for hash object */
      h.defineKey('ra');
      h.defineData('ra', 'obj');
      h.defineDone();
      /* avoid uninitialized variable notes */
      call missing(ra, obj);
end;
/* Create output data set from hash object */
rc = h.output(dataset: 'work.out');
run;

proc print data=work.out;
   var ra obj;
   title 'Messier Objects Sorted by Right-Ascension Values';
run;
```

アウトプット 2.2 赤経値で並べ替えられたメシエ天体

Messier Objects Sorted by Right-Ascension Values

Obs	ra	obj
1	00 42.7	M31
2	01 36.7	M74
3	02 42.0	M34
4	06 46.0	M41
5	08 40.1	M44
6	09 55.6	M81
7	09 55.8	M82
8	12 13.8	M98
9	12 22.9	M100
10	12 29.8	M49
11	12 39.5	M68
12	12 42.0	M59
13	13 29.9	M51
14	13 42.2	M3
15	16 41.7	M13
16	16 57.1	M10
17	17 37.6	M14
18	17 56.8	M23
19	18 20.8	M17
20	18 31.6	M25
21	18 36.4	M22
22	18 53.6	M57
23	19 53.8	M71
24	20 23.9	M29
25	21 32.2	M39

関連項目:

- “Saving Hash Object Data in a Data Set” (*SAS Language Reference: Concepts*)

メソッド:

- “DEFINEDATA メソッド” (19 ページ)

演算子:

- “_NEW_ Operator、ハッシュオブジェクトとハッシュ反復子オブジェクト” (38 ページ)

ステートメント:

- “DECLARE ステートメントのハッシュオブジェクトとハッシュ反復子オブジェクト” (12 ページ)

PREV メソッド

基となるハッシュオブジェクトの前の値を返します。

適用対象: ハッシュ反復子オブジェクト

構文

rc=*object*.**PREV**( );

引数

rc
メソッドが成功したか失敗したかを示します。

ゼロのリターンコードは成功を表し、ゼロ以外の値は失敗を表します。メソッド呼び出しでリターンコード変数を指定せず、そのメソッドが失敗した場合、該当するエラーメッセージがログに書き込まれます。

オブジェクト
ハッシュ反復子オブジェクト名を指定します。

詳細

ハッシュオブジェクトを移動して逆のキー順序でデータ項目を返すには、PREV メソッドを反復して使用します。

FIRST メソッドは、ハッシュオブジェクト内の最初のデータ項目を返します。LAST メソッドは、ハッシュオブジェクト内の最後のデータ項目を返します。

ハッシュオブジェクトの次のデータ項目を返すには、NEXT メソッドを使用できます。

注: PREV メソッドは、メソッド呼び出し後に使用できるように、データ変数をデータ項目の値に設定します。

関連項目:

- “Using the Hash Iterator Object ” (*SAS Language Reference: Concepts*)

メソッド:

- “FIRST メソッド” (31 ページ)
- “LAST メソッド” (37 ページ)

- “NEXT メソッド” (42 ページ)

演算子:

- “_NEW_ Operator、ハッシュオブジェクトとハッシュ反復子オブジェクト” (38 ページ)

ステートメント:

- “DECLARE ステートメントのハッシュオブジェクトとハッシュ反復子オブジェクト” (12 ページ)

REF メソッド

CHECK メソッドと ADD メソッドを 1 つのメソッド呼び出しに統合します。

適用対象: ハッシュオブジェクト

構文

rc=*object*.**REF** (<<KEY: *keyvalue-1*>, … <KEY: *keyvalue-n*>, <DATA: *datavalue-1*> , …<DATA: *datavalue-n*>>);

引数

rc
メソッドが成功したか失敗したかを示します。

ゼロのリターンコードは成功を表し、ゼロ以外の値は失敗を表します。メソッド呼び出しでリターンコード変数を指定せず、そのメソッドが失敗した場合、該当するエラーメッセージがログに書き込まれます。

object
ハッシュオブジェクト名を指定します。

KEY: *keyvalue*
DEFINEKEY メソッド呼び出しで指定された、対応するキー変数に一致する型のキー値を指定します。

"KEY: *keyvalue*"ペアの数は、DEFINEKEY メソッドを使用して定義されたキー変数の数によって変わります。

DATA: *datavalue*
DEFINEDATA メソッド呼び出しで指定された、対応するデータ変数に一致する型のデータ値を指定します。

"DATA: *datavalue*"ペアの数は、DEFINEDATA メソッドを使用して定義されたデータ変数の数によって変わります。

詳細

CHECK メソッドと ADD メソッドを 1 つの REF メソッド呼び出しに統合できます。次のコードを変更できます。

```
rc = h.check();
  if (rc ne 0) then
    rc = h.add();
```

変更後のコード

```
rc = h.ref();
```

REF メソッドは、ハッシュオブジェクト内の各キーの出現数を数えるのに便利です。REF メソッドは、最初の ADD で各キーのキー集計を初期化し、後続の CHECK が出現するたびに ADD を変更します。

キー集計の詳細については、“Maintaining Key Summaries” (*SAS Language Reference: Concepts*) を参照してください。

例: REF メソッドを使用したキー集計

次の例では、キー集計に REF メソッドを使用します。

```
data keys;
 input key;
datalines;
1
2
1
3
5
2
3
2
4
1
5
1
;
data count;
 length count key 8;
 keep key count;
 if _n_ = 1 then do;
   declare hash myhash(suminc: "count", ordered: "y");
   declare hiter iter("myhash");
   myhash.defineKey('key');
   myhash.defineDone();
   count = 1;
  end;
 do while (not done);
   set keys end=done;
   rc = myhash.ref();
 end;
 rc = iter.first();
 do while(rc = 0);
   rc = myhash.sum(sum: count);
   output;
   rc = iter.next();
 end;
 stop;
run;

proc print data=count;
run;
```

アウトプット 2.3 *REF メソッドを使用した DATA の出力*

The SAS System

Obs	count	key
1	4	1
2	3	2
3	2	3
4	1	4
5	2	5

関連項目:

メソッド:

- “ADD メソッド” (7 ページ)
- “CHECK メソッド” (9 ページ)

REMOVE メソッド

指定したキーに関連付けられているデータをハッシュオブジェクトから削除します。

適用対象: ハッシュオブジェクト

構文

rc=*object*.**REMOVE**(<KEY: *keyvalue-1*, …KEY: *keyvalue-n*>);

引数

rc
メソッドが成功したか失敗したかを示します。

ゼロのリターンコードは成功を表し、ゼロ以外の値は失敗を表します。メソッド呼び出しでリターンコード変数を指定せず、そのメソッドが失敗した場合、該当するエラーメッセージがログに書き込まれます。

オブジェクト
ハッシュオブジェクト名を指定します。

KEY: ***keyvalue***
DEFINEKEY メソッド呼び出しで指定されている対応するキー変数と型が一致するキー値を指定します。

"KEY: *keyvalue*"ペアの数は、DEFINEKEY メソッドを使用して定義されたキー変数の数によって変わります。

制限事項 関連付けられているハッシュ反復子が *keyvalue* を示している場合、REMOVE メソッドはハッシュオブジェクトからキーまたはデータを削除しません。エラーメッセージが発行されます。

詳細

REMOVE メソッドは、ハッシュオブジェクトからキーとデータの両方を削除します。

次の 2 つの方法のいずれかで REMOVE メソッドを使用して、ハッシュオブジェクトのキーおよびデータを削除できます。

1 つ目は、次のコードに示すように、キーを指定してから REMOVE メソッドを使用する方法です。

```
data _null_;
length k $8;
length d $12;
if _N_ = 1 then do;
declare hash h();
rc = h.defineKey('k');
rc = h.defineData('d');
rc = h.defineDone();
/* avoid uninitialized variable notes */
call missing(k, d);
end;
rc = h.add(key: 'Joyce', data: 'Ulysses');
/* Specify the key */
k = 'Joyce';
/* Use the REMOVE method to remove the key and data */
rc = h.remove();
if (rc = 0) then
put 'Key and data removed from the hash object.';
run;
```

2 つ目は、次のコードに示すように、ショートカットを使用して、REMOVE メソッド呼び出しでキーを直接指定する方法です。

```
data _null_;
length k $8;
length d $12;
if _N_ = 1 then do;
declare hash h();
rc = h.defineKey('k');
rc = h.defineData('d');
rc = h.defineDone();
/* avoid uninitialized variable notes */
call missing(k, d);
end;
rc = h.add(key: 'Joyce', data: 'Ulysses');
rc = h.add(key: 'Homer', data: 'Iliad');
/* Specify the key in the REMOVE method parameter */
rc = h.remove(key: 'Homer');
if (rc =0) then
put 'Key and data removed from the hash object.';
run;
```

注: REMOVE メソッドではデータ変数の値は変更されません。ハッシュオブジェクトの値が削除されるだけです。

注: ハッシュオブジェクトコンストラクタで **multidata:'y'** を指定すると、REMOVE メソッドは指定したキーのすべてのデータ項目を削除します。

例: ハッシュテーブルのキーの削除

この例では、ハッシュテーブルのキーを削除する方法を示します。

```
/* Generate test data */
data x;
do k = 65 to 70;
d = byte (k);
output;
end;
run;
data _null_;
length k 8 d $1;
/* define the hash table and iterator */
declare hash H (dataset:'x', ordered:'a');
H.defineKey ('k');
H.defineData ('k', 'd');
H.defineDone ();
call missing (k,d);
declare hiter HI ('H');
/* Use this logic to remove a key in the hash table
when an iterator is pointing to that key */
do while (hi.next() = 0);
if flag then rc=h.remove(key:key);
if d = 'C' then do;
key=k;
flag=1;
end;
end;
rc = h.output(dataset: 'work.out');
stop;
run;
proc print;
run;
```

次の出力は、3 番目のオブジェクトのキーとデータ(キー=67、データ=C)が削除されることを示しています。

アウトプット 2.4　出力から削除されるキーとデータ

The SAS System

Obs	k	d
1	65	A
2	66	B
3	68	D
4	69	E
5	70	F

関連項目:

- “Replacing and Removing Data in the Hash Object” (*SAS Language Reference: Concepts*)

メソッド:

- “ADD メソッド” (7 ページ)
- “DEFINEKEY メソッド” (22 ページ)
- “REMOVEDUP メソッド” (55 ページ)

REMOVEDUP メソッド

指定したキーの現在のデータ項目に関連付けられているデータをハッシュオブジェクトから削除します。

適用対象: ハッシュオブジェクト

構文

rc=*object*.**REMOVEDUP** (<KEY: *keyvalue-1*, …KEY: *keyvalue-n*>);

引数

rc
メソッドが成功したか失敗したかを示します。

ゼロのリターンコードは成功を表し、ゼロ以外の値は失敗を表します。メソッド呼び出しでリターンコード変数を指定せず、そのメソッドが失敗した場合、該当するエラーメッセージがログに書き込まれます。

object
ハッシュオブジェクト名を指定します。

KEY: ***keyvalue***
DEFINEKEY メソッド呼び出しで指定された、対応するキー変数に一致する型のキー値を指定します。

"KEY: *keyvalue*"ペアの数は、DEFINEKEY メソッドを使用して定義されたキー変数の数によって変わります。

制限事項 関連付けられているハッシュ反復子が *keyvalue* を示している場合、REMOVEDUP メソッドはハッシュオブジェクトからキーまたはデータを削除しません。エラーメッセージが発行されます。

詳細

REMOVEDUP メソッドは、ハッシュオブジェクトからキーとデータの両方を削除します。

次の 2 つの方法のいずれかで REMOVEDUP メソッドを使用して、ハッシュオブジェクトのキーおよびデータを削除できます。1 つ目は、キーを指定してから、REMOVEDUP メソッドを使用する方法です。2 つ目は、ショートカットを使用して、REMOVEDUP メソッド呼び出しでキーを直接指定する方法です。

注: REMOVEDUP メソッドでは、データ変数の値は変更されません。ハッシュオブジェクトの値が削除されるだけです。

注: キーのデータ項目リストにあるデータ項目が 1 つのみの場合は、キーとデータがハッシュオブジェクトから削除されます。

比較

REMOVEDUP メソッドでは、指定したキーの現在のデータ項目に関連付けられているデータがハッシュオブジェクトから削除されます。REMOVE メソッドでは、指定したキーに関連付けられているデータがハッシュオブジェクトから削除されます。

例: キーの重複する項目の削除

この例では、複数のキーに複数のデータ項目があるハッシュオブジェクトを作成します。キーの 2 番目のデータ項目が削除されます。

```
data testdup;
   length key data 8;
   input key data;
   datalines;
   1 10
   2 11
   1 15
   3 20
   2 16
   2 9
   3 100
   5 5
   1 5
   4 6
   5 99
;
data _null_;
   length r 8;
   dcl hash h(dataset:'testdup', multidata: 'y', ordered: 'y');
   h.definekey('key');
   h.definedata('key', 'data');
   h.definedone();
   call missing (key, data);
   do key = 1 to 5;
```

```
      rc = h.find();
      if (rc = 0) then do;
         h.has_next(result: r);
         if (r ne 0) then do;
            h.find_next();
            h.removedup();
         end;
      end;
   end;
   dcl hiter i('h');
   rc = i.first();
   do while (rc = 0);
      put key= data=;
      rc = i.next();
   end;
run;
```

次の行が SAS ログに書き込まれます。

```
key=1 data=10 key=1 data=5 key=2 data=11 key=2 data=9 key=3 data=20 key=4 data=6
key=5 data=5
```

関連項目:

- “Non-Unique Key and Data Pairs” (*SAS Language Reference: Concepts*)

メソッド:

- “REMOVE メソッド” (52 ページ)

REPLACE メソッド

指定したキーに関連付けられたデータを新しいデータに置き換えます。

適用対象: ハッシュオブジェクト

構文

rc=*object*.**REPLACE** (<<KEY: *keyvalue-1*>, …<KEY: *keyvalue-n*>, <DATA: *datavalue-1*>, …<DATA: *datavalue-n*>>);

引数

rc
メソッドが成功したか失敗したかを示します。

ゼロのリターンコードは成功を表し、ゼロ以外の値は失敗を表します。メソッド呼び出しでリターンコード変数を指定せず、そのメソッドが失敗した場合、該当するエラーメッセージがログに書き込まれます。

object
ハッシュオブジェクト名を指定します。

KEY: ***keyvalue***
DEFINEKEY メソッド呼び出しで指定された、対応するキー変数に一致する型のキー値を指定します。

"KEY: **keyvalue**"ペアの数は、DEFINEKEY メソッドを使用して定義されたキー変数の数によって変わります。

要件 KEY:*keyvalue* 引数は、ハッシュオブジェクト変数名が指定されていないため、ハッシュオブジェクトで定義された順序になっていなければなりません。

DATA: ***datavalue***
DEFINEDATA メソッド呼び出しで指定された、対応するデータ変数に一致する型のデータ値を指定します。

"DATA: **datavalue**"ペアの数は、DEFINEDATA メソッドを使用して定義されたデータ変数の数によって変わります。

要件 DATA:*datavalue* 引数は、ハッシュオブジェクト変数名が指定されていないため、ハッシュオブジェクトで定義された順序になっていなければなりません。

詳細

次の 2 つの方法のいずれかで REPLACE メソッドを使用して、ハッシュオブジェクト内のデータを置き換えることができます。

1 つ目は、次のコードに示すように、キーおよびデータ項目を定義してから REPLACE メソッドを使用する方法です。この例では、キー'Rottwlr'のデータが'1st'から'2nd'に変更されます。

```
data work.show;
   length brd $10 plc $8;
   input brd plc;
datalines;
Terrier    2nd
LabRetr    3rd
Rottwlr    1st
Collie     bis
ChinsCrstd 2nd
Newfnlnd   3rd
;

proc print data=work.show;
   title 'SHOW Data Set Before Replace';
run;

data _null_;
   length brd $12;
   length plc $8;
   if _N_ = 1 then do;
      declare hash h(dataset: 'work.show');
      rc = h.defineKey('brd');
      rc = h.defineData('brd', 'plc');
      rc = h.defineDone();
   end;
   /* Specify the key and new data value */
   brd = 'Rottwlr';
   plc = '2nd';
   /* Call the REPLACE method to replace the data value */
   rc = h.replace();
   /* Write the hash table to the data set. */
   rc = h.output(dataset: 'work.show');
run;
```

```
proc print data=work.show;
   title 'SHOW Data Set After Replace';
run;
```

2 つ目は、次のコードに示すように、ショートカットを使用して、REPLACE メソッド呼び出しでキーおよびデータを直接指定する方法です。

```
data work.show;
   length brd $10 plc $8;
   input brd plc;
datalines;
Terrier    2nd
LabRetr    3rd
Rottwlr    1st
Collie     bis
ChinsCrstd 2nd
Newfnlnd   3rd
;
data _null_;
   length brd $12;
   length plc $8;
   if _N_ = 1 then do;
      declare hash h(dataset: 'work.show');
      rc = h.defineKey('brd');
      rc = h.defineData('brd', 'plc');
      rc = h.defineDone();
      /* avoid uninitialized variable notes */
      call missing(brd, plc);
   end;
   /* Specify the key and new data value in the REPLACE method */
   rc = h.replace(key: 'Rottwlr', data: '2nd');
   /* Write the hash table to the data set. */
   rc = h.output(dataset: 'work.show');
run;
```

注: REPLACE メソッドを呼び出して **multidata:'y'** オプションを使用してハッシュオブジェクトを宣言した場合、キーとデータはハッシュオブジェクトに追加されます。MULTIDATA オプションの詳細については、“DECLARE ステートメントのハッシュオブジェクトとハッシュ反復子オブジェクト” (12 ページ)を参照してください。

注: REPLACE メソッドを呼び出してキーが見つからなかった場合、キーとデータがハッシュオブジェクトに追加されます。

注: REPLACE メソッドでは、データ変数の値はデータ項目の値に置き換えられません。ハッシュオブジェクト内の値のみが置き換えられます。

比較

REPLACE メソッドでは、指定したキーに関連付けられたデータが新しいデータに置き換えられます。REPLACEDUP メソッドでは、現在のキーの現在のデータ項目に関連付けられたデータが新しいデータに置き換えられます。

関連項目:

- “Replacing and Removing Data in the Hash Object” (*SAS Language Reference: Concepts*)

メソッド:

- “DEFINEDATA メソッド” (19 ページ)
- “DEFINEKEY メソッド” (22 ページ)
- “REPLACEDUP メソッド” (60 ページ)

REPLACEDUP メソッド

現在のキーの現在のデータ項目に関連付けられたデータを新しいデータに置き換えます。

適用対象: ハッシュオブジェクト

構文

rc=*object*.**REPLACEDUP**(<DATA: *datavalue-1*, …DATA: *datavalue-n*>);

引数

rc
メソッドが成功したか失敗したかを示します。

ゼロのリターンコードは成功を表し、ゼロ以外の値は失敗を表します。メソッド呼び出しでリターンコード変数を指定せず、そのメソッドが失敗した場合、該当するエラーメッセージがログに書き込まれます。

オブジェクト
ハッシュオブジェクト名を指定します。

DATA: ***datavalue***
DEFINEDATA メソッド呼び出しで指定された、対応するデータ変数に一致する型のデータ値を指定します。

"DATA: *datavalue*"ペアの数は、DEFINEDATA メソッドを使用して現在のキーに定義されたデータ変数の数によって変わります。

詳細

次の 2 つの方法のいずれかで REPLACEUP メソッドを使用して、ハッシュオブジェクト内のデータを置き換えることができます。

1 つ目は、データ項目を定義してから、REPLACEDUP メソッドを使用する方法です。2 つ目は、ショートカットを使用して、REPLACEDUP メソッド呼び出しでデータを直接指定する方法です。

注: REPLACEDUP メソッドを呼び出してキーが見つからなかった場合、キーとデータがハッシュオブジェクトに追加されます。

注: REPLACEDUP メソッドでは、データ変数の値はデータ項目の値に置き換えられません。ハッシュオブジェクト内の値のみが置き換えられます。

比較

REPLACEDUP メソッドでは、現在のキーの現在のデータ項目に関連付けられたデータが新しいデータに置き換えられます。REPLACE メソッドでは、指定したキーに関連付けられたデータが新しいデータに置き換えられます。

例: 現在のキーのデータの置換

この例では、複数のキーに複数のデータ項目があるハッシュオブジェクトを作成します。重複データ項目が見つかった場合、データ項目の値に 300 が加算されます。

```
data testdup;
length key data 8;
input key data;
datalines;
1 10
2 11
1 15
3 20
2 16
2 9
3 100
5 5
1 5
4 6
5 99
;
data _null_;
length r 8;
dcl hash h(dataset:'testdup', multidata: 'y', ordered: 'y');
h.definekey('key');
h.definedata('key', 'data');
h.definedone();
call missing (key, data);
do key = 1 to 5;
rc = h.find();
if (rc = 0) then do;
put key= data=;
h.has_next(result: r);
do while(r ne 0);
rc = h.find_next();
put 'dup ' key= data;
data = data + 300;
rc = h.replacedup();
h.has_next(result: r);
end;
end;
end;
put 'iterating...';
dcl hiter i('h');
rc = i.first();
do while (rc = 0);
put key= data=;
rc = i.next();
end;
run;
```

次の行が SAS ログに書き込まれます。

```
key=1 data=10
dup key=1 15
dup key=1 5
key=2 data=11
dup key=2 16
dup key=2 9
key=3 data=20
dup key=3 100
key=4 data=6
key=5 data=5
dup key=5 99
iterating...
key=1 data=10
key=1 data=315
key=1 data=305
key=2 data=11
key=2 data=316
key=2 data=309
key=3 data=20
key=3 data=400
key=4 data=6
key=5 data=5
key=5 data=399
```

関連項目:

- “Non-Unique Key and Data Pairs” (*SAS Language Reference: Concepts*)

メソッド:

- “REPLACE メソッド” (57 ページ)

SETCUR メソッド

反復を開始するキー項目を指定します。

適用対象: ハッシュ反復子オブジェクト

構文

rc=*object*.**SETCUR**(KEY: '*keyvalue-1*'<, …KEY: '*keyvalue-n*'>);

引数

rc
メソッドが成功したか失敗したかを示します。

ゼロのリターンコードは成功を表し、ゼロ以外の値は失敗を表します。メソッド呼び出しでリターンコード変数を指定せず、そのメソッドが失敗した場合、該当するエラーメッセージがログに書き込まれます。

オブジェクト
ハッシュ反復子オブジェクト名を指定します。

KEY: '*keyvalue*'
反復の開始キーとしてのキー値を指定します。

詳細

ハッシュ反復子を使うと、ハッシュオブジェクト内のどの項目でも反復を開始できます。SETCUR メソッドは、反復の開始キーを設定します。開始項目の指定には、KEY オプションを使用します。

例: 開始キー項目の指定

次の例では、天文学データを含むデータセットを作成します。最初の項目や最後の項目からではなく、RA= 18 31.6 から反復を開始します。データがハッシュオブジェクト内に読み込まれ、反復を開始するのに SETCUR メソッドが使用されます。*ordered* 引数タグが YES に設定されているため、出力が昇順で並べ替えられています。

```
data work.astro;
input obj $1-4 ra $6-12 dec $14-19;
datalines;
M31 00 42.7 +41 16
M71 19 53.8 +18 47
M51 13 29.9 +47 12
M98 12 13.8 +14 54
M13 16 41.7 +36 28
M39 21 32.2 +48 26
M81 09 55.6 +69 04
M100 12 22.9 +15 49
M41 06 46.0 -20 44
M44 08 40.1 +19 59
M10 16 57.1 -04 06
M57 18 53.6 +33 02
M3 13 42.2 +28 23
M22 18 36.4 -23 54
M23 17 56.8 -19 01
M49 12 29.8 +08 00
M68 12 39.5 -26 45
M17 18 20.8 -16 11
M14 17 37.6 -03 15
M29 20 23.9 +38 32
M34 02 42.0 +42 47
M82 09 55.8 +69 41
M59 12 42.0 +11 39
M74 01 36.7 +15 47
M25 18 31.6 -19 15
;
```

次のコードでは、反復の開始キーが**'18 31.6'**に設定されます。

```
data _null_;
length obj $10;
length ra $10;
length dec $10;
declare hash myhash(hashexp: 4, dataset:"work.astro", ordered:"yes");

declare hiter iter('myhash');
myhash.defineKey('ra');
myhash.defineData('obj', 'ra');
myhash.defineDone();
call missing (ra, obj, dec);
rc = iter.setcur(key: '18 31.6');
```

```
do while (rc = 0);
put obj= ra=;
rc = iter.next();
end;
run;
```

次の行が SAS ログに書き込まれます。

```
obj=M25 ra=18 31.6
obj=M22 ra=18 36.4
obj=M57 ra=18 53.6
obj=M71 ra=19 53.8
obj=M29 ra=20 23.9
obj=M39 ra=21 32.2
```

最初の項目または最後の項目から反復を開始するには、FIRST メソッドまたは LAST メソッドをそれぞれ使用します。

関連項目:

- “Using the Hash Iterator Object ” (*SAS Language Reference: Concepts*)

メソッド:

- “FIRST メソッド” (31 ページ)
- “LAST メソッド” (37 ページ)

演算子:

- “_NEW_ Operator、ハッシュオブジェクトとハッシュ反復子オブジェクト” (38 ページ)

ステートメント:

- “DECLARE ステートメントのハッシュオブジェクトとハッシュ反復子オブジェクト” (12 ページ)

SUM メソッド

ハッシュテーブルから特定のキーの集計値を取得し、その値を DATA ステップ変数に保存します。

適用対象: ハッシュオブジェクト

構文

rc=*object*.**SUM**(SUM: *variable-name*);

必須引数

rc

メソッドが成功したか失敗したかを示します。

ゼロのリターンコードは成功を表し、ゼロ以外の値は失敗を表します。メソッド呼び出しでリターンコード変数を指定せず、そのメソッドが失敗した場合、該当するエラーメッセージがログに書き込まれます。

オブジェクト
ハッシュオブジェクト名を指定します。

SUM: ***variable-name***
特定のキーの現在の集計値を保存する DATA ステップ変数を指定します。

詳細

ハッシュオブジェクトからキーの集計値を取得するには、SUM メソッドを使用します。詳細については、“Maintaining Key Summaries” (*SAS Language Reference: Concepts*) を参照してください。

比較

SUM メソッドでは、1 つのキーにつき 1 つのデータ項目しか存在しないときに、特定のキーの集計値を取得します。SUMDUP メソッドでは、1 つのキーにつき複数のデータ項目が存在するときに、現在のキーの現在のデータ項目について集計値を取得します。

例: 特定のキーのキー集計値の取得

次の例では、2 つのキー K=99 および K=100 を指定し、SUM メソッドを使用して各キーの集計値を取得します。

```
 k = 99;
count = 1;
h.add();
/* key=99 summary is now 1 */
k = 100;
h.add();
/* key=100 summary is now 1 */
k = 99;
h.find();
/* key=99 summary is now 2 */
count = 2;
h.find();
/* key=99 summary is now 4 */
k = 100;
h.find();
/* key=100 summary is now 3 */
h.sum(sum: total);
put 'total for key 100 = ' total;
k = 99;
h.sum(sum:total);
put 'total for key 99 = ' total;
run;
```

最初の PUT ステートメントは k=100 の集計値を出力します。

```
total for key 100 = 3
```

2 番目の PUT ステートメントは k=99 の集計値を出力します。

```
total for key 99 = 4
```

関連項目:

メソッド:

- “ADD メソッド” (7 ページ)
- “FIND メソッド” (26 ページ)
- “CHECK メソッド” (9 ページ)
- “REF メソッド” (50 ページ)
- “SUMDUP メソッド” (66 ページ)

演算子:

- “_NEW_ Operator、ハッシュオブジェクトとハッシュ反復子オブジェクト” (38 ページ)

ステートメント:

- “DECLARE ステートメントのハッシュオブジェクトとハッシュ反復子オブジェクト” (12 ページ)

SUMDUP メソッド

現在のキーの現在のデータ項目について集計値を取得し、その値を DATA ステップ変数に保存します。

適用対象: ハッシュオブジェクト

構文

rc=*object*.**SUMDUP**(SUM: *variable-name*);

引数

rc
メソッドが成功したか失敗したかを示します。

ゼロのリターンコードは成功を表し、ゼロ以外の値は失敗を表します。メソッド呼び出しでリターンコード変数を指定せず、そのメソッドが失敗した場合、対応するエラーメッセージがログに出力されます。

オブジェクト
ハッシュオブジェクト名を指定します。

SUM: ***variable-name***
現在のキーの現在のデータ項目について取得した集計値を保存する DATA ステップ変数を指定します。

詳細

キーに複数のデータ項目がある場合、ハッシュオブジェクトからキーの集計値を取得するには、SUMDUP メソッドを使用します。詳細については、“Maintaining Key Summaries” (*SAS Language Reference: Concepts*)を参照してください。

比較

SUMDUP メソッドでは、1 つのキーにつき複数のデータ項目が存在するときに、現在のキーの現在のデータ項目について集計値を取得します。SUM メソッドでは、1 つのキーにつき 1 つのデータ項目しか存在しないときに、特定のキーの集計値を取得します。

例: 集計値の取得

次の例では、SUMDUP メソッドを使って、現在のデータ項目の集計値を取得します。この例は、HAS_PREV メソッドと FIND_PREV メソッドを使うと、リストの前方にループできることも示しています。FIND_PREV メソッドは、現在のリスト項目については FIND_NEXT メソッドと同様の処理を行います。異なるのは、複数の項目リストで前方に移動する点です。

```
data dup;
length key data 8;
input key data;
cards;
1 10
2 11
1 15
3 20
2 16
2 9
3 100
5 5
1 5
4 6
5 99
;
data _null_;
length r i sum 8;
i = 0;
dcl hash h(dataset:'dup', multidata: 'y', suminc: 'i');
h.definekey('key');
h.definedata('key', 'data');
h.definedone();
call missing (key, data);
i = 1;
do key = 1 to 5;
rc = h.find();
if (rc = 0) then do;
h.has_next(result: r);
do while(r ne 0);
rc = h.find_next();
rc = h.find_prev();
rc = h.find_next();
h.has_next(result: r);
end;
end;
end;
i = 0;
do key = 1 to 5;
rc = h.find();
if (rc = 0) then do;
h.sum(sum: sum);
put key= data= sum=;
h.has_next(result: r);
do while(r ne 0);
rc = h.find_next();
h.sumdup(sum: sum);
put 'dup ' key= data= sum=;
h.has_next(result: r);
```

```
end;
end;
end;
run;
```

次の行が SAS ログに書き込まれます。

```
key=1 data=10 sum=2
dup key=1 data=15 sum=3
dup key=1 data=5 sum=2
key=2 data=11 sum=2
dup key=2 data=16 sum=3
dup key=2 data=9 sum=2
key=3 data=20 sum=2
dup key=3 data=100 sum=2
key=4 data=6 sum=1
key=5 data=5 sum=2
dup key=5 data=99 sum=2
```

10、15 および 5(この順序で保存されているものとします)の 3 つのデータ値を持つキー 1 を例にとって、この処理を見てみます。

```
key=1 data=10 sum=2
dup key=1 data=15 sum=3
dup key=1 data=5 sum=2
```

ループのデータリストを移動中、最初の FIND メソッド呼び出しで 10 のキー集計値が 1 に設定されます。最初の FIND_NEXT メソッド呼び出しで 5 のキー集計値が 1 に設定されます。次の FIND_PREV メソッド呼び出しでは、データ値 10 に戻ってキー集計値が 2 に増分されます。さらに、最後の FIND_NEXT メソッド呼び出しで、5 のキー集計値が 2 に設定されます。ループの次の反復では、15 のキー集計値が 1 に設定され、5 のキー集計値が 3 に設定されます(リストで 5 が 15 よりも前に保存されているため)。さらに、15 のキー集計値が 2 に増分されます。このキー 1 の処理結果の出力は、出力 5.10 のとおりです。

この例では、リストに以前のエントリが存在していることがわかっているため、FIND_PREV メソッドを呼び出す前に HAS_PREV メソッドを呼び出していません。存在しなかった場合は、ループに入りません。

この例では、すべてのデータが同じキーを持つため、特定キーの複数のデータ項目については順序が一定しないことがわかります。キーを使って並べ替えることはできません。リスト内の順序(10, 5, 15)は、項目が追加された順番と一致しません。

このことから、重複処理によっては特殊なメソッドが必要なこともわかります(この場合、SUMDUP メソッドが SUM メソッドと同様の処理を、現在のリスト項目のキー集計値について行います)。

関連項目:

- “Non-Unique Key and Data Pairs” (*SAS Language Reference: Concepts*)

メソッド:

- “SUM メソッド” (64 ページ)

3 章

Java オブジェクトのディクショナリ

カテゴリ別の Java オブジェクトメソッド

Java オブジェクトメソッドには 5 つのカテゴリがあります。

表 3.1 *カテゴリ別の Java オブジェクトメソッド*

カテゴリ	説明
削除	Java オブジェクトを削除できます。
例外	例外情報を収集して例外をクリアできます。
フィールド参照	Java オブジェクトの静的なインスタンスフィールドまたは静的でないインスタンスフィールドの値を返したり設定したりできます。
メソッド参照	静的な Java メソッドと静的でない Java メソッドにアクセスできます。
出力	出力先に Java 出力を即座に送信できます。

次の表に、Java オブジェクトメソッドの概要を示します。詳細については、各メソッドの辞書エントリを参照してください。

カテゴリ	言語要素	説明
削除	DELETE メソッド (p. 77)	Java オブジェクトを削除します。
出力	FLUSHJAVAOUTPUT メソッド (p. 82)	出力先に Java 出力を送信するように指定します。
フィールド参照	GET*type*FIELD メソッド (p. 84)	Java オブジェクトの静的でないフィールドの値を返します。
	GETSTATIC*type*FIELD メソッド (p. 86)	Java オブジェクトの静的なフィールドの値を返します。
	SET*type*FIELD メソッド (p. 89)	Java オブジェクトの静的でないフィールドの値を変更します。
	SETSTATIC*type*FIELD メソッド (p. 91)	Java オブジェクトの静的フィールドの値を変更します。
メソッド参照	CALL*type*METHOD メソッド (p. 70)	静的でない Java メソッドから Java オブジェクトのインスタンスメソッドを呼び出します。
	CALLSTATIC*type*METHOD メソッド (p. 73)	静的な Java メソッドから Java オブジェクトのインスタンスメソッドを呼び出します。
例外	EXCEPTIONCHECK メソッド (p. 78)	メソッド呼び出し時に例外が発生したかどうかを確認します。
	EXCEPTIONCLEAR メソッド (p. 79)	現在、発生している例外をクリアします。
	EXCEPTIONDESCRIBE メソッド (p. 81)	デバッグログのオン/オフを切り替え、例外情報を出力します。

ディクショナリ

CALL*type*METHOD メソッド

静的でない Java メソッドから Java オブジェクトのインスタンスメソッドを呼び出します。

カテゴリ: メソッド参照

適用対象: Java オブジェクト

構文

object.**CALL***type***METHOD** ("*method-name*", <*method-argument-1*, …*method-argument-n*>, <*return-value*>);

引数

オブジェクト
Java オブジェクト名を指定します。

type
静的でない Java メソッドの結果型を指定します。型には次のいずれかの値を指定できます。

BOOLEAN
結果型に BOOLEAN を指定します。

BYTE
結果型に BYTE を指定します。

CHAR
結果型に CHAR を指定します。

DOUBLE
結果型に DOUBLE を指定します。

FLOAT
結果型に FLOAT を指定します。

INT
結果型に INT を指定します。

LONG
結果型に LONG を指定します。

SHORT
結果型に SHORT を指定します。

STRING
結果型に STRING を指定します。

VOID
結果型に VOID を指定します。

参照項目 “Type Issues” (*SAS Language Reference: Concepts*)

method-name
静的でない Java メソッドの名前を指定します。

要件 メソッド名は一重引用符または二重引用符で囲む必要があります。

method-argument
メソッドに渡すパラメータを指定します。

return-value
メソッドで値を返す場合の戻り値を指定します。

詳細

Java オブジェクトをインスタンス化したら、CALL*type*METHOD メソッドを使用した Java オブジェクトでのメソッド呼び出しによって、静的でない Java メソッドにアクセスできます。

注: *type* 引数は Java のデータ型を表します。Java のデータ型と SAS のデータ型の関連性の詳細については、“Type Issues” (*SAS Language Reference: Concepts*) を参照してください。

比較

静的でない Java メソッドには CALL*type*METHOD メソッドを使用します。Java メソッドが静的な場合は、CALLSTATIC*type*METHOD メソッドを使用します。

例: フィールドの値の設定と取得

次の例では、3 つの静的でないフィールドを含む単純なクラスを作成します。Java オブジェクト j がインスタンス化され、CALL*type*FIELD メソッドを使用してフィールドの値が設定および取得されます。

```
/* Java code */
import java.util.*;
import java.lang.*;
public class ttest
{
public int i;
public double d;
public string s;
public int im()
{
return i;
}
public String sm()
{
return s;
}
public double dm()
{
return d;
}
}
```

```
/* DATA step code */
data _null_;
dcl javaobj j("ttest");
length val 8;
length str $20;
j.setIntField("i", 100);
j.setDoubleField("d", 3.14159);
j.setStringField("s", "abc");
j.callIntMethod("im", val);
 put val=;
 j.callDoubleMethod("dm", val);
 put val=;
 j.callStringMethod("sm", str);
 put str=;
run;
```

次の行が SAS ログに書き込まれます。

```
val=100
val=3.14159
str=abc
```

関連項目:

メソッド:

- “CALLSTATICtypeMETHOD メソッド” (73 ページ)

CALLSTATIC*type*METHOD メソッド

静的な Java メソッドから Java オブジェクトのインスタンスメソッドを呼び出します。

カテゴリ: メソッド参照

適用対象: Java オブジェクト

構文

object.**CALLSTATIC***type***METHOD** ("*method-name*", <*method-argument-1*, …*method-argument-n*>, <*return-value*>);

引数

オブジェクト
Java オブジェクト名を指定します。

type
静的な Java メソッドの結果型を指定します。型には次のいずれかの値を指定できます。

BOOLEAN
結果型に BOOLEAN を指定します。

BYTE
結果型に BYTE を指定します。

CHAR
結果型に CHAR を指定します。

DOUBLE
結果型に DOUBLE を指定します。

FLOAT
結果型に FLOAT を指定します。

INT
結果型に INT を指定します。

LONG
結果型に LONG を指定します。

SHORT
結果型に SHORT を指定します。

STRING
結果型に STRING を指定します。

VOID
結果型に VOID を指定します。

参照項目 “Type Issues” (*SAS Language Reference: Concepts*)

method-name
静的な Java メソッド名を指定します。

要件 メソッド名は一重引用符または二重引用符で囲む必要があります。

method-argument
メソッドに渡すパラメータを指定します。

return-value
メソッドで値を返す場合の戻り値を指定します。

詳細

Java オブジェクトをインスタンス化したら、CALLSTATIC*type*METHOD メソッドを使用した Java オブジェクトでのメソッド呼び出しによって、静的な Java メソッドにアクセスできます。

注: *type* 引数は Java のデータ型を表します。Java のデータ型と SAS のデータ型の関連性の詳細については、“Type Issues” (*SAS Language Reference: Concepts*) を参照してください。

比較

静的な Java メソッドには CALLSTATIC*type*METHOD メソッドを使用します。Java メソッドが静的でない場合は、CALL*type*METHOD メソッドを使用します。

例: 静的なフィールドの設定と取得

次の例では、3 つの静的なフィールドを含む単純なクラスを作成します。Java オブジェクト j がインスタンス化され、CALLSTATIC*type*FIELD メソッドを使用してフィールドの値が設定および取得されます。

```
/* Java code */
import java.util.*;
import java.lang.*;
public class ttestc
{
public static double d;
public static double dm()
{
return d;
}
```

```
/* DATA step code */
data x;
declare javaobj j("ttestc");
length d 8;
j.SetStaticDoubleField("d", 3.14159);
j.callStaticDoubleMethod("dm", d);
 put d=;
run;
```

次の行が SAS ログに書き込まれます。

```
d=3.14159
```

関連項目:

メソッド:

- “CALLtypeMETHOD メソッド” (70 ページ)

DECLARE ステートメント、Java オブジェクト

Java オブジェクトを宣言します。Java オブジェクトのインスタンスを作成し、データを初期化します。

別名: DCL

構文

形式 1: **DECLARE JAVAOBJ** *object-reference*;

形式 2: **DECLARE JAVAOBJ** *object-reference* ("*java-class*", <*argument-1*, … *argument-n*>);

引数

object-reference
Java オブジェクトのオブジェクト参照名を指定します。

java-class
インスタンス化する Java クラスの名前を指定します。

要件 Java クラス名は二重引用符または一重引用符で囲む必要があります。

Java パッケージのパスを指定する場合、パスにはピリオド(.)ではなくフォワードスラッシュ(/)を使用する必要があります。たとえば、*"java.util.Hashtable"*は正しいクラス名ではありません。正しいクラス名は*"java/util/Hashtable"*です。

引数
Java オブジェクトのインスタンスの作成に使用する情報を指定します。*argument* の有効な値は Java オブジェクトによって異なります。

参照項目 “DECLARE ステートメントを使用した Java オブジェクトのインスタンス生成(フォーム 2)” (76 ページ)

詳細

基本

SAS プログラムで DATA ステップコンポーネントオブジェクトを使用するには、オブジェクトを宣言して作成(インスタンス化)する必要があります。DATA ステップコンポーネントインターフェイスは、DATA ステップ内から定義済みのコンポーネントオブジェクトにアクセスするメカニズムを提供します。

詳細については、“Using DATA Step Component Objects” (*SAS Language Reference: Concepts*)を参照してください。

Java オブジェクトの宣言(フォーム 1)

DECLARE ステートメントを使用して Java オブジェクトを宣言します。

```
declare javaobj j;
```

DECLARE ステートメントは、オブジェクト参照 J が Java オブジェクトであることを SAS に示します。

新しい Java オブジェクトを宣言した後に、_NEW_演算子を使用してオブジェクトをインスタンス化します。たとえば、次のコード行では、_NEW_演算子で Java オブジェクトが作成され、オブジェクト参照 J に割り当てられます。

```
j = _new_ javaobj("somejavaclass");
```

DECLARE ステートメントを使用した Java オブジェクトのインスタンス生成(フォーム 2)

DECLARE ステートメントと_NEW_演算子を使用して Java オブジェクトを宣言し、インスタンス化する 2 ステップのプロセスの代わりに、DECLARE ステートメントを使用して、Java オブジェクトを 1 つのステップで宣言し、インスタンス化することができます。たとえば、次のコード行では、DECLARE ステートメントで Java オブジェクトが宣言およびインスタンス化され、Java オブジェクトがオブジェクト参照 J に割り当てられます。

```
declare javaobj j("somejavaclass");
```

前述のコード行は、次のコードを使用するのと同等です。

```
declare javaobj j;
j = _new_ javaobj("somejavaclass");
```

*コンストラクタ*は、コンポーネントオブジェクトをインスタンス化し、コンポーネントオブジェクトデータを初期化するために使用できるメソッドです。たとえば、次のコード行では、DECLARE ステートメントで Java オブジェクトが宣言およびインスタンス化され、Java オブジェクトがオブジェクト参照 J に割り当てられます。Java オブジェクトコンストラクタで唯一必要な引数は、インスタンス化する Java クラスの名前です。その他の引数はすべて Java クラス自体のコンストラクタ引数です。次の例では、Java クラス名 **testjavaclass** がコンストラクタで、値 **100** と **.8** はコンストラクタ引数です。

```
declare javaobj j("testjavaclass", 100, .8);
```

比較

Java オブジェクトを宣言し、インスタンスを作成するには、DECLARE ステートメントと_NEW_演算子を使用するか、DECLARE ステートメントのみを使用します。

例

例 1: DECLARE ステートメントと_NEW_演算子を使用した Java オブジェクトの宣言とインスタンス化

次の例では、単純な Java クラスが作成されます。DECLARE ステートメントと_NEW_演算子を使用して、このクラスのインスタンスを作成します。

```
/* Java code */
import java.util.*;
import java.lang.*;
public class simpleclass
{
   public int i;
   public double d;
}

/* DATA step code
data _null_;
```

```
   declare javaobj myjo;
   myjo = _new_ javaobj("simpleclass");
run;
```

例 2: DECLARE ステートメントを使用した Java オブジェクトの作成とインスタンス化

次の例では、ハッシュテーブルの Java クラスが作成されます。DECLARE ステートメントを使用して、容量および負荷係数を指定し、このクラスのインスタンスを作成します。この例では、DATA ステップの唯一の数値型が Java のデータ型 DOUBLE と同等であるため、ラッパークラス **mhash** が必要です。

```
/* Java code */
import java.util.*;
public class mhash extends Hashtable;
{
   mhash (double size, double load)
      {
         super ((int)size, (float)load);
      }
}

/* DATA step code */
data _null_;
   declare javaobj h("mhash", 100, .8);
run;
```

関連項目:

- “Using DATA Step Component Objects” (*SAS Language Reference: Concepts*)

演算子:

- “_NEW_演算子の Java オブジェクト” (88 ページ)

DELETE メソッド

Java オブジェクトを削除します。

カテゴリ: 削除

適用対象: Java オブジェクト

構文

object.**DELETE**( );

引数

オブジェクト

Java オブジェクト名を指定します。

詳細

DATA ステップのコンポーネントオブジェクトは、DATA ステップの終了時に自動的に削除されます。他の Java オブジェクトコンストラクタでオブジェクト参照変数を再利用する場合は、DELETE メソッドで Java オブジェクトを削除する必要があります。

削除した Java オブジェクトを使用しようとすると、エラーがログに書き込まれます。

EXCEPTIONCHECK メソッド

メソッド呼び出し時に例外が発生したかどうかを確認します。

カテゴリ: 例外

適用対象: Java オブジェクト

構文

object.**EXCEPTIONCHECK**(*status*);

引数

オブジェクト
Java オブジェクト名を指定します。

ステータス
返される例外ステータスを指定します。

ヒント Java から返されるステータス値のデータ型は DOUBLE です。これは SAS の数値データ値に対応します。

詳細

Java 例外は EXCEPTIONCHECK メソッド、EXCEPTIONCLEAR メソッドおよび EXCEPTIONDESCRIBE メソッドを介して処理されます。

メソッド呼び出し時に例外が発生したかどうかを確認するには、EXCEPTIONCHECK メソッドを使います。例外が発生する可能性のある Java メソッドを呼び出した後は、毎回 EXCEPTIONCHECK メソッドを呼び出すのが理想的です。

例: 例外の確認

次の例では、例外が発生するメソッドが Java クラスに含まれています。DATA ステップはそのメソッドを呼び出し、例外の有無を確認します。

```
/* Java code */
public class a
{
public void m() throws NullPointerException
{
throw new NullPointerException();
}
}

/* DATA step code */
data _null_;
length e 8;
```

```
dcl javaobj j('a');
rc = j.callvoidmethod('m');
/* Check for exception. Value is returned in variable 'e' */
rc = j.exceptioncheck(e);
 if (e) then
put 'exception';
else
put 'no exception';
run;
```

次の行が SAS ログに書き込まれます。

```
exception
```

関連項目:

メソッド:

- “EXCEPTIONCLEAR メソッド” (79 ページ)
- “EXCEPTIONDESCRIBE メソッド” (81 ページ)

EXCEPTIONCLEAR メソッド

現在、発生している例外をクリアします。

カテゴリ: 例外

適用対象: Java オブジェクト

構文

object.**EXCEPTIONCLEAR**( );

引数

オブジェクト
Java オブジェクト名を指定します。

詳細

Java 例外は EXCEPTIONCHECK メソッド、EXCEPTIONCLEAR メソッドおよび EXCEPTIONDESCRIBE メソッドを介して処理されます。

例外が発生するメソッドを呼び出す場合は、呼び出し後に例外の有無を確認することをお勧めします。例外が発生した場合は、適切な操作を実行してから、EXCEPTIONCLEAR メソッドを使って例外をクリアします。

例外が発生していない場合、このメソッドには何の効果もありません。

例

例 1: 例外の確認とクリア

次の例では、例外が発生するメソッドが Java クラスに含まれています。DATA ステップでこのメソッドを呼び出した後、例外をクリアします。

```
/* Java code */
public class a
{
public void m() throws NullPointerException
{
throw new NullPointerException();
}
}
```

```
/* DATA step code */
data _null_;
length e 8;
dcl javaobj j('a');
rc = j.callvoidmethod('m');
/* Check for exception. Value is returned in variable 'e' */
rc = j.exceptioncheck(e);
if (e) then
put 'exception';
else
put 'no exception';
/* Clear the exception and check it again */
rc = j.exceptionclear( );
 rc = j.exceptioncheck(e);
if (e) then
put 'exception';
else
put 'no exception';
run;
```

次の行が SAS ログに書き込まれます。

```
exception
no exception
```

例 2: 外部ファイル読み込み時の例外のチェック

次の例では、Java IO クラスを使用して DATA ステップから外部ファイルを読み込みます。この Java コードは **DataInputStream** のラッパークラスを作成し、**FileInputStream** をコンストラクタに渡せるようにします。コンストラクタが実際に取得するのは、**FileInputStream** の親である **InputStream** です。現在のメソッドルックアップはスーパークラスルックアップを実行できるほど堅牢でないため、ラッパーが必要です。

```
/* Java code */
public class myDataInputStream extends java.io.DataInputStream
{
myDataInputStream(java.io.FileInputStream fi)
{
super(fi);
}
}
```

作成したラッパークラスを使って外部ファイルの **DataInputStream** を作成し、ファイルの終端に達するまでファイルを読み込みます。EXCEPTIONCHECK メソッドを使って **readInt** メソッドでの **EOFException** 発生を判定し、入力ループを終了させることができます。

```
/* DATA step code */
data _null_;
```

```
length d e 8;
dcl javaobj f("java/io/File", "c:\temp\binint.txt");
dcl javaobj fi("java/io/FileInputStream", f);
dcl javaobj di("myDataInputStream", fi);
do while(1);
di.callIntMethod("readInt", d);
di.ExceptionCheck(e);
if (e) then
leave;
else
put d=;
end;
run;
```

関連項目:

メソッド:

- “EXCEPTIONCHECK メソッド” (78 ページ)
- “EXCEPTIONDESCRIBE メソッド” (81 ページ)

EXCEPTIONDESCRIBE メソッド

デバッグログのオン/オフを切り替え、例外情報を出力します。

カテゴリ: 例外

適用対象: Java オブジェクト

構文

object.**EXCEPTIONDESCRIBE**(*status*);

引数

オブジェクト
Java オブジェクト名を指定します。

ステータス
デバッグログのオン/オフを指定します。`status` 引数は次のいずれかの値です。

0
デバッグログをオフに指定します。

1
デバッグログをオンに指定します。

デフォルト 0(オフ)

ヒント Java から返されるステータス値のデータ型は DOUBLE です。これは SAS の数値データ値に対応します。

詳細

EXCEPTIONDESCRIBE メソッドは、例外のデバッグログのオンとオフを切り替えます。例外のデバッグログがオンの場合、例外情報が JVM の標準出力に出力されます。

注: デフォルトでは、JVM の標準出力は SAS ログにリダイレクトされます。

例: 例外情報の標準出力への出力

次の例では、例外情報は標準出力に出力されます。

```
/* Java code */
public class a
{
public void m() throws NullPointerException
{
throw new NullPointerException();
}
}
```

```
/* DATA step code */
data _null_;
length e 8;
dcl javaobj j('a');
j.exceptiondescribe(1);
 rc = j.callvoidmethod('m');
run;
```

次の行が SAS ログに書き込まれます。

```
java.lang.NullPointerException
at a.m(a.java:5)
```

関連項目:

メソッド:

- “EXCEPTIONCHECK メソッド” (78 ページ)
- “EXCEPTIONCLEAR メソッド” (79 ページ)

FLUSHJAVAOUTPUT メソッド

出力先に Java 出力を送信するように指定します。

カテゴリ: 出力

適用対象: Java オブジェクト

構文

object.**FLUSHJAVAOUTPUT**( );

引数

オブジェクト
Java オブジェクト名を指定します。

詳細

SAS ログに対する Java 出力は、DATA ステップの終了時にフラッシュされます。FLUSHJAVAOUTPUT メソッドを使用した場合、Java 出力は DATA ステップの実行中に発行されたすべての出力の後に表示されます。

例: Java 出力の表示

次の例では、DATA ステップの完了後に"In Java class"行が書き込まれます。

```
/* Java code */
public class p
{
void p()
{
System.out.println("In Java class");
}
}
```

```
/* DATA step code */
data _null_;
dcl javaobj j('p');
do i = 1 to 3;
j.callVoidMethod('p');
put 'In DATA Step';
end;
run;
```

次の行が SAS ログに書き込まれます。

```
In DATA Step
In DATA Step
In DATA Step
In Java class
In Java class
In Java class
```

FLUSHJAVAOUTPUT メソッドを使用した場合、Java 出力が実行順に SAS ログに書き込まれます。

```
/* DATA step code */
data _null_;
dcl javaobj j('p');
do i = 1 to 3;
j.callVoidMethod('p');
j.flushJavaOutput();
 put 'In DATA Step';
end;
run;
```

次の行が SAS ログに書き込まれます。

```
In Java class
In DATA Step
In Java class
```

```
In DATA Step
In Java class
In DATA Step
```

関連項目:

“Java Standard Output” (*SAS Language Reference: Concepts*)

GET*type*FIELD メソッド

Java オブジェクトの静的でないフィールドの値を返します。

カテゴリ: フィールド参照

適用対象: Java オブジェクト

構文

object.**GET***type*FIELD("*field-name*", *value*);

引数

オブジェクト
Java オブジェクト名を指定します。

type
Java フィールドの型を指定します。型には次のいずれかの値を指定できます。

BOOLEAN
フィールドの型に BOOLEAN を指定します。

BYTE
フィールドの型に BYTE を指定します。

CHAR
フィールドの型に CHAR を指定します。

DOUBLE
フィールドの型に DOUBLE を指定します。

FLOAT
フィールドの型に FLOAT を指定します。

INT
フィールドの型に INT を指定します。

LONG
フィールドの型に LONG を指定します。

SHORT
フィールドの型に SHORT を指定します。

STRING
フィールドの型に STRING を指定します。

参照項目 “Type Issues” (*SAS Language Reference: Concepts*)

field-name
Java フィールド名を指定します。

要件 フィールド名は一重引用符または二重引用符で囲む必要があります。

value
返されたフィールド値を受け取るする変数名を指定します。

詳細

Java オブジェクトをインスタンス化したら、その Java オブジェクトのメソッド呼び出しを使用し、パブリックフィールドにアクセスして編集できます。GET*type*FIELD メソッドでは、静的でないフィールドにアクセスできます。

注: type 引数は Java のデータ型を表します。Java のデータ型と SAS のデータ型の関連性の詳細については、“Type Issues” (*SAS Language Reference: Concepts*)を参照してください。

比較

GET*type*FIELD メソッドは、Java オブジェクトの静的でないフィールドの値を返します。静的なフィールドの値を返すには、GETSTATIC*type*FIELD メソッドを使用します。

例: 静的でないフィールドの値の取得

次の例では、3 つの静的でないフィールドを含む単純なクラスを作成します。Java オブジェクト j がインスタンス化され、GET*type*FIELD メソッドを使用してフィールドの値が変更および取得されます。

```
/* Java code */
import java.util.*;
import java.lang.*;
public class ttest
{
public int i;
public double d;
public string s;
}
}
```

```
/* DATA step code */
data _null_;
dcl javaobj j("ttest");
length val 8;
length str $20;
j.setIntField("i", 100);
j.setDoubleField("d", 3.14159);
j.setStringField("s", "abc");
j.getIntField("i", val);
put val=;
j.getDoubleField("d", val);
 put val=;
 j.getStringField("s", str);
 put str=;
run;
```

次の行が SAS ログに書き込まれます。

```
val=100
val=3.14159
str=abc
```

関連項目:

メソッド:

- “GETSTATICtypeFIELD メソッド” (86 ページ)
- “SETtypeFIELD メソッド” (89 ページ)

GETSTATIC*type*FIELD メソッド

Java オブジェクトの静的なフィールドの値を返します。

カテゴリ: フィールド参照

適用対象: Java オブジェクト

構文

object.**GETSTATIC***type***FIELD**("*field-name*", *value*);

引数

オブジェクト
Java オブジェクト名を指定します。

type
Java フィールドの型を指定します。型には次のいずれかの値を指定できます。

BOOLEAN
フィールドの型に BOOLEAN を指定します。

BYTE
フィールドの型に BYTE を指定します。

CHAR
フィールドの型に CHAR を指定します。

DOUBLE
フィールドの型に DOUBLE を指定します。

FLOAT
フィールドの型に FLOAT を指定します。

INT
フィールドの型に INT を指定します。

LONG
フィールドの型に LONG を指定します。

SHORT
フィールドの型に SHORT を指定します。

STRING
フィールドの型に STRING を指定します。

参照項目 “Type Issues” (*SAS Language Reference: Concepts*)

field-name
Java フィールド名を指定します。

要件 フィールド名は一重引用符または二重引用符で囲む必要があります。

value
返されたフィールド値を受け取るする変数名を指定します。

詳細

Java オブジェクトをインスタンス化したら、その Java オブジェクトのメソッド呼び出しを使用し、パブリックフィールドにアクセスして編集できます。GETSTATIC*type*FIELD メソッドでは、静的なフィールドにアクセスできます。

注: type 引数は Java のデータ型を表します。Java のデータ型と SAS のデータ型の関連性の詳細については、"Type Issues" (*SAS Language Reference: Concepts*)を参照してください。

比較

GETSTATIC*type*FIELD メソッドは、Java オブジェクトの静的なフィールドの値を返します。静的でないフィールドの値を返すには、GET*type*FIELD メソッドを使用します。

例: 静的なフィールドの値の取得

次の例では、3 つの静的なフィールドを含む単純なクラスを作成します。Java オブジェクト j がインスタンス化され、GETSTATIC*type*FIELD メソッドを使用してフィールドの値が設定および取得されます。

```
/* Java code */
import java.util.*;
import java.lang.*;
public class ttest
{
public int i;
public double d;
public string s;
}
}
```

```
/* DATA step code */
data _null_;
dcl javaobj j("ttest");
length val 8;
length str $20;
j.setStaticIntField("i", 100);
j.setStaticDoubleField("d", 3.14159);
j.setStaticStringField("s", "abc");
j.getStaticIntField("i", val);
put val=;
j.getStaticDoubleField("d", val);
 put val=;
 j.getStaticStringField("s", str);
 put str=;
run;
```

次の行が SAS ログに書き込まれます。

```
val=100
val=3.14159
str=abc
```

関連項目:

メソッド:

- “GETtypeFIELD メソッド” (84 ページ)
- “SETSTATICtypeFIELD メソッド” (91 ページ)

_NEW_演算子の Java オブジェクト

Java オブジェクトのインスタンスを作成します。

該当要素: DATA ステップ

適用対象: Java オブジェクト

構文

object-reference = **_NEW_ JAVAOBJ** ("*java-class*", <*argument-1*, …*argument-n*>);

引数

object-reference
Java オブジェクトのオブジェクト参照名を指定します。

java-class
インスタンス化する Java クラスの名前を指定します。

要件 Java クラス名は一重引用符または二重引用符で囲む必要があります。

引数
Java オブジェクトのインスタンスの作成に使用する情報を指定します。*argument* の有効な値は Java オブジェクトによって異なります。

詳細

SAS プログラムで DATA ステップコンポーネントオブジェクトを使用するには、オブジェクトを宣言して作成(インスタンス化)する必要があります。DATA ステップコンポーネントインターフェイスは、DATA ステップ内から定義済みのコンポーネントオブジェクトにアクセスするメカニズムを提供します。

_NEW_演算子を使用して Java オブジェクトをインスタンス化する場合は、最初に DECLARE ステートメントを使用して Java オブジェクトを宣言する必要があります。たとえば、次のコード行では、DECLARE ステートメントは、オブジェクト参照 J が Java オブジェクトであることを SAS に示します。_NEW_演算子で Java オブジェクトが作成され、オブジェクト参照 J に割り当てられます。

```
declare javaobj j;
j = _new_ javaobj("somejavaclass" );
```

注: DECLARE ステートメントを使用して Java オブジェクトの宣言とインスタンス化を 1 つのステップで実行できます。

コンストラクタは、コンポーネントオブジェクトをインスタンス化し、コンポーネントオブジェクトデータを初期化するために使用するメソッドです。たとえば、次のコード行では、_NEW_演算子で Java オブジェクトがインスタンス化され、オブジェクト参照 J に割り当てられます。Java オブジェクトコンストラクタで唯一必要な引数は、インスタンス化する

Java クラスの名前です。その他の引数はすべて Java クラス自体のコンストラクタ引数です。次の例では、Java クラス名 **testjavaclass** がコンストラクタで、値 **100** と **.8** はコンストラクタ引数です。

```
declare javaobj j;
j = _new_ javaobj("testjavaclass", 100, .8);
```

定義済みの DATA ステップコンポーネントオブジェクトおよびコンストラクタの詳細については、“Using DATA Step Component Objects” (*SAS Language Reference: Concepts*) を参照してください。

比較

Java オブジェクトを宣言し、インスタンスを作成するには、DECLARE ステートメントと _NEW_ 演算子を使用するか、DECLARE ステートメントのみを使用します。

例: _NEW_ 演算子を使用した Java クラスのインスタンス化と初期化

次の例では、ハッシュテーブルの Java クラスが作成されます。_NEW_ 演算子を使用して、容量および負荷係数を指定し、このクラスのインスタンスを作成します。この例では、DATA ステップの唯一の数値型が Java のデータ型 DOUBLE と同等であるため、ラッパークラス **mhash** が必要です。

```
/* Java code */
import java.util.*;
public class mhash extends Hashtable;
{
mhash (double size, double load)
{
super ((int)size, (float)load);
}
}

/* DATA step code */
data _null_;
declare javaobj h;
 h = _new_ javaobj("mhash", 100, .8);
run;
```

関連項目:

- “Using DATA Step Component Objects” (*SAS Language Reference: Concepts*)

ステートメント:

- “DECLARE ステートメント、Java オブジェクト” (75 ページ)

SET*type*FIELD メソッド

Java オブジェクトの静的でないフィールドの値を変更します。

カテゴリ: フィールド参照

適用対象: Java オブジェクト

構文

object.**SET***type***FIELD**("*field-name*", *value*);

引数

オブジェクト
Java オブジェクト名を指定します。

type
Java フィールドの型を指定します。型には次のいずれかの値を指定できます。

BOOLEAN
フィールドの型に BOOLEAN を指定します。

BYTE
フィールドの型に BYTE を指定します。

CHAR
フィールドの型に CHAR を指定します。

DOUBLE
フィールドの型に DOUBLE を指定します。

FLOAT
フィールドの型に FLOAT を指定します。

INT
フィールドの型に INT を指定します。

LONG
フィールドの型に LONG を指定します。

SHORT
フィールドの型に SHORT を指定します。

STRING
フィールドの型に STRING を指定します。

参照項目 “Type Issues” (*SAS Language Reference: Concepts*)

field-name
Java フィールド名を指定します。

要件 フィールド名は一重引用符または二重引用符で囲む必要があります。

value
フィールドの値を指定します。

詳細

Java オブジェクトをインスタンス化したら、その Java オブジェクトのメソッド呼び出しを使用し、パブリックフィールドにアクセスして編集できます。SET*type*FIELD メソッドを使うと、静的でないフィールドの値を変更できます。

注: *type* 引数は Java のデータ型を表します。Java のデータ型と SAS のデータ型の関連性の詳細については、“Type Issues” (*SAS Language Reference: Concepts*)を参照してください。

比較

SET*type*FIELD メソッドは、Java オブジェクトの静的でないフィールドの値を変更します。静的フィールドの値を変更するには、SETSTATIC*type*FIELD メソッドを使用します。

例: 静的でないフィールドを持つ Java クラスの作成

次の例では、3 つの静的でないフィールドを含む単純なクラスを作成します。Java オブジェクト j のインスタンス化され、SET*type*FIELD メソッドを使用してフィールド値が設定されてから、フィールド値が取得されます。

```
/* Java code */
import java.util.*;
import java.lang.*;
public class ttest
{
public int i;
public double d;
public string s;
}
}
```

```
/* DATA step code */
data _null_;
dcl javaobj j("ttest");
length val 8;
length str $20;
j.setIntField("i", 100);
 j.setDoubleField("d", 3.14159);
 j.setStringField("s", "abc");
 j.getIntField("i", val);
put val=;
j.getDoubleField("d", val);
put val=;
j.getStringField("s", str);
put str=;
run;
```

次の行が SAS ログに書き込まれます。

```
val=100
val=3.14159
str=abc
```

関連項目:

メソッド:

- “GETtypeFIELD メソッド” (84 ページ)
- “SETSTATICtypeFIELD メソッド” (91 ページ)

SETSTATIC*type*FIELD メソッド

Java オブジェクトの静的フィールドの値を変更します。

カテゴリ: フィールド参照

適用対象: Java オブジェクト

構文

object.**SETSTATIC***type***FIELD**("*field-name*", *value*);

引数

オブジェクト
Java オブジェクト名を指定します。

type
Java フィールドの型を指定します。型には次のいずれかの値を指定できます。

BOOLEAN
フィールドの型に BOOLEAN を指定します。

BYTE
フィールドの型に BYTE を指定します。

CHAR
フィールドの型に CHAR を指定します。

DOUBLE
フィールドの型に DOUBLE を指定します。

FLOAT
フィールドの型に FLOAT を指定します。

INT
フィールドの型に INT を指定します。

LONG
フィールドの型に LONG を指定します。

SHORT
フィールドの型に SHORT を指定します。

STRING
フィールドの型に STRING を指定します。

参照項目 “Type Issues” (*SAS Language Reference: Concepts*)

field-name
Java フィールド名を指定します。

要件 フィールド名は一重引用符または二重引用符で囲む必要があります。

value
フィールドの値を指定します。

詳細

Java オブジェクトをインスタンス化したら、その Java オブジェクトのメソッド呼び出しを使用し、パブリックフィールドにアクセスして編集できます。SETSTATIC*type*FIELD メソッドを使うと、静的フィールドの値を変更できます。

注: type 引数は Java のデータ型を表します。Java のデータ型と SAS のデータ型の関連性の詳細については、“Type Issues” (*SAS Language Reference: Concepts*)を参照してください。

比較

SETSTATIC*type*FIELD メソッドは、Java オブジェクトの静的フィールドの値を変更します。静的でないフィールドの値を変更するには、SET*type*FIELD メソッドを使用します。

例: 静的フィールドを持つ Java クラスの作成

次の例では、3 つの静的なフィールドを含む単純なクラスを作成します。Java オブジェクト j がインスタンス化され、SETSTATIC*type*FIELD メソッドでフィールド値が設定されてから、フィールド値が取得されます。

```
/* Java code */
import java.util.*;
import java.lang.*;
public class ttestc
{
public static double d;
public static double dm()
{
return d;
}
}
```

```
/* DATA step code */
data _null_;
dcl javaobj j("ttest");
length val 8;
length str $20;
j.setStaticIntField("i", 100);
 j.setStaticDoubleField("d", 3.14159);
  j.setStaticStringField("s", "abc");
 j.getStaticIntField("i", val);
put val=;
j.getStaticDoubleField("d", val);
put val=;
j.getStaticStringField("s", str);
put str=;
run;
```

次の行が SAS ログに書き込まれます。

```
val=100
val=3.14159
str=abc
```

関連項目:

メソッド:

- “GETSTATICtypeFIELD メソッド” (86 ページ)
- “SETtypeFIELD メソッド” (89 ページ)

推奨資料

このタイトルに関連した推奨される参考資料のリストを次に示します。

- *SAS データセットオプション: リファレンス*
- *SAS 言語リファレンス: 解説編*
- *SAS ログ機能: 構成とプログラミングリファレンス*
- *SAS ステートメント: リファレンス*

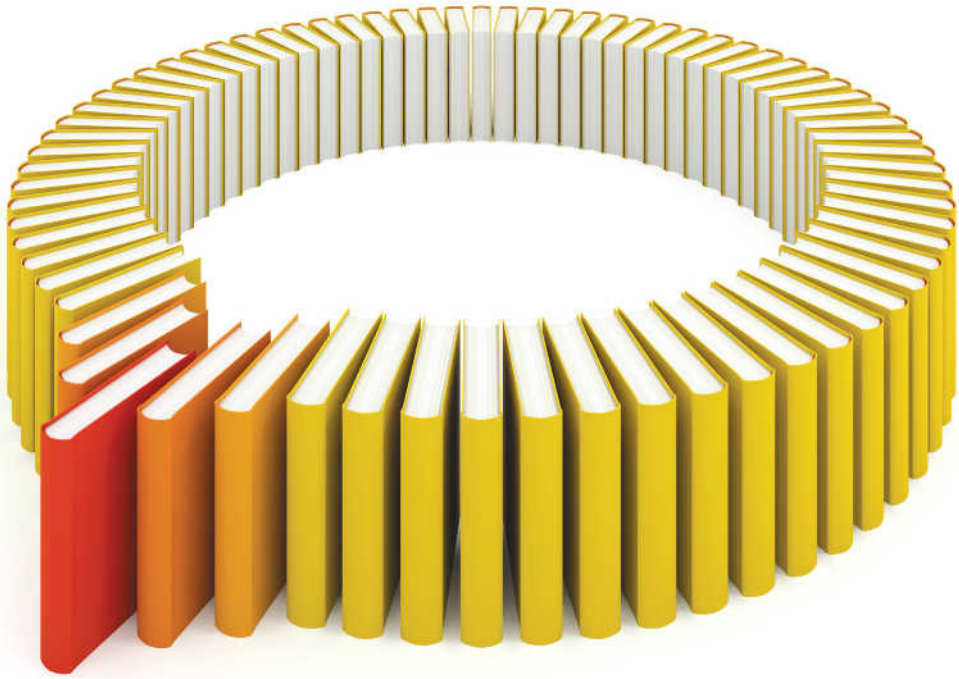
SAS Press から発行されている推奨ドキュメントには次のものがあります。

- *SAS Hash Object Programming Made Easy*

SAS 刊行物の一覧については、sas.com/store/books から入手できます。必要な書籍についての質問は SAS 担当者までお寄せください:

SAS Books
SAS Campus Drive
Cary, NC 27513-2414
電話: 1-800-727-0025
ファクシミリ: 1-919-677-4444
メール: sasbook@sas.com
Web アドレス: sas.com/store/books

キーワード



Gain Greater Insight into Your SAS® Software with SAS Books.

Discover all that you need on your journey to knowledge and empowerment.

support.sas.com/bookstore
for additional books and resources.